Pontifícia Universidade Católica
do Rio de Janeiro

Francisco Sergio de Freitas Filho

Aprendizado com Restrição de Tempo: Problemas de Classificação

Tese de Doutorado

Tese apresentada como requisito parcial para obtenção do grau de Doutor pelo Programa de Pós–graduação em Informática, do Departamento de Informática da PUC-Rio .

Orientador : Prof. Eduardo Sany Laber
Co-orientador: Prof. Marco Serpa Molinaro

Rio de Janeiro
Maio de 2023

Pontifícia Universidade Católica
do Rio de Janeiro

Francisco Sergio de Freitas Filho

Aprendizado com Restrição de Tempo: Problemas de Classificação

Tese apresentada como requisito parcial para obtenção do grau de Doutor pelo Programa de Pós–graduação em Informática da PUC-Rio . Aprovada pela Comissão Examinadora abaixo:

Prof. Eduardo Sany Laber
Orientador
Departamento de Informática – PUC-Rio

Prof. Marco Serpa Molinaro
Co-orientador
Departamento de Informática – PUC-Rio

Prof. Edward Hermann Haeusler
Departamento de Informática – PUC-Rio

Prof. Diego Parente Paiva Mesquita
FGV

Prof. Fábio André Machado Porto
LNCC

Prof. Bianca Zadrozny
IBM Research Brazil

Rio de Janeiro, 24 de Maio de 2023

Todos os direitos reservados. A reprodução, total ou parcial do trabalho, é proibida sem a autorização da universidade, do autor e do orientador.

Francisco Sergio de Freitas Filho

Graduado em Ciência da Computação pela Universidade Federal do Ceará (2017). Mestre em Ciência da Computação pela Universidade Federal do Ceará (2019).

Ficha Catalográfica

Freitas, Francisco Sergio

Aprendizado com Restrição de Tempo: Problemas de Classificação / Francisco Sergio de Freitas Filho; orientador: Eduardo Sany Laber; co-orientador: Marco Serpa Molinaro. – 2023.

120 f: il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2023.

Inclui bibliografia

1. Informática – Teses. 2. Machine Teaching. 3. Aprendizado com Restrição de Tempo. 4. Métodos de Classificação. I. Laber, Eduardo. II. Molinaro, Marco. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

Agradecimentos

A Deus, pela vida e por cada conquista alcançada até aqui.

Ao meu orientador, Eduardo Laber, pela grande ajuda no desenvolvimento deste trabalho e por todas as críticas construtivas, sugestões valiosas e disponibilidade durante esses quatro anos. Ao Professor Marco Molinaro, pelas discussões e contribuições fundamentais para a realização desta pesquisa. Ao Pedro Lazéra, por suas contribuições, em especial pela ajuda com a implementação dos métodos empregados neste trabalho.

Aos colegas da PUC-RIO, que sempre buscaram contribuir de alguma forma positiva. Em especial, aos integrantes dos laboratórios Galgos e Daslab, que sempre estiveram disponíveis para debatermos diversos assuntos de interesses acadêmicos. Agradeço também pelos bons momentos de descontração.

Aos meus pais, Sergio e Valéria, pelo apoio em todos os momentos e pelos esforços em me proporcionarem uma boa educação. Aos meus avós paternos, Mauro e Terezinha, e minha avó materna, Nazaré. Às minhas irmãs, pelo apoio incondicional, e à Danielly, minha namorada, pelo companheirismo em todos os momentos.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Resumo

Freitas, Francisco Sergio; Laber, Eduardo; Molinaro, Marco. **Aprendizado com Restrição de Tempo: Problemas de Classificação**. Rio de Janeiro, 2023. 120p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Com a crescente quantidade de dados sendo gerados e coletados, torna-se mais comum cenários em que se dispõe de dados rotulados em larga escala, mas com recursos computacionais limitados, de modo que não seja possível treinar modelos preditivos utilizando todas as amostras disponíveis. Diante dessa realidade, adotamos o paradigma de Machine Teaching como uma alternativa para obter modelos eficazes utilizando um subconjunto representativo dos dados disponíveis.

Inicialmente, consideramos um problema central da área de *Machine Teaching* que consiste em encontrar o menor conjunto de amostras necessário para obter uma dada hipótese alvo h^*. Adotamos o modelo de ensino *black-box learner* introduzido em (DASGUPTA et al., 2019), em que o ensino é feito interativamente sem qualquer conhecimento sobre o algoritmo do *learner* e sua classe de hipóteses, exceto que ela contém a hipótese alvo h^*. Refinamos alguns resultados existentes para esse modelo e estudamos variantes dele. Em particular, estendemos um resultado de (DASGUPTA et al., 2019) para o cenário mais realista em que h^* pode não estar contido na classe de hipóteses do *learner* e, portanto, o objetivo do teacher é fazer o *learner* convergir para a melhor aproximação disponível de h^*. Também consideramos o cenário com *black-box learners* não adversários e mostramos que podemos obter melhores resultados para o tipo de *learner* que se move para a próxima hipótese de maneira suave, preferindo hipóteses que são mais próximas da hipótese atual.

Em seguida, definimos e abordamos o problema de Aprendizado com Restrição de Tempo considerando um cenário em que temos um enorme conjunto de dados e um limite de tempo para treinar um dado *learner* usando esse conjunto. Propomos o método TCT, um algoritmo para essa tarefa, desenvolvido com base nos princípios de Machine Teaching. Apresentamos um estudo experimental envolvendo 5 diferentes learners e 20 datasets no qual mostramos que TCT supera métodos alternativos considerados. Finalmente, provamos garantias de aproximação para uma versão simplificada do TCT.

Palavras-chave

Machine Teaching; Aprendizado com Restrição de Tempo; Métodos de Classificação.

Abstract

Freitas, Francisco Sergio; Laber, Eduardo (Advisor); Molinaro, Marco (Co-Advisor). **Time Constrained Learning: Classification Problems**. Rio de Janeiro, 2023. 120p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

With the growing amount of data being generated and collected, it becomes increasingly common to have scenarios where there are large-scale labeled data but limited computational resources, making it impossible to train predictive models using all available samples. Faced with this reality, we adopt the Machine Teaching paradigm as an alternative to obtain effective models using a representative subset of available data.

Initially, we consider a central problem of the Machine Teaching area which consists of finding the smallest set of samples necessary to obtain a given target hypothesis h^*. We adopt the black-box learner teaching model introduced in (DASGUPTA et al., 2019), where teaching is done interactively without any knowledge about the learner's algorithm and its hypothesis class, except that it contains the target hypothesis h^*. We refine some existing results for this model and study its variants. In particular, we extend a result from (DASGUPTA et al., 2019) to the more realistic scenario where h^* may not be contained in the learner's hypothesis class, and therefore, the teacher's objective is to make the learner converge to the best available approximation of h^*. We also consider the scenario with non-adversarial black-box learners and show that we can obtain better results for the type of learner that moves to the next hypothesis smoothly, preferring hypotheses that are closer to the current hypothesis.

Next, we address the Time-Constrained Learning problem, considering a scenario where we have a huge dataset and a time limit to train a given learner using this dataset. We propose the TCT method, an algorithm for this task, developed based on Machine Teaching principles. We present an experimental study involving 5 different learners and 20 datasets in which we show that TCT outperforms alternative methods considered. Finally, we prove approximation guarantees for a simplified version of TCT.

Keywords

Machine Teaching; Time Constrained Learning; Classification Methods.

Sumário

Lista de figuras

Lista de tabelas

1
Introdução

Com o aumento ao acesso a ferramentas que permitem gerar e compartilhar informações, a quantidade de dados gerados tem crescido de forma considerável. Um exemplo claro disso é a grande quantidade de informações que podemos encontrar disponíveis na internet: fóruns de discussões, páginas de notícias, fotos e vídeos compartilhados em redes sociais, etc. Em particular, empresas que criam e coletam dados podem extrair valor dessas informações aplicando *business intelligence*. Com isso, também podemos observar a popularização e o aumento no número de aplicações que utilizam aprendizado de máquina em larga escala de dados.

Diante dessa realidade, treinar um modelo pode levar muito tempo e demandar muitos recursos computacionais. Isso porque, em geral, o tempo de treinamento do modelo é proporcional ao tamanho do conjunto de dados. Neste trabalho, estamos interessados em problemas de classificação no cenário em que dispomos de uma quantidade massiva de dados rotulados, mas com recursos limitados, de modo que seja impraticável treinar um modelo de *Machine Learning* utilizando todas as amostras disponíveis.

Um exemplo prático em que isso ocorre se dá quando o orçamento financeiro for um fator limitante para o treinamento do modelo e necessitamos utilizar serviços de aprendizado de máquina baseados em nuvem, como o Azure ML da Microsoft. Nesse caso, é necessário estabelecer um limite de tempo de treinamento para que o processo seja concluído dentro do orçamento disponível. Outro exemplo é o treinamento com *logs* de sistemas, em que os *logs* podem ser excessivamente grandes e demandar mais recursos, como tempo ou uso de memória, do que dispomos. Para esses casos, é necessário tentar garantir que o modelo seja treinado de maneira eficiente e eficaz dentro das nossas limitações.

Uma abordagem possível para contornar esse problema em que dispomos de muitos dados, mas com recursos limitados para treinar utilizandos todos eles, é selecionar um subconjunto de amostras aleatoriamente. Entretanto, a seleção aleatória pode incluir exemplos de treinamento que são redundantes ou pouco informativos, o que pode resultar em um classificador menos eficiente quando comparamos com uma técnica de seleção mais sofisticada.

Em nosso trabalho, exploramos o paradigma de *Machine Teaching* para encontrar as amostras mais relevantes e representativas (o *teaching set*), permitindo que o modelo aprenda de forma mais eficiente. Em linhas gerais, *Machine Teaching* é uma subárea de *Machine Learning* que estuda como

encontrar o menor subconjunto de amostras, de modo que treinar nesse subconjunto seja o suficiente para obter um modelo tão bom quanto seria possível obter treinando em todo o conjunto de exemplos. Conceitos básicos da área de Machine Teaching são discutidos na Seção 2.3.

Em uma primeira etapa desta pesquisa, adotamos o paradigma de *Machine Teaching* em sua forma tradicional, ou seja, com o objetivo de minimizar o tamanho do *teaching set*. Embora seja bastante abordado na literatura, geralmente as garantias de qualidade sobre o tamanho do *teaching set* consideram algumas suposições sobre o algoritmo de aprendizado do classificador ou que o modelo de aprendizado sempre consegue retornar um classificador perfeito. Uma contribuição de nosso trabalho é estender os melhores resultados presentes na literatura para um cenário mais genérico e realista, em que não fazemos suposições sobre o algoritmo de aprendizado e também consideramos o caso em que pode não existir um classificador perfeito. Os resultados obtidos nessa etapa da pesquisa são apresentados no Capítulo 3 e foram publicados em forma de artigo na *International Conference on Machine Learning* (ICML) (CICALESE et al., 2020). A partir desses resultados, pudemos constatar a eficácia de aplicar *Machine Teaching* para obter classificadores precisos utilizando um subconjunto reduzido do dataset, o que é essencial para esse cenário de aprendizado em larga escala e com recursos limitados.

Apesar dos bons resultados obtidos utilizando esse paradigma, pudemos observar que os métodos convencionais de *Machine Teaching* ignoram um fator que pode ser importante em nosso cenário: o tempo que leva para construir o *teaching set*. Por conta disso e pelo potencial de aplicação, resolvemos dar continuidade em nossa pesquisa introduzindo e abordando um cenário prático de Aprendizado com Restrição de Tempo ou, em inglês, *Time-Constrained Learning* (TCL), que pode ser visto como uma formulação alternativa para o problema de *Machine Teaching* (Seção 2.3.1) em que o objetivo é encontrar o *teaching set* para o qual o *learner* retorne o melhor classificador possível e não extrapole o limite de tempo. Novamente, estamos interessados em um cenário mais geral e realista no qual não temos conhecimento sobre o algoritmo de aprendizado, tampouco sobre a existência de um classificador perfeito. Diante desse cenário, não estamos cientes de nenhuma proposta de solução concreta para o problema e propomos um método baseado nos fundamentos de *Machine Teaching*. A motivação, a definição formal para esse problema e as nossas contribuições são apresentadas no Capítulo 4. Destacamos que estes resultados foram recentemente aceitos para publicação em forma de artigo no periódico científico *Pattern Recognition* (FREITAS et al., 2023). Também apresentamos, no Apêndice A, uma contribuição adicional para o problema de

TCL: um *framework* Python que pode ser facilmente instalado e é compatível com modelos de aprendizado fornecidos pela biblioteca scikit-learn.

O restante deste trabalho é dividido como segue. O Capítulo 2 trata de conceitos que vão auxiliar no entendimento do texto: discutimos o paradigma de *Machine Teaching* e apresentamos conceitos da área de teoria do aprendizado estatístico que são importantes para obter as garantias obtidas ao longo deste trabalho. Por fim, no Capítulo 5, encerramos nosso texto com conclusões sobre nosso trabalho e apontamos direcionamentos para pesquisas futuras.

2
Fundamentação Teórica

Neste capítulo, apresentamos alguns conceitos fundamentais para auxiliar o leitor a compreender melhor o texto. Na Seção 2.1, oferecemos uma visão simplificada sobre a tarefa de classificação e definimos alguns termos que serão utilizados ao longo deste trabalho. É importante destacar que assumimos que o leitor possui familiaridade com *Machine Learning*, principalmente no que diz respeito à tarefa de classificação. Na Seção 2.2, abordamos alguns conceitos essenciais da teoria do aprendizado estatístico que são relevantes para a compreensão das garantias obtidas em nossa pesquisa. Caso seja de interesse do leitor uma imersão mais aprofundada nesses assuntos, recomendamos (SHWARTZ; DAVID, 2014) e (ANTHONY; BARTLETT, 1999). Por fim, na Seção 2.3, discutimos o paradigma de *Machine Teaching*, que serviu como base para os métodos propostos em nosso estudo.

2.1
O Problema de Classificação

O problema de classificação, um problema clássico em *machine learning*, é um tipo de tarefa de aprendizado supervisionado e consiste em atribuir a um exemplo um rótulo de classe. O objetivo é construir um classificador capaz de aprender a relação entre as características dos exemplos e as classes correspondentes, a partir de um conjunto de dados de treinamento rotulados, o dataset, e usar esse modelo para classificar novos exemplos. Esse tipo de problema é comum em muitos domínios, incluindo reconhecimento de imagens, diagnóstico médico, detecção de fraudes e análise de sentimentos.

Para ilustrar esse conceito, vamos considerar um exemplo simples de classificação de spam. Nesse caso, o objetivo é construir um classificador que possa distinguir entre mensagens legítimas (não spam) e mensagens indesejadas (spam), com base em um conjunto de atributos ou características de e-mails. Para isso, podemos coletar um conjunto de e-mails rotulados como spam ou não spam e usar esse conjunto para treinar o modelo. Durante o treinamento, o modelo aprende a reconhecer padrões nas mensagens de e-mail que distinguem spam de e-mails legítimos. Esses padrões podem incluir características como palavras ou frases comuns usadas em spam, como "oferta imperdível", bem como a presença de *links* suspeitos ou anexos maliciosos. Uma vez treinado, o classificador pode ser usado para rotular novos e-mails que não foram usados no treinamento.

Existem diferentes algoritmos de *machine learning* que podem ser usados para resolver problemas de classificação, como árvores de decisão, redes neurais, SVM (*Support Vector Machines*), Naive Bayes, entre outros. De forma concisa, um algoritmo de aprendizado consiste em um conjunto de regras e procedimentos que ajudam a construir um classificador capaz de realizar previsões precisas (minimizando a diferença entre as previsões do modelo e as classes reais) a partir dos dados de treinamento. Ao longo do texto, também fazemos o uso do termo "learner" como sinônimo para algoritmo de aprendizado, e utilizamos "hipótese" como sinônimo para classificador.

É importante lembrar que a qualidade das previsões do modelo depende da qualidade dos dados de treinamento. É necessário garantir que o conjunto de dados seja representativo para que o modelo seja capaz de generalizar a partir dos dados de treinamento para fazer previsões precisas em novos dados (abordamos isso mais detalhadamente na Seção 2.2).

A seguir, na Seção 2.1.1 definimos formalmente os conceitos abordados nesta seção.

2.1.1 Notação

- Um exemplo é um vetor x de números reais;
- $\mathcal{X}$ é o domínio de exemplos;
- $\mathcal{Y}$ é um conjunto finito de rótulos possíveis para cada exemplo $x \in \mathcal{X}$;
- Um *dataset* rotulado $D = (X, Y) = ((x_1, y_1), (x_2, y_2), ...(x_m, y_m))$ é uma sequência finita de pares em $\mathcal{X} \times \mathcal{Y}$. Em nosso trabalho, sempre consideramos dados rotulados e, portanto, podemos omitir o uso do termo "rotulado" ao longo do texto.
- Um classificador ou uma hipótese $h : \mathcal{X} \mapsto \mathcal{Y}$ é uma função que mapeia cada exemplo $x \in \mathcal{X}$ a um rótulo $h(x) \in \mathcal{Y}$.
- Um *learner* L é um algoritmo que recebe um *dataset* como entrada e retorna uma hipótese $h \in \mathcal{H}$, em que $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$ é sua classe de hipóteses, ou seja, o conjunto de hipóteses que L consegue gerar;

2.2
Conceitos de Teoria do Aprendizado Estatístico

Nesta seção, tratamos de alguns conceitos de teoria do aprendizado estatístico. Para isso, adotamos a notação adotada em (SHWARTZ; DAVID, 2014). Na Seção 2.2.1, falamos sobre o paradigma ERM e da possibilidade de ocorrer *overfitting*, que definimos na Seção 2.2.2. Em seguida, na Seção 2.2.3, abordamos garantias de aprendizado, sob certas condições, para um número suficientemente grande de amostras.

2.2.1
Empirical Risk Minimization (ERM)

Seja μ uma distribuição de probabilidade qualquer sobre o domínio de exemplos $\mathcal{X}$ e seja $f : \mathcal{X} \mapsto \mathcal{Y}$ uma função qualquer que mapeia exemplos para rótulos. A priori, assumimos que os conjuntos de dados de treinamento são gerados aleatoriamente a partir de μ e rotulados por f. Dessa maneira, entende-se por erro de um classificador como a sua probabilidade de predizer incorretamente o rótulo de um exemplo com respeito à distribuição de probabilidades μ e função f. Mais precisamente, o erro de uma hipótese $h : \mathcal{X} \mapsto \mathcal{Y}$ é definido por:

$$E_{(\mu,f)}(h) = \mathop{\mathbb{P}}_{x\sim\mu}[h(x) \neq f(x)] \tag{2-1}$$

$E_{(\mu,f)}(h)$ também é comumente conhecido por erro de generalização, risco ou erro real de h.

Dado que o *learner* não conhece μ e f, ele não pode calcular diretamente o erro real. No entanto, o *learner* tem uma noção de erro que pode ser facilmente calculada por ele – o erro de treinamento – classificando o conjunto de treino $S = \{(x_1, y_1), ..., (x_m, y_m)\}$:

$$E_S(h) = \frac{|\{i \in [m] : h(x_i) \neq y_i\}|}{m}, \tag{2-2}$$

em que $[m] = \{1, ..., m\}$.

Os termos erro empírico, erro amostral e risco empírico também são utilizados para se referir a esse erro.

Uma vez que o conjunto de treinamento deveria "representar" bem os demais dados do domínio, é razoável esperar que o *learner* busque uma solução que funcione bem nesse conjunto. Esse paradigma de aprendizado no qual o *learner* retorna uma hipótese h que minimiza $E_S(h)$ se denomina *Empirical Risk Minimization* (ERM).

Daqui em diante, assumimos que o *learner* é um *Empirical Risk Minimizer*, ou seja, que seu algoritmo retorna uma hipótese de menor erro empírico

entre os classificadores que ele pode gerar.

2.2.2 *Overfitting*

Embora a regra de aprendizado ERM pareça muito natural, sem o devido cuidado, essa abordagem pode resultar em *overfitting*, que é quando a hipótese consegue "decorar" perfeitamente o conjunto de treino, mas não tem um bom desempenho para exemplos não vistos.

Para ilustrar o que mencionamos, vamos recorrer ao exemplo adotado na Seção 2.2 de (SHWARTZ; DAVID, 2014). Considere o problema de predizer se um mamão é saboroso (rótulo 1) ou não (rótulo 0) com base na sua maciez e cor (atributos reais). A maciez pode ser vista como uma intensidade que varia de duro para macio, enquanto a intensidade de cor representa o intervalo de verde escuro, passando por amarelo e avermelhado, para marrom escuro. Agora, considere uma amostra como ilustrada na Figura 2.1.

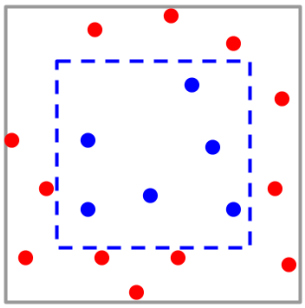

Figura 2.1: Exemplo de amostras de mamões (saborosos em azul).
Fonte: Ilustração da Seção 2.2 de (SHWARTZ; DAVID, 2014).

Assuma que a distribuição μ é de tal forma que os mamões estão distribuídos uniformemente dentro do quadrado cinza da Figura 2.1 e a função que rotula perfeitamente, f, determina que o rótulo é 1 se o exemplo está dentro do quadrado azul, e 0 caso contrário. Além disso, a área do quadrado cinza é 2, e a área do quadrado azul é 1. Considere o seguinte classificador:

$$h_S(x) = \begin{cases} y_i, & \text{se } \exists i \in [m] \text{ tal que } x_i = x \\ 0, & \text{caso contrário.} \end{cases} \tag{2-3}$$

Claramente, temos que $E_S(h_S) = 0$ e, portanto, essa hipótese pode ser escolhida por um algoritmo ERM (ela é uma das hipóteses de erro empírico mínimo). Por outro lado, para nosso exemplo, o erro de generalização de qualquer classificador que atribui rótulo 1 apenas em um número finito de exemplos é $1/2$ (pela distribuição μ ilustrada). Portanto, temos $E_{(\mu,f)} = 1/2$. Note que encontramos uma hipótese que é perfeita no conjunto de treino, mas que é muito ruim em exemplos não vistos.

Uma forma comum de evitar *overfitting* é aplicar ERM sobre um espaço de busca restrito, em que o *learner* deve escolher um conjunto de hipóteses antecipadamente (antes de ver os dados). Esse conjunto é denotado por $\mathcal{H}$ e é denominado como a classe de hipóteses do *learner*. Para uma dada classe $\mathcal{H}$ e um conjunto de treino S, o *learner* $\text{ERM}_{\mathcal{H}}$ realiza ERM para escolher uma hipótese $h \in \mathcal{H}$ com o menor erro possível no conjunto S. Formalmente,

$$ERM_{\mathcal{H}}(S) \in \underset{h \in \mathcal{H}}{\operatorname{argmin}} E_S(h),$$

onde argmin define o conjunto de hipóteses em $\mathcal{H}$ que alcança o valor mínimo de $E_S(h)$ em S. Quando necessário, utilizamos h_S como hipótese resultante em aplicar $\text{ERM}_{\mathcal{H}}$ a S.

O modo mais simples de restringir uma classe de hipóteses é impor um limite no seu tamanho (ou seja, limitar o número de hipóteses em $\mathcal{H}$). De fato, é possível demonstrar que se $\mathcal{H}$ é um conjunto finito, então aplicar ERM sobre $\mathcal{H}$ não resulta em *overfitting* (Seção 2.3 de (SHWARTZ; DAVID, 2014)) desde que seja fornecido um conjunto de treino suficientemente grande (o tamanho depende de $|\mathcal{H}|$).

Apesar de parecer muito restritivo assumir que o tamanho de $\mathcal{H}$ é finito, essa é uma condição bem razoável na prática, pois contamos com recursos limitados. Por exemplo, $\mathcal{H}$ pode ser o conjunto de todos os classificadores que podem ser implementados em uma determinada linguagem de programação utilizando no máximo 10^9 *bits* de código (SHWARTZ; DAVID, 2014). Outro tipo de limitação comum em linguagens de programação é que valores numéricos costumam ser limitados (valores mínimo e máximo suportados) e números reais podem ser discretizados, digamos, usando uma representação de pontos flutuantes de 64 *bits*.

2.2.3 Complexidade de Amostragem

Veremos a seguir que, sob certas condições, é possível ter garantias de que o modelo de aprendizado vai convergir para um classificador de boa qualidade não somente nos dados de treinamento mas também para os demais dados do domínio. Essas garantias podem ser obtidas selecionando exemplos aleatoriamente e estão associadas ao número de exemplos fornecidos ao *learner*.

Primeiramente, destacamos que é comum tratar o aprendizado sob dois casos: (i) caso realizável, em que presumimos existir $h^* \in \mathcal{H}$ tal que $E_{(\mu,f)}(h^*) = 0$ e, consequentemente, para qualquer conjunto S de exemplos

aleatórios sorteados de acordo com μ e rotulados por f, temos $E_S(h^*) = 0$[1]; (ii) caso agnóstico, em que não supomos o caso anterior.

A primeira condição para obter tais garantias, e uma das premissas básicas da teoria do aprendizado estatístico, é que os exemplos do conjunto de treino são gerados independentemente e identicamente distribuídos (i.i.d.) de acordo com uma distribuição μ considerada.

2.2.3.1 Caso Realizável

Uma vez que $E_{(\mu,f)}(h_S)$ depende do conjunto de treino S, e esse conjunto é gerado por um processo aleatório, então existe aleatoriedade na escolha da hipótese h_S e, consequentemente, no erro $E_{(\mu,f)}(h_S)$. Sendo assim, devemos considerar que existe a probabilidade de um conjunto de treino não representar bem a distribuição real dos dados. É importante, portanto, calcular a probabilidade de gerar um conjunto de treino para o qual $E_{(\mu,f)}(h_S)$ não é tão grande. Usualmente, denota-se essa probabilidade de sortear um conjunto não representativo por δ, e utiliza-se $(1-\delta)$ como parâmetro de confiança da predição obtida.

Seguindo o raciocínio, uma vez que não podemos garantir uma predição perfeita, utiliza-se outro parâmetro para a qualidade da predição, o parâmetro de precisão, comumente denotado por ϵ. Interpreta-se $E_{(\mu,f)}(h_S) > \epsilon$ como falha do *learner*, e $E_{(\mu,f)}(h_S) \leq \epsilon$ como sucesso.

A partir desses dois parâmetros (δ, ϵ) é possível determinar um tamanho suficientemente grande para um conjunto de treino S, de modo que o modelo resultante de $\text{ERM}_{\mathcal{H}}$ sobre uma classe finita de hipóteses seja provavelmente (com confiança $1-\delta$) aproximadamente (com erro de no máximo ϵ) correto. Adiante, abordamos mais profundamente esses fatores e definimos formalmente a noção de aprendizado Provavelmente Aproximadamente Correto (PAC).

Definição 2.2.1 (Definição 3.1 de (SHWARTZ; DAVID, 2014)) *Uma classe de hipóteses $\mathcal{H}$ é PAC aprendível se existe uma função $m_{\mathcal{H}} : (0,1)^2 \mapsto \mathbb{N}$ e um algoritmo de aprendizado com a seguinte propriedade: para qualquer $\epsilon, \delta \in (0,1)$, para qualquer distribuição μ sobre $\mathcal{X}$, e para qualquer função de rotulação $f : \mathcal{X} \mapsto \mathcal{Y}$, se o caso realizável está assegurado com respeito a $\mathcal{H}$, μ, f, então quando executado o algoritmo de aprendizado em $m \geq m_{\mathcal{H}}(\epsilon, \delta)$ exemplos i.i.d. gerados por μ e rotulados por f, o algoritmo retorna uma hipótese h tal que, com probabilidade de pelo menos $1-\delta$ (sobre as escolhas dos exemplos), $E_{(\mu,f)}(h) \leq \epsilon$.*

[1]Sendo mais preciso, isso ocorre com probabilidade 1. Para simplificar a notação, podemos deixar isso de forma implícita no decorrer do texto.

A Definição 2.2.1 de PAC aprendível contém dois parâmetros aproximativos: (i) o parâmetro de precisão ϵ, que determina o quão longe a hipótese resultante está da melhor hipótese; e (ii) o parâmetro de confiança δ, que indica o quão provavelmente a hipótese resultante atende esse requisito de precisão. A função $m_{\mathcal{H}} : (0,1)^2 \mapsto \mathbb{N}$ determina a complexidade de amostragem de aprender H, que é o número de exemplos necessários para garantir a solução provavelmente aproximadamente correta. Observe que se $\mathcal{H}$ é PAC aprendível, existem muitas funções $m_{\mathcal{H}}$ que satisfazem a Definição 2.2.1. Portanto, sendo mais preciso, considera-se a complexidade de amostragem de aprender $\mathcal{H}$ como "função mínima", isto é, para qualquer ϵ e δ, $m_{\mathcal{H}}(\epsilon, \delta)$ é a função que retorna o menor inteiro que satisfaz a Definição 2.2.1 com precisão ϵ e confiança $(1-\delta)$.

Em particular, tem-se o seguinte resultado quando o tamanho de $\mathcal{H}$ é finito:

Corolário 1 (Corolário 3.2 de (SHWARTZ; DAVID, 2014))
Qualquer classe de hipóteses finita é PAC aprendível para um algoritmo $ERM_{\mathcal{H}}$ com complexidade de amostragem

$$m_{\mathcal{H}}(\epsilon, \delta) \leq \left\lceil \frac{log(|\mathcal{H}|/\delta)}{\epsilon} \right\rceil.$$

O Corolário 1 nos diz que para qualquer função de rotulação f e qualquer distribuição μ, assumindo que existe $h \in \mathcal{H}$ tal que $E_{\mu,f}(h) = 0$ (caso realizável), então para um conjunto de treino S de tamanho suficientemente grande ($|S| \geq log(|\mathcal{H}|/\delta)/\epsilon$), a hipótese h_S obtida por um *learner* $\mathrm{ERM}_{\mathcal{H}}$ sobre uma classe de hipóteses finita será provavelmente aproximadamente correta, isto é, $E_{(\mu,f)}(h_S) \leq \epsilon$.

2.2.3.2 Caso Agnóstico

É possível também generalizar esses resultados para um cenário menos restrito e mais realista do ponto de vista prático[2], no qual não é presumido o caso realizável.

Primeiramente, redefinimos μ como a distribuição de probabilidades sobre $\mathcal{X} \times \mathcal{Y}$. Isto é, μ agora é uma distribuição conjunta sobre os domínios de exemplos e rótulos e pode ser decomposta em duas partes: a distribuição marginal μ_x sobre o domínio de exemplos $\mathcal{X}$, e a distribuição condicional para cada exemplo x, $\mu((x,y)|x)$, sobre o domínio de rótulos $\mathcal{Y}$.

[2]O caso realizável assume a existência de um classificador perfeito, o que não é assegurado em muitos problemas práticos de classificação.

Observe que não podemos supor a existência de uma função f que rotula perfeitamente os dados, visto que existe uma probabilidade de serem atribuídas classes diferentes para dois exemplos idênticos. Portanto, redefinimos a noção de erro de generalização de uma hipótese h para:

$$E_\mu(h) = \mathop{\mathbb{P}}_{(x,y)\sim\mu}[h(x) \neq y]$$

Uma vez que pode não existir uma hipótese que rotula os exemplos perfeitamente, o novo objetivo é, portanto, encontrar garantias de encontrar a hipótese h que minimize o erro real, $E_\mu(h)$, conforme definido a seguir.

Definição 2.2.2 (Definição 3.3 de (SHWARTZ; DAVID, 2014)) *Uma classe de hipóteses $\mathcal{H}$ é agnóstica PAC aprendível se existe uma função $m_\mathcal{H} : (0,1)^2 \mapsto \mathbb{N}$ e um algoritmo de aprendizado com a seguinte propriedade: para qualquer $\epsilon, \delta \in (0,1)$, para qualquer distribuição μ sobre $\mathcal{X} \times \mathcal{Y}$, quando executado o algoritmo de aprendizado em $m \geq m_\mathcal{H}(\epsilon, \delta)$ exemplos i.i.d. gerados por μ, o algoritmo retorna uma hipótese h tal que, com probabilidade de pelo menos $1 - \delta$ (sobre as escolhas dos exemplos),*

$$E_\mu(h) \leq \min_{h' \in \mathcal{H}} E_\mu(h') + \epsilon$$

,

em que o primeiro termo do lado direito da desigualdade é relativo ao menor erro possível que podemos obter considerando todas as hipóteses em $\mathcal{H}$.

Note que a Definição 2.2.2 generaliza a Definição 2.2.1. Claramente, se o caso realizável está assegurado, a definição de agnóstico PAC aprendível fornece a mesma garantia de PAC aprendível. Em particular, tem-se o seguinte resultado quando o tamanho de $\mathcal{H}$ é finito:

Corolário 2 (Corolário 4.6 de (SHWARTZ; DAVID, 2014))
Qualquer classe finita de hipóteses $\mathcal{H}$ é agnóstica PAC aprendível usando um algoritmo de aprendizado ERM com complexidade de amostragem

$$m_\mathcal{H}(\epsilon, \delta) \leq \left\lceil \frac{2log(2|\mathcal{H}|/\delta)}{\epsilon^2} \right\rceil.$$

O Corolário 2 nos diz que, para qualquer distribuição μ, a hipótese h_S obtida por um *learner* $\text{ERM}_\mathcal{H}$ sobre uma classe finita de hipóteses $\mathcal{H}$ será provavelmente aproximadamente correta, desde que o treinamento seja realizado em um conjunto de treino S de tamanho suficientemente grande ($|S| \geq 2log(|2\mathcal{H}|/\delta)/\epsilon^2$).

2.2.3.3 Dimensão VC

Vimos nas seções anteriores que classes de hipóteses finitas são aprendíveis e que a complexidade de amostragem de uma classe de hipóteses é limitada pelo *log* do seu tamanho. Em outras palavras, vimos que a finitude de $\mathcal{H}$ é uma condição suficiente para aprendizagem. Entretanto, essa não é uma condição necessária.

Em teoria de aprendizado existe uma outra noção, chamada dimensão VC, que caracteriza se uma classe de hipóteses é ou não (PAC) aprendível. Essa noção considera não o tamanho de $\mathcal{H}$, mas o quão flexível é uma classe de hipóteses em um conjunto de exemplos. A definição de dimensão VC é dada adiante.

Definição 2.2.3 (Definição 6.3 de (SHWARTZ; DAVID, 2014)) *Seja $\mathcal{H}$ uma classe de funções que mapeiam $\mathcal{X}$ em $\{0,1\}$, $\mathcal{X} \mapsto \{0,1\}$, e seja $C = \{c_1, ..., c_m\} \subset \mathcal{X}$. Considere uma função que mapeia C em $\{0,1\}$, $C \mapsto \{0,1\}$, como um vetor em $\{0,1\}^{|C|}$. Dizemos que $\mathcal{H}$ particiona o conjunto finito C (ou que C pode ser particionado por $\mathcal{H}$) se o conjunto de funções que mapeiam C em $\{0,1\}$, $C \mapsto \{0,1\}$, que podem ser derivadas de $\mathcal{H}$, dado por*

$$\mathcal{H}_C = \{(h(c_1), ..., h(c_m)) : h \in \mathcal{H}\},$$

é o conjunto de todas as funções que mapeiam C em $\{0,1\}$, ou seja, $|\mathcal{H}_C| = 2^{|C|}$.

Definição 2.2.4 (Definição 6.5 de (SHWARTZ; DAVID, 2014)) *A dimensão VC[3] de uma classe de hipóteses $\mathcal{H}$, denotada por $VCdim(\mathcal{H})$, é a maior cardinalidade de um conjunto $C \subset \mathcal{X}$ que pode ser particionado por $\mathcal{H}$. Se $\mathcal{H}$ particiona conjuntos de tamanho arbitrariamente grandes, então dizemos que $\mathcal{H}$ tem dimensão VC infinita.*

Conforme definido no teorema a seguir, uma classe de hipóteses é PAC aprendível se, e somente se, sua dimensão VC é finita. Portanto, temos que $VCdim(\mathcal{H})$ caracteriza PAC aprendizagem.

Teorema 2.1 (Teorema 6.7 de (SHWARTZ; DAVID, 2014)) *Seja $\mathcal{H}$ uma classe de hipóteses de funções de um domínio $\mathcal{X}$ para $\{0,1\}$. Então, as seguintes afirmações são equivalentes:*

1 $\mathcal{H}$ é agnóstica PAC aprendível.

[3] O termo VC é uma abreviação para Vapnik-Chervonenkis, sobrenomes de Vladimir Vapnik e Alexey Chervonenkis, que originalmente definiram esse conceito.

2 $\mathcal{H}$ *é PAC aprendível.*

3 $\mathcal{H}$ *tem dimensão VC finita.*

A partir da dimensão VC também é possível obter limites no tamanho necessário para o conjunto de dados de treinamento. Por sua vez, esses limites deixam de depender de $|\mathcal{H}|$ e são definidos em função de $VCdim(\mathcal{H})$. Observe que, para um classe finita de hipóteses $\mathcal{H}$ e para um conjunto C qualquer, temos que $\mathcal{H}_C \leq \mathcal{H}$. Portanto, se $|\mathcal{H}| < 2^{|C|}$, então C não pode ser particionado por $\mathcal{H}$. Isso implica que $VCdim(\mathcal{H}) \leq log_2(|\mathcal{H}|)$.

Para uma leitura mais aprofundada sobre todos esses conceitos abordados, bem como sobre detalhes desses resultados presentes na literatura, sugerimos ao leitor duas fontes: (SHWARTZ; DAVID, 2014) e (ANTHONY; BARTLETT, 1999).

2.3 Machine Teaching

Considere um *learner* sendo um algoritmo de *Machine Learning*, por exemplo, um *Support Vector Machine* (SVM) ou *Kmeans clustering*. Agora considere um *teacher* como uma entidade que quer que o *learner* aprenda uma hipótese alvo h^*. Por exemplo, h^* pode ser um hiperplano específico em SVM ou a localização dos k centróides em *Kmeans*. O *teacher* conhece h^* e a ensina para o *learner* fornecendo exemplos de treino, o *teaching set* (GOLDMAN; KEARNS, 1995): um conjunto de exemplos D tal que, para cada $h \neq h^*$, existe um exemplo $e \in D$ para o qual $h(e) \neq h^*(e)$. A grosso modo, esse conjunto distingue h^* das demais hipóteses que o *learner* pode gerar. A área de *Machine Teaching* (SHINOHARA, 1991) estuda o problema de encontrar um *teaching set* ótimo D. Por esse motivo, simplesmente nos referimos a esse objetivo como o problema de *Machine Teaching*. Geralmente se utiliza a cardinalidade de D como medida de otimalidade: quanto menor $|D|$, melhor (ZHU, 2015). Entretanto, existem outras medidas, como discutimos mais adiante.

Machine Teaching pode ser tratado sob diversos cenários. Em alguns casos, o *teacher* pode criar exemplos de forma arbitrária. Neste trabalho, em particular, o foco se dá em um cenário denominado *pool-based*, no qual o *teacher* recebe um conjunto finito de candidatos e deve selecionar os exemplos apenas desse conjunto. Além desses casos, outros aspectos de cenários de ensino podem ser considerados. Os trabalhos clássicos (SHINOHARA, 1991; GOLDMAN; KEARNS, 1995) consideram o cenário em que o *teacher* envia de uma só vez um conjunto de exemplos rotulados para o *learner*, que então tem que retornar a hipótese alvo. Já em trabalhos mais recentes, o foco tem sido o ambiente

interativo (LIU et al., 2017; CHEN et al., 2018; LIU et al., 2018; DASGUPTA et al., 2019) — no qual o *teacher* e o *learner* interagem em várias rodadas. Em cada rodada, o *teacher* envia exemplos para o *learner*, que retorna algum *feedback*. Esse processo continua até que o *learner* alcance a hipótese desejada (ou uma boa aproximação dela). Outro fator a ser considerado é se o *teacher* tem muitas informações sobre o *learner* (LIU et al., 2018) ou um conhecimento limitado sobre ele (*black-box learners* (DASGUPTA et al., 2019)).

Machine Teaching tem sido utilizado em vários contextos como *crowdsourcing* (JOHNS; AODHA; BROSTOW, 2015; ZHOU; NELAKURTHI; HE, 2018), sistemas tutores inteligentes (RAFFERTY et al., 2016), análise de *training set attacks* (MEI; ZHU, 2015). Além disso, ferramentas comerciais baseadas no paradigma de *Machine Teaching* estão sendo desenvolvidas pelo *Microsoft Machine Teaching Group* (conforme detalhado em sua página web[4]) como, por exemplo, PICL, que emprega uma seleção de exemplos em função da interação com o *teacher* com o objetivo de permitir que não especialistas em aprendizado de máquina construam seus modelos; LUIS para compreensão de linguagem natural; e outros projetos de construção de modelos para sistemas autônomos.

2.3.1 Formulações para o Problema

Zhu et al. (2018) discutem duas formulações para o problema de *Machine Teaching*. Seja $TeachingCost(D)$ uma função que atribui um custo associado a um conjunto de treinamento D. Seja $TeachingRisk(\theta)$ uma função que atribui um custo associado a um conjunto de hipóteses θ. E seja $Hypotheses(D)$ uma função que retorna um conjunto de hipótese em função de um conjunto de treinamento D.

A primeira formulação tenta minimizar o custo de ensino sob um aprendizado suficiente:

$$\begin{aligned} \min_{D,\theta} \ & TeachingCost(D) \\ s.a: \ & TeachingRisk(\theta) \leq \epsilon \\ & \theta = Hypotheses(D) \end{aligned}$$

Essa formulação define tanto uma aproximação quanto o caso em que o *learner* deve aprender exatamente a hipótese h^*. De fato, isso inclui a definição clássica geralmente adotada na literatura. Mais precisamente, seja $TeachingCost(D) = |D|$ a cardinalidade do *teaching set*, seja $Hypotheses(D)$ o espaço de hipóteses

[4]Site: www.microsoft.com/en-us/research/group/machine-teaching-group/

do *learner* depois de ver o conjunto D, e seja $TeachingRisk(\theta) = 0$ se θ é o conjunto unitário $\{h^*\}$, e $TeachingRisk(\theta) = \infty$ para outros espaços de hipóteses. Desse modo, a formulação define precisamente o problema de encontrar o menor conjunto de treino D que generalize h^*.

A outra formulação permite otimizar o aprendizado sob restrições de ensino:

$$\begin{aligned} \min_{D,\theta} \ & TeachingRisk(\theta) \\ s.a: \ & TeachingCost(D) \leq B \\ & \theta = Hypotheses(D) \end{aligned}$$

Considerando $Hypotheses(D)$ como o espaço de hipóteses do *learner* após receber o conjunto D. Seja $TeachingRisk(\theta)$ a função que mede o quão insatisfeito o *teacher* está com θ (podendo utilizar h^* para isso). Essa função pode ser, por exemplo, o erro para a pior das hipóteses do conjunto θ. Seja $TeachingCost(D)$ a função que mede o tempo de treinar o *learner* utilizando D. Nesse caso, queremos encontrar o conjunto de treino que resulte no melhor classificador que pode ser treinado dentro de um limite de tempo B, que é o problema que tratamos no Capítulo 4.

2.3.2 As Diferenças Entre Machine Teaching e Outros Paradigmas

Nesta seção, destacamos a diferença entre *Machine Teaching* e outros dois paradigmas tradicionais de aprendizado: *Passive Learning* e *Active Learning*. Para ilustrar essas diferenças, recorremos à Figura 2.2, uma adaptação da Figura 1 de (LIU et al., 2017).

Em *Passive Learning*, constrói-se um *dataset* a partir de uma amostra aleatória *iid* uniformemente de exemplos rotulados. Esse conjunto é enviado ao *learner*, em uma única vez, para que ele treine e retorne uma hipótese.

Em *Active Learning* (AL), o *learner* é responsável por escolher os dados com os quais ele vai treinar. A princípio, o conjunto de dados não está rotulado e o custo está associado a rotular exemplos. Isso é feito por meio de requisições (as chamadas *queries*), de modo que o *learner* solicita de um oráculo o rótulo de quais exemplos ele quer. O objetivo é que o modelo alcance um bom desempenho utilizando o menor número possível de dados rotulados (SETTLES, 2009). Observe que em AL o *learner* é encarregado de escolher exemplos, enquanto em *Machine Teaching* isso é tarefa do *teacher*. Outra diferença importante é que apesar de saber rotular os exemplos, o oráculo não conhece (de antemão) h^*.

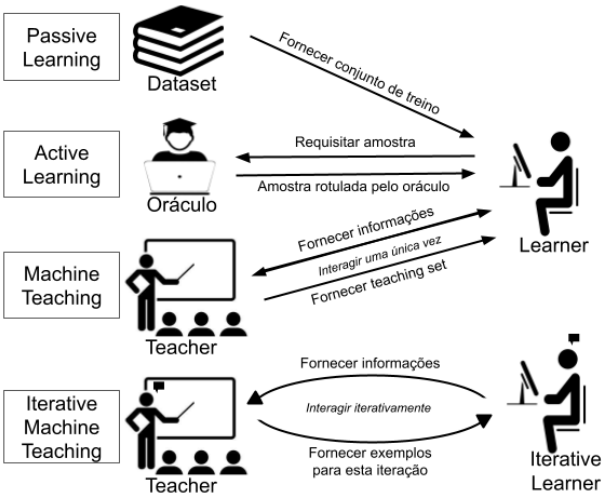

Figura 2.2: Comparação entre *Machine Teaching* e outros paradigmas de aprendizado.
Fonte: Adaptação da Figura 1 de (LIU et al., 2017).

Também podemos analisar a diferença desses métodos observando suas particularidades e desempenhos obtidos em um problema simplório de aprendizado. Assim como em (SETTLES, 2009), adotamos o exemplo clássico geralmente utlizado para destacar o bom desempenho de AL, no qual se deseja ensinar o melhor *threshold* para um conjunto de valores reais. Suponha que exemplos são pontos distribuídos em uma linha unidimensional, e nossa classe de hipóteses é uma função binária simples g parametrizada por θ:

$$g(x;\theta) = \begin{cases} 1, & \text{se } x > \theta \\ 0, & \text{caso contrário.} \end{cases}$$

De acordo com o conceito de (PAC) aprendível (VALIANT, 1984), se a distribuição de dados pode ser perfeitamente classificada por alguma hipótese h, então é suficiente amostrar aleatoriamente $O(1/\epsilon)$ exemplos rotulados para alcançar uma hipótese de erro máximo ϵ (note que esse é o número de exemplos necessários para *Passive Learning)*. Agora considere o cenário *pool-based* de *Active Learning*, no qual podemos adquirir de forma gratuita (ou a um custo irrisório) o mesmo número de exemplos não rotulados a partir dessa distribuição, e o custo está relacionado a requisitar rótulos. Se ordenarmos esses pontos (valores reais) em uma linha, seus rótulos (desconhecidos) serão uma sequência de zeros seguidos por uns, e o objetivo será descobrir o local em que a transição ocorre requisitando o menor número de rótulos possível. Realizando uma busca binária simples nesses pontos, um classificador com erro no máximo ϵ pode ser alcançado após rotular $O(\log 1/\epsilon)$ exemplos – uma vez que todos os outros rótulos podem ser inferidos – resultando em uma redução

exponencial no número de amostras rotuladas necessárias. Por outro lado, em *Machine Teaching* o mesmo limite superior de erro pode ser obtido com apenas dois exemplos : $(\theta^* - \frac{\epsilon}{2},\ 0)$ e $(\theta^* - \frac{\epsilon}{2},\ 1)$ – isso ocorre porque o *teacher* conhece θ^* de antemão e não necessita explorar outros exemplos.

2.3.3 Complexidade do Problema

Zhu et al. (2018) destacam que a natureza combinatorial e *bilevel*[5] para o problema de otimização de *Machine Teaching* (formulações na Seção 2.3.1) torna essa tarefa bastante complexa. Até mesmo instâncias simples do problema são NP-Difíceis. De fato, é fácil perceber que, em uma instância na qual o *learner* é um *Empirical Risk Minimizer* (ERM) e $h^* \in \mathcal{H}$, encontrar o *teaching set* ótimo é equivalente a resolver o problema de *Set Cover* (um problema clássico NP-Difícil): cada exemplo e elimina um subconjunto subótimo de hipóteses ($H(e) = \{h \in \mathcal{H} | h(e) \neq h^*(e)\}$), e um *teaching set* é um conjunto de exemplos que elimina (ou "cobre") todas as hipóteses subótimas ($\mathcal{H} \setminus \{h^*\}$).

Por conta dessa complexidade, geralmente são propostos algoritmos com garantias aproximativas ou heurísticas para resolver o problema.

[5] A otimização *bilevel* é um tipo especial de otimização em que um problema é incorporado em outro.

3
Machine Teaching com Informações Limitadas Sobre o *Learner*

Reservamos este capítulo para tratar do problema de *Machine Teaching* em sua forma tradicionalmente abordada: minimizar o tamanho do *teaching set*. Em particular, estamos interessados no cenário em que o *teacher* tem conhecimento limitado sobre o *learner*. Destacamos que estes resultados foram publicados em forma de artigo na *International Conference on Machine Learning* (ICML) (CICALESE et al., 2020).

3.1
Introdução

A maioria dos trabalhos da literatura de *Machine Teaching* assumem que o *teacher* tem conhecimento significativo sobre o *learner*, por exemplo, sobre sua classe de hipóteses e o procedimento específico empregado para aprender uma hipótese a partir de exemplos rotulados. No entanto, essa suposição exclui muitas situações importantes como ensino humano e modelos automáticos com comportamento *black-box* (por exemplo, *Deep Nets*). Assim, trabalhos recentes na área têm se concentrado em analisar cenários nos quais o conhecimento do *teacher* sobre o *learner* é limitado (LIU et al., 2018; DASGUPTA et al., 2019).

Em particular, Dasgupta et al. (2019) resolveram o problema para *black-box learners* considerando o caso realizável, em que o único conhecimento do *teacher* sobre o *learner* é que $h^* \in \mathcal{H}$, isto é, sua classe de hipóteses contém a hipótese alvo. Eles consideraram um modelo de interação no qual o *teacher* envia, iterativamente, exemplos rotulados para o *learner* e este retorna uma hipótese consistente[1] com todos os exemplos já recebidos até então. Como garantia de qualidade do algoritmo proposto, os autores fornecem um limite superior no número de exemplos necessários para ensinar h^* a um *learner* de pior caso (SHINOHARA, 1991; GOLDMAN; KEARNS, 1995), que pode selecionar qualquer hipótese (da sua classe de hipóteses) entre aquelas de erro mínimo nos exemplos já recebidos.

Nesta etapa da pesquisa, nós refinamos alguns resultados para o modelo de *Machine Teaching* com *black-box learners* considerado em (DASGUPTA et al., 2019) e também introduzimos e analisamos novas variantes dele. Nossa motivação se dá, em parte, pela necessidade de considerar um cenário mais

[1]Uma hipótese h é consistente com um conjunto de exemplos S se $h(x) = h^*(x)$ para todo $x \in S$.

realista ao qual pode não ser possível guiar o *learner* para uma hipótese perfeita (a hipótese alvo não pertence a classe de hipóteses do *learner*) e, em parte, pelo fato de que o *learner* pode não ser um carrasco/adversário para o *teacher*. De forma mais clara, enquanto um *learner* de pior caso pode retornar qualquer das hipóteses de erro mínimo nos exemplos já recebidos, um *learner* menos carrasco pode ter um comportamento mais previsível durante a escolha de qual hipótese retornar (entre as de erro mínimo nos exemplos recebidos). Como discutiremos adiante, assumindo um modelo de *learner* que se move suavemente pela classe de hipóteses, podemos obter um *teacher* com um melhor limite superior no número de exemplos enviados.

3.1.1 Modelo de Ensino

Antes de enunciar nossos resultados, gostaríamos de destacar o modelo de ensino adotado. Existe um conjunto $\mathcal{X}$ de exemplos, e um conjunto finito $\mathcal{Y}$ de rótulos possíveis para cada exemplo. Por hipótese, queremos dizer uma função que mapeia cada exemplo em $\mathcal{X}$ para um rótulo em $\mathcal{Y}$. Primeiramente, assumimos que:

- ***Teacher:*** tem uma hipótese alvo $h^* : \mathcal{X} \mapsto \mathcal{Y}$, desconhecida pelo *learner*;
- ***Learner:*** tem uma classe de hipóteses $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$, desconhecida pelo *teacher*.

Consideramos o cenário no qual o *teacher* envia, iterativamente, um conjunto de exemplos rotulados $(e, h^*(e))$ ao *learner* que, por sua vez, retorna uma hipótese (da sua classe de hipóteses) que minimiza o erro nos exemplos já recebidos até então. O objetivo do *teacher* é enviar o mínimo de exemplos possível para fazer o *learner* retornar $h \in \mathcal{H}$ com o menor número de erros em todo *dataset* D (no caso realizável, isso significa que o *learner* retornará h^*). Vale ressaltar que o *teacher* não tem qualquer informação adicional sobre o algoritmo de aprendizado utilizado pelo *learner* para a seleção de hipótese dentre suas hipóteses de erro mínimo.

Uma noção fundamental em *Machine Teaching* é a de *teaching set* para h^* (GOLDMAN; KEARNS, 1995), que é um conjunto de exemplos $X \subseteq \mathcal{X}$ que distingue h^* de todas as outras hipóteses em $\mathcal{H}$, ou seja, para cada $h \neq h^*$ existe um exemplo $e \in X$ para o qual $h(e) \neq h^*(e)$. Utilizamos $\mathcal{TS}(\mathcal{H}, h^*)$ para denotar o menor *teaching set* para h^*. Quando $(\mathcal{H}, h^*)$ é claro pelo contexto, utilizamos apenas $\mathcal{TS}$. Além disso, usamos $m = |\mathcal{X}|$ e $n = |\mathcal{H}|$ para os tamanhos dos conjuntos de exemplos e hipóteses, respectivamente, e $erros(h)$ para denotar o conjunto de exemplos no qual h falha $(h(x) \neq h^*(x))$.

3.2 Contribuições e Trabalhos Relacionados

Nossa primeira contribuição é um algoritmo de ensino (*teacher*) denominado $\texttt{OSCT}_{base}$ que com alta probabilidade garante a convergência para a hipótese alvo h^* usando $O(\mathcal{TS}\ log\, m\ log\, n)$ exemplos (Teoremas 3.1 e 3.2). Sua corretude recai em uma nova análise do algoritmo para o problema de *Online Set Cover*[2] proposto em (ALON et al., 2009). A principal dificuldade para derivar esse resultado é lidar com dependência não trivial entre as ações do *teacher* e do *learner* no decorrer das iterações. Para superar essa dificuldade, contamos com técnicas de martingale e argumentos provenientes de *decoupling* (VICTOR; GINÉ, 1999).

Dasgupta et al. (2019) apresentaram resultados interessantes para *black-box learners* e um desses resultados também é um algoritmo de *teacher* baseado em uma adaptação (diferente) do algoritmo proposto em (ALON et al., 2009) para o problema *Online Set Cover*. Assim como nosso método, seu algoritmo também converge para h^* com alta probabilidade enviando $O(\mathcal{TS}\ log\, m\ log\, n)$ exemplos, apesar de que sua análise é baseada no conhecimento de um limite superior para n. No entanto, algumas sutilezas relevantes decorrentes da interdependência entre *teacher* e *learner* não foram abordadas nas análises de (DASGUPTA et al., 2019). Nesse sentido, consideramos nossa análise (pela aplicação do nosso Lema 3.3.2) como uma contribuição adicional para clarear e formalizar a validade de uma afirmação crucial em seus argumentos (Lema 5 de (DASGUPTA et al., 2019)). Detalhes são apresentados no Apêndice B. Dito isso, gostaríamos de enfatizar que o algoritmo e os resultados obtidos em (DASGUPTA et al., 2019) estão corretos.

Também usamos nosso algoritmo $\texttt{OSCT}_{base}$ como base para obter melhores resultados e para outras variantes do problema. Na Seção 3.3.2, propomos um algoritmo modificado que obtém um limite mais forte que depende da distribuição (desconhecida) do número de erros entre as hipóteses em $\mathcal{H}$ (Teorema 3.3). Na Seção 3.3.3, generalizamos o limite acima para o *caso não realizável*, no qual o *teacher* não pode assumir que a classe de hipóteses do *learner* contém a hipótese alvo. Nosso algoritmo garante que o *learner* converge para a hipótese $\tilde{h}$ que é a mais próxima de h^* em sua classe de hipótese, após receber $O(\mathcal{TS}_k \log m \log(m+n))$ exemplos, em que $\mathcal{TS}_k$ é um limite inferior no número de exemplos necessários para essa tarefa.

Esses resultados são válidos para o modelo de *learner* de *pior caso* (SHINOHARA, 1991; GOLDMAN; KEARNS, 1995), no qual nenhuma suposição é feita sobre qual hipótese o *learner* seleciona entre as de erro mínimo nos

[2]Daí a inspiração para o nome $\texttt{OSCT}_{base}$: *Online Set Cover Teacher*.

exemplos recebidos até agora. Diferentes modelos para o comportamento do *learner* foram recentemente considerados (ZILLES et al., 2011; GAO et al., 2017; CHEN et al., 2018; MANSOURI et al., 2019; KIRKPATRICK; SIMON; ZILLES, 2019). A suposição de que o *learner* navega suavemente em sua classe de hipóteses, sempre atualizando sua hipótese atual para uma que esteja "próxima" dela, foi usada para motivar o modelo de *preferência local* introduzido em (CHEN et al., 2018) e estendido em (MANSOURI et al., 2019). Para esse tipo de *learner*, quando a proximidade é medida em termos de distância de Hamming, apresentamos um algoritmo de ensino que com alta probabilidade envia $O(\mathcal{TS} \log n(\log err_1 + \log\log n))$ exemplos, em que err_1 é o número de erros da primeira hipótese fornecida pelo *learner* (Teoremas 3.5 e 3.6)[3]. Assim, este algoritmo de ensino se beneficia de um *learner* que começa próximo à hipótese alvo. É possível mostrar que esse limite não é alcançável por algoritmos eficientes, no modelo de pior caso, através de uma simples modificação de um limite inferior apresentado em (KORMAN, 2004a).

Também obtivemos limites melhores para o modelo em que o *teacher* pode solicitar ao *learner* um lote de hipóteses aleatórias consistentes com os exemplos apresentados até agora. Propomos um *teacher* que, com alta probabilidade, ensina h^* enviando no total $O(\mathcal{TS}\log(n+m))$ exemplos e solicitando $O(\mathcal{TS}\log(m+n))$ hipóteses aleatórias por rodada (Teorema 3.7). Para o caso relevante em que o número de hipóteses n é maior que o número de exemplos m, o limite do número de exemplos é o melhor possível – sob a suposição $\mathcal{P} \neq \mathcal{NP}$ – para algoritmos de tempo polinomial, mesmo quando o *teacher* conhece a classe de hipóteses $\mathcal{H}$(RAZ; SAFRA, 1997).

Vale destacar que nossos modelos de *learners* não adversários estão relacionados a modelos que já foram considerados (CHEN et al., 2018; BALBACH; ZEUGMANN, 2011; SINGLA et al., 2014; ANGLUIN; DOHRN, 2020). De fato, o modelo de transições suaves pode ser visto como uma instância do modelo de preferência local de (CHEN et al., 2018), em que hipóteses próximas à atual, em termos da distância de Hamming, são preferidas. Além disso, os *learners* que retornam uma hipótese aleatória foram considerados em (BALBACH; ZEUGMANN, 2011; SINGLA et al., 2014). Todavia, nesses trabalhos, ao contrário do nosso, o *teacher* tem conhecimento da classe de hipóteses do *learner*. Um resultado em que se comparam adversários de pior caso com adversários aleatórios, em outro contexto de ensino, é apresentado em (ANGLUIN; DOHRN, 2020).

Embora os algoritmos de ensino mencionados até agora tenham como

[3]Observação: esse resultado vale se $n = |\mathcal{H}|$ é redefinido como o número de hipóteses não equivalentes em $\mathcal{H}$ com respeito a h^*, em que duas hipóteses são equivalentes se concordarem com h^* exatamente nos mesmos exemplos de $\mathcal{X}$; portanto, $n \leq 2^m$ e $\log\log n \leq \log m$.

objetivo a obtenção de um *teaching set* pequeno, eles podem acabar produzindo um conjunto um não minimal (com respeito a deleção de exemplos[4]). Na Seção 3.5, mostramos que, com uma quantidade limitada de interações extras, o *teacher* é capaz de construir um *teaching set* minimal. Esse resultado pode ser útil quando o objetivo principal é obter um conjunto de treinamento comprimido.

Finalmente, para complementar nossos resultados teóricos, apresentamos na Seção 3.6 experimentos com 12 *datasets* reais que mostram que nosso *teacher* básico (o `OSCT` da Seção 3.3) envia significativamente menos exemplos, para atingir um determinado nível de acurácia, do que um *teacher* que não interage com o *learner*.

3.3 Ensino com *Learners* de Pior Caso

Relembre o modelo de ensino definido na Seção 3.1.1. Nesta seção consideramos o ensino para um *learner* de pior caso, que pode retornar qualquer hipótese $h \in \mathcal{H}$ que minimiza o erro nos exemplos já recebidos até então.

3.3.1 Caso Realizável

Primeiro consideramos o caso realizável em que a hipótese alvo h^* pertence a classe de hipóteses $\mathcal{H}$ do *learner*. Note que nesse caso o *learner* sempre envia uma hipótese que é consistente com todos os exemplos já vistos pelo *learner*.

Assim como em (DASGUPTA et al., 2019), nós aproveitamos a relação entre *machine teaching* e *set cover*. Dizemos que um exemplo $e \in \mathcal{X}$ cobre uma hipótese $h \in \mathcal{H}$ se essa hipótese classifica e erroneamente, ou seja, $h(e) \neq h^*(e)$. Observe que hipóteses cobertas pelos exemplos já recebidos pelo *learner* não são mais consideradas, pois ele nunca retorna uma dessas hipóteses. Portanto, uma vez que o *teacher* envia um conjunto de exemplos que cobre todas as outras hipóteses diferentes de h^*, o *learner* tem que retornar a hipótese alvo h^*, alcançando o objetivo do aprendizado. Isso significa que podemos reduzir o problema de *machine teaching* para o problema de *online set cover*: no início de cada iteração, o *teacher* recebe uma hipótese do *learner* e envia exemplos que cobrem essa hipótese (e com sorte outras hipóteses desconhecidas), de modo que o número total de exemplos enviados seja pequeno.

[4]Um *teaching set* S é não minimal se existe $x \in S$ tal que $S \setminus \{x\}$ também é um *teaching set*.

Nosso algoritmo de *teacher* é baseado no algoritmo para *online set cover* de (ALON et al., 2009) e pode ser descrito como segue (Figura 3.1). Para cada exemplo e e iteração t, o algoritmo mantém um peso W_e^t. Quando uma nova hipótese h é retornada pelo *learner* – e, portanto, ainda não coberta pelos exemplos já recebidos até então –, o *teacher* primeiro verifica se $h = h^*$. Em caso positivo, ele aceita h. Caso contrário, ele aumenta, de forma exponencial, os pesos dos exemplos em que h erra até que seus pesos se somem pelo menos 1; então, o algoritmo sorteia aleatoriamente alguns desses exemplos com probabilidade proporcional ao aumento dos seus pesos na iteração corrente e os envia ao *learner*. Se h não for aceita nem for coberta (nenhum exemplo é enviado) ao término da iteração, o algoritmo falha (retornando FALHA).

Algorithm $\texttt{OSCT}_{base}$

Entrada: Conjunto de exemplos $\mathcal{X}$, (palpite do) número de hipóteses do *learner* N, parâmetro de pesos iniciais $\omega \geq 0$

1. Inicializar pesos $W_e^0 = \frac{1}{2m}$ para todos os exemplos $e \in \mathcal{X}$
2. Para cada iteração $t = 1, 2, \ldots$:
   - Receber hipótese $H_t \in \mathcal{H}$ do *learner*
   - Se H_t é correta em todos os exemplos, pare e retorne H_t
   - **(Atualização de pesos)** Dobre os pesos de todos os exemplos errados até seus pesos somarem pelo menos 1. Ou seja, defina:

$$W_e^t = \begin{cases} 2^\ell \cdot W_e^{t-1} & , \text{ se } e \in erros(H_t) \\ W_e^{t-1} & , \text{ se } e \notin erros(H_t), \end{cases}$$

     onde ℓ é o menor inteiro não negativo tal que $W^t(H_t) := \sum_{e \in erros(H_t)} W_e^t \geq 1$
   - **(Enviando exemplos)** Para cada exemplo e, seja $D_e^t := W_e^t - W_e^{t-1}$ o aumento de peso do exemplo e (note que $D_e^t = 0$ se H_t não erra e)
     Repetir 4log N vezes: sortear no máximo um exemplo de modo que e seja amostrado com probabilidade D_e^t, e enviá-lo ao *learner* juntamente com seu rótulo correto (observe que H_t erra classificando esse exemplo)
   - Se nenhum exemplo foi enviado, retorne FALHA.

Figura 3.1: Algoritmo de *teacher* para o caso realizável.

Enquanto nosso *teacher* é baseado no algoritmo proposto em (ALON et al., 2009), a principal novidade está em sua análise: as hipóteses ("elementos" a serem cobertos) dependem dos exemplos ("conjuntos") enviados, e não apenas a análise em (ALON et al., 2009) não admite tal dependência, como também é conhecido que são necessários m exemplos para dependências mais gerais [(KORMAN, 2004b), Teorema 2.1.3]. Entretanto, podemos lidar com

as dependências que surgem neste contexto de ensino usando técnicas de martingale.

Teorema 3.1 *Considere o ensino de um learner de pior caso. No caso realizável $h^* \in \mathcal{H}$, o algoritmo $\texttt{OSCT}_{base}$ (Figura 3.1) inicializado com $N \geq n$ e $\omega = m$ sempre envia $O(\mathcal{TS} \log m \log N)$ exemplos e retorna a hipótese correta h^* com probabilidade pelo menos $1 - \frac{1}{N}$.*

Prova do Teorema 3.1. A primeira observação importante é que o algoritmo termina em no máximo $O(\mathcal{TS} \log \omega)$ iterações [(ALON et al., 2009), Lema 1]. Além disso, dado que em cada rodada o algoritmo envia no máximo $O(\log N)$ exemplos, obtemos o seguinte resultado:

Lema 3.3.1 $\texttt{OSCT}_{base}$ *envia $O(\mathcal{TS} \log \omega \log N)$ exemplos.*

Portanto, para provar o teorema, só precisamos limitar superiormente a probabilidade de que o algoritmo retorne FALHA. Seja $W^t(h) := \sum_{e \in erros(h)} W_e^t$ o peso de h ao término da iteração t, e seja $D^t(h) := W^t(h) - W^{t-1}(h)$ o aumento desse peso na rodada t. A intuição de por que a probabilidade de falha deve ser baixa é a seguinte: se o algoritmo falhar em uma hipótese h isso significa que seu peso é pelo menos 1 ao término da iteração e nenhum exemplo que cobre h foi enviado ao *learner*. Seja X_t uma variável indicadora que é igual a 0 se nenhum exemplo que cobre h for enviado na rodada t. Temos $Pr[X_t = 0] = (1 - D^t(h))^{4 \log N}$, então a probabilidade de falha deveria ser em torno de $\prod_t (1 - D^t(h))^{4 \log N} \approx e^{-(4 \log N) \sum_t D^t(h)} \leq e^{-2 \log N} = 1/N^2$, em que a última inequação é assegurada porque no início do algoritmo o peso de h é no máximo $1/2$ e ao término do algoritmo é pelo menos 1[5]. Aplicando *union bound*[6] sobre todas as hipóteses $h \in H$, concluiríamos que a probabilidade de falha é no máximo $1/N$.

O problema é que esse argumento ignora dependências estocásticas cruciais: os exemplos que são enviados afetam (por meio da resposta do *learner*) a evolução do peso de h, de modo que o conjunto de variáveis aleatórias $\{X_t\}$ não são independentes e, portanto, não podemos tomar o produto de probabilidades como acima. Para lidar com essa situação, nós a abstraímos como uma sequência de variáveis aleatórias dependentes de Bernoulli X_t's cujos vieses[7] (correspondentes a $1 - (1 - D^t(h))^{4 \log n}$) dependem do histórico. Nosso principal lema técnico mostra que, independente das correlações, a probabilidade de nenhum dos indicadores X_t estarem ativos é o que esperamos.

[5] $\sum_t D^t(h) = W^t(h) - W^0(h) \geq 1/2$

[6] *Union Bound* ou desigualdade de Boole: para qualquer coleção contável de eventos A_1, A_2, ..., A_n, temos que $Pr(\bigcup_{i=1}^n A_i) \leq \sum_{i=1}^n Pr(A_i)$

[7] Probabilidade de sucesso de cada variável.

Lema 3.3.2 (Bernoulli Adaptativo) *Considere um espaço de probabilidade finito com filtragem $\mathcal{F}_1 \subseteq \mathcal{F}_2 \subseteq \ldots$ e seja $X^1, \ldots, X^n \in \{0,1\}$ uma sequência adaptada de variáveis aleatórias de Bernoulli, possivelmente correlacionadas. Seja $Z^t := \Pr(X^t = 0 \mid \mathcal{F}_{t-1})$ a probabilidade condicional de que X_t seja 0. Então para qualquer tempo de parada τ, com respeito a $\mathcal{F}$, e $\alpha \geq 0$*

$$\Pr\left(X^1 = \ldots = X^\tau = 0 \quad e \quad \prod_{t \leq \tau} Z^t \leq \alpha\right) \leq \alpha.$$

Prova. Podemos assumir, sem perda de generalidade, que não há tempo de parada (ou seja, τ sempre é igual a n): podemos aplicar o resultado às variáveis $\tilde{X}^t := \mathbf{1}(\tau \geq t) \cdot X^t$ e $\tilde{B}^t := (1 - \mathbf{1}(\tau \geq t)) \cdot Z^t = \Pr(\tilde{X}^t = 0 \mid \mathcal{F}_{t-1})$ para obter o resultado no caso de parada.

Nesta prova em específico, usamos negrito para vetores e letras maiúsculas para variáveis aleatórias. Além disso, para um vetor $\mathbf{v} = (v^1, \ldots, v^n)$ e $t \leq n$, usamos $\mathbf{v}^{\leq t}$ (resp. $\mathbf{v}^{<t}$) para denotar o vetor $(v^1, \ldots, v^t)$ (resp. $(v^1, \ldots, v^{t-1})$). A mesma notação é empregada para restringir uma sequência de variáveis aleatórias aos seus t primeiros elementos.

Seja $X = (X^1, X^2, \ldots, X^n)$ e seja $Z = (Z^1, Z^2, \ldots, Z^n)$. Considerando as variáveis na ordem $Z^1, X^1, Z^2, X^2, \ldots, X^n, Z^n$, temos que para qualquer vetor fixo $\mathbf{z} = (z^0, z^1, \ldots, z^n)$ (sem qualquer suposição de independência)

$$\Pr\left(X = 0 \text{ e } Z = \mathbf{z}\right) = \prod_{t=1}^{n} f(t, \mathbf{z}) \cdot \prod_{t=1}^{n} g(t, \mathbf{z})$$

em que

$$\begin{aligned} f(t, \mathbf{z}) &= \Pr(X^t = 0 \mid Z^{\leq t} = \mathbf{z}^{\leq t}, X^{<t} = 0), \\ g(t, \mathbf{z}) &= \Pr(Z^t = z^t \mid Z^{<t} = \mathbf{z}^{<t}, X^{<t} = 0). \end{aligned}$$

Para qualquer histórico $\sigma \in \mathcal{F}_{t-1}$ até o momento $t-1$, no qual $Z^t = z^t$, temos $\Pr(X^t = 0 \mid \sigma) = z^t$, o que implica $\Pr(X^t = 0 \mid Z^{\leq t} = \mathbf{z}^{\leq t}, X^{<t} = 0) = z^t$. Assim, obtemos $\Pr(X = 0 \text{ e } Z = \mathbf{z}) = \mathit{prod}(\mathbf{z}) \cdot \prod_{t=1}^{n} z^t$, em que

$$\mathit{prod}(\mathbf{z}) = \prod_{t=1}^{n} g(t, \mathbf{z}).$$

Sendo Ω o conjunto de todos os $\mathbf{z}$'s tal que $\prod_{t=1}^{n} z^t \leq \alpha$, temos

$$\Pr\left(X = 0 \text{ e } \prod_{t=1}^{n} Z^t \leq \alpha\right) \leq \alpha \sum_{\mathbf{z} \in \Omega} \mathit{prod}(\mathbf{z}) \leq \alpha \sum_{z^1} \ldots \sum_{z^n} \mathit{prod}(\mathbf{z}),$$

em que a soma $\sum_{z^t}$ abrange todos os valores possíveis de Z^t (lembre-se de que assumimos que o espaço de probabilidade é finito). Finalmente, afirmamos que a soma no lado direito é igual a 1: usando a definição de $prod(\mathbf{z})$ obtemos que

$$\begin{aligned}\sum_{z^1}\dots\sum_{z^n} prod(\mathbf{z}) &= \sum_{z^1}\dots\sum_{z^n} prod(\mathbf{z}^{<n})\cdot g(n,\mathbf{z}) \\ &= \sum_{z^1}\dots\sum_{z^{n-1}} prod(\mathbf{z}^{<n})\left(\sum_{z^n} g(n,\mathbf{z})\right) \\ &= \sum_{z^1}\dots\sum_{z^{n-1}} prod(\mathbf{z}^{<n}).\end{aligned}$$

A iteração deste argumento $n-1$ vezes resulta em

$$\sum_{z^1}\dots\sum_{z^n} prod(\mathbf{z}) = \sum_{z^1}\Pr(Z^1 = z^1) = 1,$$

o que conclui a prova. ■

Com isso podemos limitar a probabilidade de falha do algoritmo, concluindo sua análise.

Lema 3.3.3 $\texttt{OSCT}_{base}$ *com* $\omega \geq m$ *e* $N \geq n$ *retorna* Falha *com probabilidade no máximo* $\frac{1}{N}$.

Prova. Fixe uma hipótese $h \in \mathcal{H}$. Considerando o *union bound* sobre todas as hipóteses, basta mostrar que a probabilidade de $\texttt{OSCT}_{base}$ falhar ao receber a hipótese h é no máximo $\frac{1}{N^2}$.

Observe que h é recebida no máximo uma vez por $\texttt{OSCT}_{base}$: depois de receber h pela primeira vez, o algoritmo envia um exemplo que cobre h (assim o *learner* nunca reenvia h) ou retorna Falha. Portanto, seja τ o momento em que h é recebida, ou seja, $H_\tau = h$ (seja τ a última rodada se h nunca for recebida), e seja X^t o indicador de que no tempo t um exemplo que cobre h foi enviado ao *learner* pelo algoritmo. Como mencionado anteriormente, pela atualização de pesos do algoritmo, na rodada τ temos o peso $W^\tau(h) = W^\tau(H_\tau) \geq 1$. Desde que $\omega \geq m$, os pesos iniciais satisfazem $W^0(h) \leq \frac{1}{2}$ e, portanto, os incrementos de peso até τ satisfazem $\sum_{t\leq\tau} D^t(h) \geq \frac{1}{2}$. Além disso, se o algoritmo falhar em h temos $X^1 = \ldots = X^\tau = 0$, temos

$$\Pr(\text{falha em } h) \leq \Pr\left(X^t = 0, \forall t \leq \tau, \text{ e } \sum_{t\leq\tau} D^t(h) \geq \frac{1}{2}\right). \tag{3-1}$$

Seja $\mathcal{F}_{t-1}$ a σ-álgebra gerada pelo histórico até a rodada $t-1$ mais a hipótese na rodada t. Pelo procedimento de amostragem, a probabilidade condicional $Z^t := \Pr(X^t = 0 \mid \mathcal{F}_{t-1})$ de que nenhum exemplo que cobre h foi

adicionado na rodada t é

$$Z^t = (1 - D^t(h))^{4 \log N} \leq e^{-4(\log N) D^t(h)}, \tag{3-2}$$

lembrando que $(1 - x) \leq e^{-x}$ para todo x. Então, $\sum_{t \leq \tau} D^t(h) \geq \frac{1}{2}$ implica $\prod_{t \leq \tau} Z^t \leq e^{-2\ logN} = \frac{1}{N^2}$, e o lado direito de (3-1) pode ser limitado superiormente:

$$\Pr(\text{falha em } h) \leq \Pr\left(X^t = 0, \forall t \leq \tau \text{ e } \prod_{t \leq \tau} Z^t \leq \frac{1}{N^2}\right).$$

Do Lema 3.3.2, isso ainda pode ser limitado superiormente por $\frac{1}{N^2}$ e, desse modo, $\Pr(\text{falha em } h) \leq \frac{1}{N^2}$. Isso conclui a prova. ■

Tornando o algoritmo agnóstico ao tamanho de $\mathcal{H}$. No Teorema 3.1, assumimos que o *teacher* conhece o número de hipóteses $n = |\mathcal{H}|$ do *learner*. No entanto, uma estratégia de adivinhar e dobrar pode ser usada para superar essa limitação. Mais concretamente, seja `OSCT` um algoritmo de ensino que implementa uma sequência de chamadas para $\texttt{OSCT}_{base}$, de modo que na i-ésima chamada o parâmetro N é definido como 2^{2^i} enquanto ω é definido como m. Além disso, para $i > 1$, a hipótese inicial para a i-ésima chamada é aquela que falhou na chamada $i-1$. O procedimento termina assim que alguma chamada para $\texttt{OSCT}_{base}$ aceitar h^*.

Seja $t = \lceil \log \log n \rceil$. Na t-ésima chamada de $\texttt{OSCT}_{base}$, o parâmetro N não é menor que n. Dessa forma, segue do Teorema 3.1 que essa chamada envia $O(\mathcal{TS} \cdot \log m \cdot 2^t)$ e retorna h^* com probabilidade de pelo menos $1 - 1/n$. Além disso, pelo Lema 3.3.1, as chamadas anteriores para $\texttt{OSCT}_{base}$ enviam $\sum_{i=1}^{t-1} O(\mathcal{TS} \log m \cdot 2^i) = O(\mathcal{TS} \log m \log n)$ exemplos. Portanto, temos o seguinte resultado.

Teorema 3.2 *Considere o ensino de um learner de pior caso no cenário realizável $h^* \in \mathcal{H}$. O algoritmo* `OSCT`*, com probabilidade de pelo menos $1 - \frac{1}{n}$, retorna a hipótese correta h^* e envia no máximo $O(\mathcal{TS} \log m \log n)$ exemplos.*

3.3.2
Garantia Melhorada Baseada na Qualidade das Hipóteses

O próximo teorema mostra que é possível obter uma melhoria no limite superior do número de exemplos quando se leva em conta a distribuição do número de erros das hipóteses em $\mathcal{H}$. Por exemplo, se apenas $O(1)$ hipóteses fizerem um número não constante de erros, então o limite no número de

exemplos enviados é melhorado de $O(\mathcal{TS} \log m \log n)$ para $O(\mathcal{TS}(\log n + \log m))$.

Teorema 3.3 *Considere o ensino para um learner de pior caso no caso realizável ($h^* \in \mathcal{H}$). Seja n_i o número de hipóteses em $\mathcal{H}$ cujo número de erros está entre $[2^{2^i}, 2^{2^{i+1}})$ para $i \geq 1$, e seja n_0 o número de hipóteses com erro em $[1, 4)$. Então, existe um algoritmo de teacher que, com probabilidade de pelo menos $\frac{4}{5}$, retorna uma hipótese correta h^*, e o número de exemplos enviados é*

$$O(\mathcal{TS}(\mathcal{H})) \cdot \left(\log m + \sum_{i=0}^{\log \log m} 2^i \log(n_i + 1) \right)$$

Em particular, isso é $O(\mathcal{TS}(\mathcal{H}) \log m \log(\max_i n_i + 1))$.

O ponto de partida é perceber que se rodarmos $\texttt{OSCT}_{base}$ inicializado com ω sendo o número máximo de erros de uma hipótese em $\mathcal{H}$, digamos err, então ele envia no máximo $O(\mathcal{TS} \log err \cdot \log n)$ exemplos. Desse modo, a ideia do algoritmo do Teorema 3.3 é a seguinte: instanciar cópias $\mathcal{A}_1, \ldots, \mathcal{A}_{\log \log m}$ do algoritmo $\texttt{OSCT}_{base}$, onde $\mathcal{A}_i$ é inicializado com $\omega = 2^{2^i}$. Então, quando uma hipótese h vem do *learner*, vemos em qual faixa $[2^{2^i}, 2^{2^{i+1}})$ seu número de erros cai e enviamos a hipótese para o algoritmo $\mathcal{A}_i$.

Como o tamanho do menor *teaching set* para as hipóteses que caem na mesma faixa de erros não é maior que $\mathcal{TS}(\mathcal{H}, h^*)$, podemos combinar as garantias dos algoritmos $\mathcal{A}_i$'s para obter a garantia acima (Detalhes no Apêndice C).

3.3.3 Caso Não Realizável

Agora consideramos o caso não realizável, no qual a hipótese correta h^* pode não pertencer à classe de hipóteses $\mathcal{H}$ do *learner*. Lembre-se que nesse caso, a cada rodada, o *learner* envia uma hipótese em $\mathcal{H}$ com o menor número de erros nos exemplos recebidos até então, e o objetivo do *teacher* é fazer com que o *learner* retorne uma hipótese em $\mathcal{H}$ com o menor número de erros totais em todo o conjunto de exemplos $\mathcal{X}$.

Precisamos primeiro de uma generalização da noção de *teaching set*. Informalmente, se a melhor hipótese em $\mathcal{H}$ possui k erros em $\mathcal{X}$, para isolá-la o *teacher* deverá enviar exemplos que certifiquem que as demais hipóteses possuem pelo menos $k+1$ erros. Dizemos que um conjunto de exemplos $\mathcal{X}' \subseteq \mathcal{X}$ é um *k-extended teaching set* com respeito a h^* se para cada hipótese $h \in \mathcal{H}$ com mais do que k erros, há pelo menos $k+1$ exemplos em $\mathcal{X}'$ em que h está

errado (difere de h^*). Usamos $\mathcal{TS}_k = \mathcal{TS}_k(\mathcal{H}, h^*)$ para denotar o tamanho do menor *k-extended teaching set* com respeito a h^*.

Observe que após o *learner* receber um conjunto de exemplos rotulados que contém um *k-extended teaching set*, ele retorna uma hipótese com no máximo k erros totais, pois tais hipóteses têm no máximo k erros nos exemplos recebidos, enquanto todas as outras hipóteses têm pelo menos $k+1$ erros nesses mesmos exemplos. Se k for definido como o número de erros da melhor hipótese em $\mathcal{H}$, o *learner* retorna uma hipótese ótima.

Teorema 3.4 *Considere o ensino de um learner de pior caso, em que h^* pode não pertencer a $\mathcal{H}$. Seja k o menor número de erros de uma hipótese em $\mathcal{H}$. Então existe um algoritmo de teacher que com probabilidade de pelo menos $1 - \frac{1}{m}$ retorna uma hipótese que comete k erros e envia no máximo $O(\mathcal{TS}_k \log m \log(m+n))$ exemplos.*

Em alto nível, a ideia do algoritmo é a mesma do caso realizável: ele tenta calcular de forma online um *k-extended teaching set* de tamanho pequeno, mas agora baseado em um algoritmo para uma generalização do problema *Online Set Cover* (BUCHBINDER; NAOR, 2009), em que os elementos podem precisar ser cobertos várias vezes. Porém, como o número mínimo de erros k é desconhecido, o algoritmo também precisa manter um limite inferior em k que é dado pelo número de erros da última hipótese recebida sobre os exemplos já enviados. O algoritmo termina quando recebe do *learner* uma hipótese cujo número total de erros corresponda a esse limite inferior. Os detalhes desse resultado são fornecidos no Apêndice D.

3.4 Outros Modelos de Learners

Nesta seção, mostramos que limites melhores são possíveis sob suposições razoáveis sobre a maneira como o *learner* pode escolher as hipóteses consistentes para retornar. Tratamos o caso realizável no qual $h^* \in \mathcal{H}$ e, aqui, usamos $\mathcal{H}_t \subseteq \mathcal{H}$ para denotar o conjunto de hipóteses consistente com todos os exemplos enviados pelo *teacher* nas rodadas $1, \ldots, t-1$ (sendo assim, $\mathcal{H}_t$ é o conjunto de possíveis hipóteses que o *learner* pode enviar na rodada t).

3.4.1 Learners de Transições Suaves

Primeiro consideramos o *modelo de transição suave* em que presumimos que o *learner* envia uma hipótese "próxima" àquela enviada na rodada anterior. Concretamente, usamos o número de discordâncias entre hipóteses $d(h, h') =$

$|\{x \in \mathcal{X} \mid h(x) \neq h'(x)\}|$ como medida de proximidade e supomos que a hipótese h_t enviada pelo *learner* na rodada t é uma em $\mathcal{H}_t$ com $(1+\alpha)$-distância mínima aproximada para a hipótese h_{t-1} enviada na rodada anterior, ou seja,

$$d(h_t, h_{t-1}) \leq (1+\alpha) \min_{h \in \mathcal{H}_t} d(h, h_{t-1}).$$

Disponibilizamos para esse modelo um algoritmo $\mathcal{A}^{\alpha}_{\text{close}}$ cuja garantia depende do número de erros err_1 da primeira hipótese enviada pelo *learner*. Isso significa que, se o *learner* tiver um bom palpite para a hipótese correta, menos exemplos serão necessários para completar o ensino. O algoritmo $\mathcal{A}^{\alpha}_{\text{close}}$ é obtido de $\texttt{OSCT}_{\text{base}}$ por meio de duas modificações simples: os pesos iniciais dos exemplos W^0_e são definidos como $1/2err_1$ em vez de $1/2m$, e o número de exemplos amostrados por rodada é $\frac{8}{1-2\alpha} \log N$ em vez de $4 \log N$. Esse algoritmo tem a seguinte garantia.

Teorema 3.5 *Considere o modelo de transição suave com $\alpha \in [0, \frac{1}{2})$ no caso realizável $h^* \in \mathcal{H}$. Seja err_1 o número de erros em $\mathcal{X}$ da hipótese inicialmente enviada pelo learner. Então, o algoritmo $\mathcal{A}^{\alpha}_{close}$, definido com $N \geq n$, envia $O(\mathcal{TS}\frac{1}{1-2\alpha} \log err_1 \log N)$ exemplos e, com probabilidade de pelo menos $1 - \frac{1}{N}$, retorna a hipótese correta h^*.*

A principal observação, para a obtenção de uma garantia que depende de err_1, é que no modelo de transição suave o número de erros das hipóteses enviadas pelo *learner* não pode aumentar rapidamente. Temos o seguinte:

Lema 3.4.1 *Sejam h, h' as hipóteses retornadas pelo learner nas rodadas $t-1$ e t, respectivamente. Então:*

a) $|erros(h')| \leq 2\,|erros(h') \cap erros(h)| + \alpha\,|erros(h)|$

b) $|erros(h')| \leq (4 + 2\alpha)\, err_1$.

Prova. Primeiro provamos o item (a). Seja $\text{erros}(h \setminus h')$ (resp. $\text{erros}(h' \setminus h)$) o conjunto de exemplos em que apenas h (resp. h') está errado. Além disso, seja $DIFF$ (resp. EQ) o número de exemplos em que ambos h e h' estão errados, mas dão uma classificação diferente (resp. igual). Formalmente,

$$DIFF = |\{e \in \text{erros}(h) \cap \text{erros}(h') \mid h(e) \neq h'(e)\}|$$
$$EQ = |\{e \in \text{erros}(h) \cap \text{erros}(h') \mid h(e) = h'(e)\}|.$$

O número de discordâncias entre essas hipóteses é

$$d(h, h') = |\text{erros}(h \setminus h')| + |\text{erros}(h' \setminus h)| + DIFF$$

$$d(h, h^*) = |\text{erros}(h \setminus h')| + DIFF + EQ$$

O modelo de transição suave garante que $d(h, h') \leq (1 + \alpha)\, d(h, h^*)$ de modo que $|\text{erros}(h' \setminus h)| \leq \alpha\, |\text{erros}(h \setminus h')| + (1 + \alpha)\, |\text{erros}(h) \cap \text{erros}(h')|$ e, portanto,

$$\begin{aligned} |\text{erros}(h')| &= |\text{erros}(h' \setminus h)| + |\text{erros}(h) \cap \text{erros}(h')| \\ &\leq \alpha\, |\text{erros}(h \setminus h')| + (2 + \alpha)\, |\text{erros}(h) \cap \text{erros}(h')| \\ &= \alpha\, |\text{erros}(h)| + 2|\text{erros}(h) \cap \text{erros}(h')|, \end{aligned}$$

que estabelece o item (a).

Prova do item [b]. Se h' é a primeira hipótese, o resultado é claramente válido porque a primeira hipótese comete erros err_1.

Assim, seja $t > 1$ a rodada em que h' é recebida e seja h a hipótese recebida na rodada $t - 1$. Temos que $|\text{erros}(h)| < 2err_1$, caso contrário h teria peso pelo menos 1[8] no início da rodada $t - 1$ e, consequentemente, $t - 1$ teria sido a última iteração do algoritmo. Assim, decorre do item (a) e por $|\text{erros}(h) \cap \text{erros}(h')| \leq |\text{erros}(h)|$ que $|\text{erros}(h')| \leq (4 + 2\alpha)err_1$. ■

Prova do Teorema 3.5. O limite do número de exemplos segue diretamente do Lema 3.3.1, já que $\mathcal{A}^{\alpha}_{\text{close}}$ se comporta como $\texttt{OSCT}_{\text{base}}$ inicializado com $\omega = err_1$, mas enviando $\frac{2}{1-\alpha}$ vezes mais exemplos por rodada.

A prova de que a probabilidade de retornar a hipótese correta é de pelo menos $1 - \frac{1}{N}$ é semelhante à do Lema 3.3.3: precisamos mostrar que se o algoritmo recebe a hipótese h' na rodada τ, então $\sum_{t \leq \tau} D^t(h')$, o aumento total dos pesos dos exemplos errados de uma hipótese h' é "grande". Isso é suficiente, pois os argumentos de concentração, após a desigualdade (3-1), garantem que a probabilidade de falha nesse ponto (ou seja, nenhum exemplo cobrindo h' foi enviado) é pequena. Mais precisamente, basta mostrar

$$\sum_{t \leq \tau} D^t(h') \geq \frac{1 - 2\alpha}{4}. \tag{3-3}$$

Notamos que no Lema 3.3.3 temos o limite inferior mais forte com o lado direito igual a $\frac{1}{2}$; a diferença é compensada pelo número extra de exemplos enviados por $\mathcal{A}^{\alpha}_{\text{close}}$ em cada rodada. Provamos a desigualdade (3-3) considerando dois casos:

Caso 1. $|\text{erros}(h')| \leq err_1$. Neste caso, $\sum_{t \leq \tau} D^t(h')$ é pelo menos $\frac{1}{2} > \frac{1-2\alpha}{4}$ uma vez que o peso inicial de h' é no máximo $\frac{1}{2}$ e seu peso final é de pelo menos 1 pela etapa de atualização de peso do algoritmo.

[8] Relembrando que em $\mathcal{A}^{\alpha}_{\text{close}}$ os exemplos têm pesos iniciais definidos como $1/2err_1$.

Caso 2. $|\text{erros}(h')| > err_1$. Segue do item (a) do Lema 3.4.1 que a hipótese recebida na rodada $\tau - 1$, digamos h, compartilha pelo menos $(|\text{erros}(h')| - \alpha\,|\text{erros}(h)|)/2$ exemplos errados com h'. Além disso, temos $|\text{erros}(h)| \leq 2err_1$: caso contrário, o peso inicial dessa hipótese, $W^0(h)$, já é pelo menos 1 e, portanto, o algoritmo falha na rodada $\tau - 1$, contradizendo que falha na rodada τ. Juntamente com a suposição de estar no Caso 2, isso dá $|\text{erros}(h)| \leq 2|\text{erros}(h')|$ e, consequentemente, o número de exemplos errados comuns entre h e h' é pelo menos $\frac{1-2\alpha}{2}\,|\text{erros}(h')|$. Como o peso de cada um desses exemplos comuns foi aumentado em pelo menos $1/2err_1$ na rodada $\tau - 1$ (os pesos são dobrados e começam em $1/2err_1$), temos que

$$\sum_{t \leq \tau} D^t(h') \geq D^{\tau-1}(h') \geq \frac{1-2\alpha}{2} \cdot err_1 \cdot \frac{1}{2err_1} = \frac{1-2\alpha}{4}.$$

Isso prova (3-3) e conclui a prova do Teorema 3.5.

Tornando o algoritmo agnóstico ao tamanho de $\mathcal{H}$. Para obter um algoritmo agnóstico para o tamanho de $\mathcal{H}$ procedemos como na Seção 3.3.1, mas realizando uma sequência de chamadas para $\mathcal{A}^{\alpha}_{\text{close}}$ em vez de $\texttt{OSCT}_{\text{base}}$.

A única questão diferente que surge na análise deste algoritmo é como limitar o número de erros $err_{1,i}$ cometidos pela primeira hipótese da i-ésima chamada de $\mathcal{A}^{\alpha}_{\text{close}}$. Usando o fato de que essa hipótese é exatamente a última retornada pela chamada anterior, juntamente com o item (b) do Lema 3.4.1, obtemos que $err_{1,i} \leq (4 + 2\alpha)\, err_{1,i-1}$ e, portanto, $err_{1,i} \leq (4 + 2\alpha)^{i-1}\, err_{1,1}$. Esta observação junto com os mesmos argumentos empregados na análise de `OSCT` nos permite estabelecer o seguinte teorema:

Teorema 3.6 *Sob as mesmas hipóteses do Teorema 3.5, existe um algoritmo de teacher agnóstico ao número de hipóteses n que envia $O(\mathcal{TS}\,\log n\,(\log err_1 + \log\log n))$ exemplos e, com probabilidade de pelo menos $1 - \frac{1}{n}$, retorna a hipótese correta h^*.*

Observe que, no modelo de aprendizado de pior caso, esse limite não é alcançável por algoritmos que executam em tempo polinomial, a menos que $\mathcal{NP} \subset \mathcal{BPP}$. Isso é mostrado através de uma simples modificação de um limite inferior de (KORMAN, 2004a) (veja Apêndice E).

3.4.2 O Modelo de *Learner* Aleatório

Supomos que em cada rodada o *learner* envia um lote de hipóteses *aleatórias* i.i.d. daquelas que são consistentes com os exemplos recebidos até então.

Mostramos que nessa situação o *teacher* pode explorar a aleatoriedade da escolha do *learner* para estimar o exemplo que cobre o maior número de hipóteses (que são consistentes com os exemplos já recebidos). Com esse conhecimento, o *teacher* pode recorrer a algoritmos para o problema de *Offline Set Cover* e melhorar significativamente o número de exemplos utilizados: mostramos que, com alta probabilidade, o *teacher* envia $O(\mathcal{TS}\,\log(n+m))$ exemplos, que é o melhor limite alcançável em tempo polinomial, sob a suposição de que $\mathcal{P} \neq \mathcal{NP}$, para o caso relevante em que o número de hipóteses n é maior que o número de exemplos m (RAZ; SAFRA, 1997).

Algoritmo $\mathcal{A}_{\text{rand}}$ (Figura 3.2) executa o algoritmo aproximativo guloso para o *Off-line Set Cover* sobre o processo empírico: a cada rodada t o *teacher* solicita um lote $\widetilde{\mathcal{H}}_t$ de T hipóteses aleatórias do conjunto $\mathcal{H}_t$ de hipóteses consistentes com os exemplos enviados até agora e envia ao *learner* um exemplo $\tilde{e}$ que abranja o maior número de hipóteses de $\widetilde{\mathcal{H}}_t$. O tamanho T do lote solicitado depende idealmente do tamanho do menor *teaching set* $\mathcal{TS}(\mathcal{H}, h^*)$; mas, como essa quantidade é desconhecida, o algoritmo também emprega uma abordagem de "adivinhar e dobrar". Na fase i ele usa $T = 2^i$ como um palpite para $\mathcal{TS}$ e executa o procedimento guloso para rodadas $2T$. Se dentro dessas rodadas o algoritmo não terminar, a fase i é concluída e a fase $i+1$ é iniciada.

Algoritmo $\mathcal{A}_{\text{rand}}$
Entrada: Exemplos $\mathcal{X}$

1. Inicialize o contador de rodadas $t = 0$
2. Para cada fase $i = 1, 2, \ldots$:
   - Atualize a estimativa de tamanho do *teaching set* $T = 2^i$
   - Para $2T$ rodadas
     - Atualize o contador de rodadas $t = t + 1$
     - Receba um *batch* $\mathcal{H}_t$ de T hipóteses aleatórias de $\mathcal{H}_t$
     - Se todas as hipóteses em $\mathcal{H}_t$ forem iguais a h^*, então **Retorne** **(*)**
     - Envie $\tilde{e} = \text{argmax}_e\,|\{h \in \mathcal{H}_t \mid e \in \text{erros}(h)\}|$

Figura 3.2: *teacher* $\mathcal{A}_{\text{rand}}$

O teorema a seguir é o principal resultado desta seção.

Teorema 3.7 *Considere o modelo de learner aleatório no caso realizável ($h^* \in \mathcal{H}$). Com probabilidade de pelo menos $1-O(1/m)$, $\mathcal{A}_{rand}$ satisfaz o seguinte: (i) aceita a hipótese alvo h^* (linha* **(*)** *de $\mathcal{A}_{rand}$) ; (ii) envia $O(\mathcal{TS}\ \cdot \log(n+m))$ exemplos e (iii) recebe $O(\mathcal{TS}\ \cdot \log(n+m))$ hipóteses por rodada.*

Notamos que sem o limite do número de hipóteses recebidas por rodada o resultado seria direto, já que o *teacher* poderia solicitar infinitas hipóteses para adquirir um conhecimento bastante preciso da classe $\mathcal{H}$ e então recorrer ao algoritmo guloso para *Off-line Set Cover*.

Dado um conjunto de hipóteses $\mathcal{H}' \subseteq \mathcal{H}$, seja $c^*(\mathcal{H}')$ o número máximo de hipóteses de $\mathcal{H}'$ que pode ser coberto por um único exemplo em $\mathcal{X}$. Para provar o Teorema 3.7, contamos com dois lemas cujas provas podem ser encontradas no Apêndice F. A primeira mostra que, com alta probabilidade, o exemplo que cobre o maior número de hipóteses no conjunto $\widetilde{\mathcal{H}}_t$ também cobre um grande número de hipóteses de $\mathcal{H}_t$.

Lema 3.4.2 *Seja $\tilde{e}$ um exemplo que cobre o maior número de hipóteses em $\tilde{\mathcal{H}}_t$ e seja $c_{\tilde{e}}$ o número de hipóteses* cobertas por $\tilde{e}$ *em $\mathcal{H}_t$. Se $|\widetilde{\mathcal{H}}_t| \geq 40\mathcal{TS}\ln m$ então,*

$$\Pr\left(c_{\tilde{e}} \leq \frac{1}{12}c^*(\mathcal{H}_t)\right) \leq \frac{2}{m^4}.$$

O segundo lema fornece um limite superior para o número de rodadas executadas por uma versão aproximativa do algoritmo guloso de *Off-line Set Cover*.

Lema 3.4.3 *Considere $0 < \alpha < 1$ e seja $\mathcal{A}_\alpha$ um algoritmo de teacher qualquer que a cada rodada t envie para o learner um exemplo que cubra pelo menos $\alpha \cdot c^*(\mathcal{H}_t)$ hipóteses de $\mathcal{H}_t$. Então, $\mathcal{A}_\alpha$ executa $O(\frac{1}{\alpha}\mathcal{TS}\ln n) = O(\mathcal{TS}\ln n)$ rodadas antes da única hipótese consistente ser h^*, ou seja, $\mathcal{H}_t = \{h^*\}$.*

Prova do Teorema 3.7. Seja $\alpha = 1/12$ e seja $\hat{i}$ a primeira fase em que $T = 2^{\hat{i}}$ é pelo menos $\mathcal{TS} \cdot \max\{\frac{1}{\alpha}\ln n, 40\ln m\}$. Como as fases $\hat{i}-1$ anteriores enviam $\sum_{j=1}^{\hat{i}-1} 2^j = O(\mathcal{TS}\ln(m+n))$ exemplos e cada uma delas solicita $O(\mathcal{TS}\ln(n+m))$ hipóteses por rodada, basta analisar a fase $\hat{i}$ em diante.

Dizemos que um exemplo é *ruim* para a rodada t se não cobrir $\alpha c^*(\mathcal{H}_t)$ hipóteses de $\mathcal{H}_t$. Devido ao Lema 3.4.2, toda rodada t que ocorre após o início da fase $\hat{i}$ envia um exemplo ruim com probabilidade no máximo $2/m^4$. Assim, segue por *union bound* que um exemplo ruim é enviado durante as $\frac{1}{\alpha}\mathcal{TS}\log n$ primeiras rodadas da fase $\hat{i}$ com probabilidade no máximo $(2\frac{1}{\alpha}\mathcal{TS}\log n)/m^4 \leq 24/m$, em que a desigualdade se mantém porque $\mathcal{TS} \leq m$ e $n \leq |\mathcal{Y}|^m \leq m^m$. Portanto, segue pelo Lema 3.4.3 que, com probabilidade de pelo menos $1 - O(1/m)$, a única hipótese consistente que resta após essas rodadas é h^*. Como cada uma dessas rodadas requer $O(\mathcal{TS}\log(n+m))$ hipóteses, o teorema está provado. □

3.5 *Teaching Sets* Não Redundantes

Dizemos que um *teaching set* X é *redundante* se contém um exemplo redundante, ou seja, um exemplo e tal que $X \setminus \{e\}$ ainda é um *teaching set*. Os algoritmos discutidos até agora podem construir *teaching sets* redundantes. Mostramos que $|X| \cdot (|\mathcal{Y}| - 1)$ rodadas adicionais são suficientes para obter um *teaching set* não redundante a partir de um *teaching set* X com respeito a $(h^*, \mathcal{H})$, em que $\mathcal{Y}$ é o conjunto de rótulos possíveis para os exemplos.

Para isso consideramos um modelo de interação mais geral em que o *teacher* envia um conjunto de exemplos rotulados para o *learner* a cada rodada, e este retorna uma hipótese que comete o menor número de erros nesse conjunto (ignorando os exemplos recebidos nas rodadas anteriores). Diferentemente das seções anteriores, aqui o *teacher* pode enviar um exemplo e com label diferente de $h^*(e)$.

A proposição a seguir fornece uma condição simples para decidir se e é redundante para o *teaching set* X.

Proposição 3.5.1 *Seja X um teaching set para $(\mathcal{H}, h^*)$. Se existe uma hipótese $h \in \mathcal{H}$ com $h(e') = h^*(e')$ para cada $e' \in X \setminus \{e\}$ e $h(e) \neq h^*(e)$, então o exemplo e é não redundante para o conjunto X, e também para qualquer teaching set contido em X. Caso contrário, e é redundante para X.*

Dada essa observação, o algoritmo para obter um *teaching set* não redundante de X é direto: ele escaneia os exemplos em X e para cada $e \in X$ verifica se e é redundante (com respeito ao conjunto de exemplos que não foram retirados de X) ou não; se for, o exemplo é removido de X e a verificação continua sobre os exemplos que ainda não foram testados. Para verificar se e é redundante, o *teacher* interage com o *learner* em $|\mathcal{Y}| - 1$ rodadas testando a existência de um rótulo $y \neq h^*(e)$ para o qual o *learner* retorna uma hipótese consistente com o conjunto rotulado de exemplos $\mathcal{D}_y = \{(e', h^*(e')) \mid e' \in X \setminus e\} \cup \{(e, y)\}$. Se tal rótulo não existir, o algoritmo conclui que e é redundante. Evidentemente, o número total de rodadas é no máximo $|X| \cdot (|\mathcal{Y}| - 1)$.

3.5.1 Desempenho do Algoritmo em Simulações com Dados Sintéticos

Para avaliar o impacto desse algoritmo, fizemos algumas simulações usando dados sintéticos. Medimos a redução no número de exemplos que o método consegue sobre os *teaching sets* obtidos quando o *teacher* é `OSCT` e o

learner retorna sempre uma hipótese aleatória consistente com os exemplos recebidos até o momento.

A seguir, descrevemos os dados empregados em nossos experimentos. Fixamos o número de hipóteses $n = 30.001$ e variamos o número de exemplos m no conjunto $\{100, 500, 2500, 12.500, 75.000\}$. Usamos apenas hipóteses binárias, ou seja, $|\mathcal{Y}| = 2$. Para cada experimento, a classe de hipóteses $\mathcal{H}$ consiste em uma hipótese h^* que não comete erros e também 5 grupos de 6.000 hipóteses cada, em que todas as hipóteses do i-ésimo grupo falham em $(2i)\%$ de seus exemplos. Focamos em hipóteses que falham em um número relativamente pequeno de exemplos, pois aquelas que cometem muitos erros são mais fáceis de descartar.

Para gerar essas hipóteses, utilizamos um parâmetro $\alpha \in [0, 1]$ que define o número de *exemplos difíceis* e um parâmetro $\beta \in [0, 1]$ que define o número de erros associados com os exemplos difíceis em cada hipótese. Mais especificamente, para gerar uma classe de hipóteses $\mathcal{H}(m, \alpha, \beta)$, primeiro fixamos os $m\alpha$ exemplos que são difíceis. Então, o conjunto de exemplos errados para uma hipótese que falha em $x\%$ de seus exemplos é construído selecionando aleatoriamente $(x/100) \cdot m \cdot \beta\%$ exemplos do conjunto difícil e selecionando aleatoriamente o restante do conjunto não difícil.

Consideramos 35 combinações de parâmetros (α, β, m)

$$\{\{0.1, 0.2, 0.3\} \times \{0.25, 0.5\} \times \{100, 500, 2500, 12.500, 75.000\}\} \cup \{\{0\} \times \{0\} \times \{100, 500, 2500, 12.500, 75.000\}\}$$

Para cada combinação geramos 10 classes de hipóteses e, então, medimos a redução no tamanho do *teaching set* obtido por `OSCT` quando o algoritmo discutido na Seção 3.5 é empregado para remover exemplos redundantes. Em média, ao longo das 350 execuções, 38% dos exemplos foram removidos e tivemos casos em que essa redução foi maior que 50%.

O código empregado para esta simulação está disponível em <https://github.com/sfilhofreitas/TeachingWithLimitedKnowledge>

3.6 Experimentos Computacionais

Embora nosso trabalho seja principalmente teórico, realizamos experimentos para entender como nosso *teacher* `OSCT` se compara em relação a um *teacher* não interativo sobre *datasets* reais.

O *teacher* não interativo, denotado por `NIT`, recebe um inteiro ℓ e então envia para o *learner* ℓ exemplos selecionados aleatoriamente. Comparamos o

número de exemplos que `OSCT` e `NIT` precisam enviar para atingir um certo nível de acurácia. Para nossa avaliação, usamos Random Forest e Light Gradient Boosting Machine (LGBM) como *learners* e conduzimos experimentos em 12 conjuntos de dados: `mnist` e 11 outros do repositório UCI (`mushroom`, `avila`, `bank_marketing`, `car`, `Credit Card`, `Firm_Teacher`, `crowdsourced`, `Electrical_grid`, `HTRU`, `nursery` e `Sensorless_drive`).

A tabela 3.1 mostra o número de exemplos (relativo ao tamanho do *dataset* completo) requerido por cada par *teacher-learner* para obter uma acurácia sobre todo o *dataset* maior que $z\%$ do obtido quando o *learner* é treinado/testado em todo o conjunto de dados. Cada entrada numérica dessa tabela é uma média de 12 valores, em que cada um deles corresponde a um conjunto de dados distinto. Mais detalhes são apresentados no Apêndice G.

Tabela 3.1: Porcentagem do tamanho do conjunto de dados completo exigido por cada *teacher-learner* para atingir uma acurácia maior que $z\%$ (com $z \in \{90, 95, 99\}$) daquela alcançada pelo *learner* quando ele é treinado e testado no conjunto de dados completo.

teacher-learner	90%	95%	99%
`OSCT`-Random Forest	10.1%	14.7%	20.6 %
`NIT`-Random Forest	14.7%	30.4%	59.9%
`OSCT`-LGBM	2.8 %	5.7 %	8.9 %
`NIT`-LGBM	3.4 %	7.8 %	28.0 %

Os resultados fornecem evidências de que `OSCT` requer significativamente menos exemplos do que `NIT` (por exemplo, um fator de 3 para atingir uma acurácia de 99%). Além disso, uma observação interessante é que a vantagem de `OSCT` aumenta à medida que o nível de acurácia solicitada aumenta. Uma explicação razoável é que, quando pouco se sabe (baixa acurácia em nosso cenário), a maioria dos exemplos é útil (portanto, a amostragem aleatória também funciona bem); entretanto, quando um determinado nível de conhecimento já foi alcançado, exemplos mais específicos, como os fornecidos por `OSCT`, são necessários para aumentá-lo ainda mais.

4
O problema de Aprendizado com Restrição de Tempo

Neste capítulo introduzimos e tratamos do problema de Aprendizado com Restrição de Tempo, no qual consideramos um cenário com larga escala de dados rotulados disponíveis e um limite de tempo para treinar um determinado *learner* usando esses dados. Mais precisamente, estamos interessados no caso em que treinar utilizando todas as amostras extrapola o limite de tempo. Diante dessa realidade, propomos e discutimos um método que se utiliza dos princípios de *Machine Teaching* para resolver o problema em questão. Vale ressaltar que estes resultados foram recentemente aceitos para publicação em forma de artigo no periódico científico *Pattern Recognition* (FREITAS et al., 2023).

4.1
Introdução

Com a atual explosão de dados sendo gerados e coletados, há um número crescente de aplicações de aprendizado de máquina supervisionado em que um grande número de exemplos rotulados está disponível, mas os recursos computacionais para treinar um modelo são limitados, a ponto de nem todos os exemplos poderem ser usados para treinamento.

Isso pode ocorrer quando alguém tem um orçamento financeiro limitado para treinar um modelo em um serviço de aprendizado de máquina baseado em nuvem, como o Azure ML da Microsoft. Nesse caso, o orçamento limitado traduz-se naturalmente num limite de tempo de treinamento. Em outro cenário, muito atual, pode-se estabelecer um limite no tempo de treinamento para reduzir seu impacto ambiental. Existem também aplicações, como anúncios de publicidade e aprendizado a partir de *logs* de pesquisas, em que o número de exemplos rotulados é enorme e é necessário treinar o modelo repetidas vezes para acompanhar as novas distribuições de comportamentos de usuários, de modo que o treinamento com todos os exemplos rotulados não seja viável.

Em uma situação mais concreta, considere o caso em que nosso orçamento financeiro só nos permite utilizar 4 horas de um serviço de computação em nuvem, mas queremos treinar uma *Random Forest* utilizando um grande *dataset* (com milhões ou até mesmo bilhões de exemplos). É possível que esse tempo não seja suficiente para treinar nosso modelo devido a grande quantidade de dados. Sendo assim, como podemos proceder?

Essa é a motivação por trás da tarefa de Aprendizado com Restrição de

Tempo (TCL[1]), que é o foco do nosso trabalho (especificamente para tarefas de classificação). Consideramos a seguinte formulação para TCL:

Entrada: Um *dataset* D de exemplos rotulados amostrados a partir de uma distribuição desconhecida μ, um *learner* L e um tempo limite T.

Saída: O modelo de classificação de maior acurácia, com respeito a μ, que pode ser construído por L dentro do tempo limite T.

TCL admite duas variações: uma na qual temos uma quantidade de informações consideráveis sobre o *learner* (como, por exemplo, sua classe de hipóteses e seu tempo de treinar/classificar exemplos) e outra em que temos informações bem limitadas (*black-box learners*). Em particular, estamos interessados na última opção por ter uma maior abrangência de aplicação. Dada a falta de informação sobre o *learner*, surge a seguinte questão:

É possível superar a amostragem aleatória (ou alguma variação natural) para TCL?

Para dar uma resposta positiva a essa questão, abordamos a tarefa de TCL via *framework* de *Machine Teaching* (SHINOHARA, 1991; ZHU et al., 2018), no qual um *teacher* e um *learner* interagem em várias rodadas e em cada uma delas o *teacher* envia exemplos selecionados para o *learner*, que retorna um modelo treinado. Nosso objetivo é desenvolver um *teacher* que guie o *learner* para o melhor classificador, com respeito a μ, da forma mais eficiente possível em termos de tempo. Sob esse ponto de vista, podemos ver TCL como um problema de *Machine Teaching* com restrição de tempo.

Embora a maioria dos trabalhos iniciais sobre *Machine Teaching* (SHINOHARA, 1991; GOLDMAN; KEARNS, 1995) suponha que o *teacher* tem conhecimento significativo sobre o *learner*, houve vários avanços recentes no caso de interesse em que o conhecimento sobre o *learner* é limitado (MELO; GUERRA; LOPES, 2018; LIU et al., 2018; DASGUPTA et al., 2019; CICALESE et al., 2020; DEVIDZE et al., 2020).

No entanto, esses métodos não abordam exatamente a tarefa TCL e, em vez disso, concentram-se em minimizar o número de amostras enviadas ao *learner*, ou seja, o tamanho do *teaching set*. Embora o tamanho do *dataset* e a complexidade do tempo estejam relacionados, existem fatores, como o tempo de treinamento e a quantidade de interação com o *learner*, que devem ser considerados quando o tempo é levado em consideração. Esse aspecto é ilustrado no início da Seção 4.4. De fato, métodos que desconsideram esses fatores não são adequados para TCL, conforme indicado por nossos experimentos. Assim,

[1]Abreviação do nome em inglês: *Time-Constrained Learning.*

novos métodos devem ser desenvolvidos para lidar adequadamente com a tarefa de Aprendizado com Restrição de Tempo.

4.2 Nossas contribuições

Nossa principal contribuição é o algoritmo *Time Constrained Teacher* (`TCT`) para a tarefa de Aprendizado com Restrição de Tempo, projetado com base em princípios/ideias bem estabelecidos da Teoria da Aprendizagem Computacional e *Machine Teaching*. Mostramos que nosso algoritmo `TCT` supera métodos alternativos de ensino/treinamento: (1) Treinamento sobre *batches* de amostras aleatórias, até que o limite de tempo seja atingido; (2) O algoritmo estado da arte de *Machine Teaching* para *black-box learners* (DASGUPTA et al., 2019; CICALESE et al., 2020) e (3) *Stochastic Gradient Descent* (quando aplicável).

Mais precisamente, nossos experimentos em 20 conjuntos de dados e usando 5 modelos/*learners* populares mostram que nosso algoritmo `TCT` supera consistentemente seus concorrentes (Seção 4.5). Em particular, como pode ser visto nas Figuras 4.1 e 4.3, `TCT` supera os métodos alternativos (2) e (3) para todos os *learners* (quando aplicável) por uma ampla margem quando observamos a acurácia média geral durante o período de treinamento. O *baseline* (1), nomeado `Double`, mostrou-se mais competitivo. Mesmo assim, nosso método foi superior em 33 das 42 configurações em que um método supera o outro com 95% de confiança, apesar de treinar com 15% menos exemplos (em média). Vale ressaltar que em alguns casos nosso método foi superior não apenas ao *baseline* mais competitivo, mas também ao treinamento usando todo o conjunto de dados disponível.

Uma característica positiva do nosso algoritmo é a sua simplicidade: ele emprega apenas um parâmetro que é fácil de definir e não requer nenhuma informação sobre o *learner*. Portanto, acreditamos que ele poderia ser facilmente implementado como um módulo em bibliotecas de aprendizado de máquina para abordar aprendizado com restrição de tempo - o usuário fornece o limite de tempo e o *learner* com o qual se sente confortável e, em seguida, `TCT` conclui o trabalho.

Embora nosso trabalho seja principalmente prático, também mostramos que uma versão simplificada do `TCT` tem garantias comprováveis (Seção 4.6). Sob suposições razoáveis, o tempo que nosso algoritmo leva para atingir uma certa acurácia nunca é muito maior do que o tempo que leva o *batch teacher* (que envia um *batch* de exemplos de tamanho ótimo) para obter uma acurácia semelhante (Teorema 4.1). Além disso, para aprender uma função

de *threshold*, um problema canônico em *Active Learning* e *Machine Teaching*, nosso algoritmo é "quase" exponencialmente mais rápido, apesar de não ser desenvolvido (especificamente) para essa classe de hipóteses (Teorema 4.2).

Também apresentamos, no Apêndice A, uma contribuição adicional para o problema de TCL: um *framework* Python que pode ser facilmente instalado e é compatível com modelos de aprendizado fornecidos pela biblioteca scikit-learn.

4.3 Trabalhos Relacionados

Learners que permitem um treinamento *online*, como é o caso de *Stochastic Gradient Descent* (SGD), têm sido utilizados com frequência em aprendizado de máquina em larga escala de dados (BOTTOU; BOUSQUET, 2007). Embora esses *learners* possam ser utilizados para o problema de TCL, não estamos cientes de trabalhos que resolvam nosso problema para *black-box learners*. No entanto, TCL é mencionada como uma aplicação em potencial para o *teacher* desenvolvido em (DASGUPTA et al., 2019). Já (DU, 2011) menciona a possibilidade de minimizar o tempo de treinamento em vez do tamanho do *teaching set*. Ainda assim, ambos os trabalhos lidam com a tarefa de minimizar a quantidade de exemplos enviados ao *learner*.

Por outro lado, podemos encontrar na literatura uma série de trabalhos para o problema de *Machine Teaching* em sua forma tradicional, em que o objetivo é minimizar o *teaching set*. Nesses trabalhos podemos destacar alguns diferentes cenários adotados como, por exemplo, em (DU, 2011; LIU et al., 2018; CHEN et al., 2018), que consideram que exemplos são selecionados sequencialmente, enquanto (SINGLA et al., 2014; MA et al., 2018) consideram uma seleção por *batch* de exemplos. Existem também *teachers* que supõem dispor de informações consideráveis sobre o *learner* (SINGLA et al., 2014; LIU et al., 2017), e os que têm informações limitadas (DASGUPTA et al., 2019; CICALESE et al., 2020; DU, 2011). Neste trabalho, consideramos o cenário em que o *teacher* seleciona um *batch* de exemplos em cada iteração e não assumimos qualquer conhecimento do *teacher* sobre o *learner*.

Trabalhos clássicos (SHINOHARA, 1991; GOLDMAN; KEARNS, 1995) de *Machine Teaching* consideram um cenário no qual o *teacher* envia para o *learner*, em uma única vez, uma amostra do conjunto de dados para que o *learner* retorne a hipótese alvo. Em nosso trabalho, consideramos o cenário iterativo (*Iterative Machine Teaching*) — adotado em (LIU et al., 2017) e que tem sido o foco de trabalhos mais recentes (CHEN et al., 2018; LIU et al., 2018; DASGUPTA et al., 2019; POURKAMALI-ANARAKI; BENNETTE, 2021) —

no qual *teacher* e *learner* interagem em múltiplas rodadas.

Muitos desses métodos de *Machine Teaching* não são facilmente adaptáveis para o nosso problema, pois necessitam de informações sobre a classe de hipóteses do *learner* ou seu algoritmo de treinamento, o que foge do cenário que consideramos para o problema de TCL. Por exemplo, Liu et al. (2018) consideram *learners* que utilizam SGD para treinamento. Uma exceção é o método `OSCT`, definido em (DASGUPTA et al., 2019) e que aprimoramos em uma primeira etapa desta pesquisa (Capítulo 3)[(CICALESE et al., 2020)], que considera um cenário no qual o *teacher* tem informações bem limitadas sobre o *learner*.

`OSCT`, assim como `TCT`, obtém um modelo treinado interagindo com um *learner* em várias rodadas. Em cada rodada, ele seleciona alguns exemplos nos quais o modelo atual falha e os adiciona ao conjunto de treinamento que é usado pelo *learner* para induzir um modelo atualizado. Como `OSCT` foi projetado para minimizar o tamanho do *teaching set* e não considera o tempo total de treinamento, ele difere do `TCT` em dois fatores: i) enquanto `TCT` dobra o tamanho do conjunto de treinamento a cada rodada, `OSCT` geralmente adiciona poucos exemplos a esse conjunto e, consequentemente, seu modelo final usaria apenas um pequeno subconjunto de exemplos em uma configuração com restrição de tempo (ver Exemplo 1); (ii) ao contrário do `TCT`, em cada rodada `OSCT` classifica com o modelo atual todos os exemplos ainda não selecionados, o que pode ser computacionalmente caro no cenário de dados de larga escala considerado neste trabalho. Portanto, no aprendizado *com restrição de tempo*, `OSCT` tem desempenho muito pior do que nosso algoritmo `TCT`, e é ainda pior do que amostragem aleatória, como mostramos em nossos experimentos.

Nosso cenário de Aprendizado com Restrição de Tempo também pode ser relacionado ao paradigma tradicional de *Active Learning* (SETTLES, 2009). Por um lado, os cenários considerados são bastante diferentes: em *Active Learning*, exemplos rotulados são restritos, enquanto em nosso cenário são abundantes. Por outro lado, nós não podemos utilizar todos os exemplos rotulados disponíveis e, portanto, precisamos escolher de forma criteriosa os que queremos utilizar para treinamento. Essa escolha criteriosa de exemplos é a principal característica de *Active Learning*, que seleciona os exemplos mais informativos. Portanto, poderíamos utilizar estratégias de *Active Learning* para selecionar exemplos para o problema de TCL. Um problema disso é que estratégias clássicas, como *Uncertainty Sampling*, podem requerer imposições sobre o *learner* (por exemplo, a capacidade de, ao classificar um exemplo, produzir a probabilidade dele pertencer a cada classe), que não é alinhado com

nosso objetivo de treinar *black-box learners*. Uma estratégia que não necessita desse tipo de imposição sobre o *learner* é *Query by Committee* (SEUNG; OPPER; SOMPOLINSKY, 1992), mas essa estratégia não é praticável em nosso cenário com tempo restrito.

4.4 O Algoritmo *Time Constrained Teacher*

Nesta seção, descrevemos nosso algoritmo `TCT`. Sua concepção leva em conta os seguintes princípios:

P1 Quanto maior o número de exemplos, melhor o aprendizado;

P2 Exemplos em que um *learner* falha são mais úteis para melhorar sua acurácia do que aqueles em que ele é bem-sucedido;

P3 Exemplos selecionados para treinar o *learner* devem seguir aproximadamente a distribuição de probabilidade μ da qual eles foram amostrados.

O Princípio 1 é muito natural e justificado por estudos empíricos e garantias estatísticas padrões. No entanto, para obter um modelo treinado em um grande número de exemplos e dentro de um limite de tempo T, é importante não enviar um número muito pequeno de exemplos em cada rodada, conforme ilustrado abaixo.

Exemplo 4.4.1 *Considere um learner com tempo de execução quadrático, ou seja, $\Theta(m^2)$ unidades de tempo são necessárias para treinar um modelo quando m exemplos estão disponíveis. Se o teacher sempre enviar um único exemplo por rodada, então o maior modelo terá $O(\sqrt[3]{T})$ exemplos, em que T é o limite de tempo. Por outro lado, se o teacher sempre dobrar o tamanho do conjunto de exemplos enviados ao learner, então o maior modelo terá $\Omega(\sqrt{T})$ exemplos.*

Notamos que, em contraste com essa ideia, trabalhos anteriores sobre *Machine Teaching* que focam em minimizar o tamanho do *teaching set* sugerem a adição de poucos exemplos por rodada (DU, 2011; DASGUPTA et al., 2019; CICALESE et al., 2020).

O Princípio 2 é motivado pelo aprendizado humano e também por trabalhos sobre *Machine Teaching* (DASGUPTA et al., 2019; CICALESE et al., 2020) e *boosting* (SCHAPIRE, 1999), em que exemplos errados têm maior prioridade do que aqueles nos quais o *learner* é bem sucedido. Embora exemplos errados sejam úteis, sua aquisição pode ser cara em termos de tempo computacional, pois encontrá-los pode exigir a classificação de um grande número de exemplos. Assim, precisamos balancear entre os benefícios de ter

exemplos errados e os custos computacionais para obtê-los. Além disso, temos que adicionar exemplos errados com parcimônia, caso contrário, podemos construir modelos usando um conjunto de exemplos que não é representativo da população real e, consequentemente, com desempenho potencialmente ruim em dados não vistos. Isso é capturado no Princípio 3. De fato, problemas com o uso de amostras tendenciosas para aprendizado estão bem documentados em *Active Learning* (DASGUPTA, 2009).

Com base nessas observações, projetamos `TCT`, apresentado no Algoritmo 1. Ele recebe um *learner* L, um *pool* de exemplos P e um limite de tempo T; ele também recebe os parâmetros m_0 (número de exemplos para o primeiro modelo) e $\alpha \in [0, 1]$ (que define a razão entre exemplos errados e exemplos aleatórios fornecidos ao *learner* em cada rodada). Para facilitar a apresentação, supomos que P é suficientemente grande para que sempre seja possível obter dele exemplos não selecionados.

O algoritmo `TCT` funciona de forma iterativa, dobrando o tamanho do conjunto de treinamento S a cada rodada. Mais precisamente, em cada iteração ele envia o conjunto de treinamento atual S para o *learner* para obter um modelo treinado M, e então seleciona $|S|$ novos exemplos (`Rand`$\cup$`Wrg` no código) a serem adicionados ao conjunto de treinamento (Linha 13). Em alto nível, a ideia é que uma fração α dos novos exemplos seja composta por amostras nas quais o modelo atual M comete um erro (essencialmente o conjunto `Wrg`), e a fração restante $(1 - \alpha)$ contenha novos exemplos aleatórios (o conjunto `Rand`). Assim, o parâmetro α permite um *trade-off* entre seguir os Princípios 2 e 3 acima. Construir ingenuamente o conjunto de exemplos errados `Wrg` requer uma busca linear em potencialmente todo o conjunto de exemplos P, o que pode ser muito demorado. Em vez disso, amostramos aleatoriamente um conjunto $A_1 \cup A_2$ de tamanho apropriado para que $A_2 \cup (A_1 \setminus$ `Rand`$)$ tenha (aproximadamente) $\alpha|S|$ exemplos errados, que são então adicionados a `Wrg` (Linhas 11 e 12). O número de amostras necessárias em que se espera obter $\alpha|S|$ exemplos errados depende da acurácia do modelo M. Então, obtemos o conjunto inicial de amostras A_1, utilizamos A_1 para obter uma estimativa acc_1 da acurácia de M, e usamos essa estimativa para calcular quantas exemplos adicionais A_2 devem ser amostrados (Linhas 6-8). Observe que se acc_1 é de fato a acurácia de M (nos exemplos não utilizados no *pool*), espera-se que o conjunto $A_2 \cup (A_1 \setminus$ `Rand`$)$, que tem tamanho $\alpha|S| + \frac{\alpha|S|acc_1}{1-acc_1}$, contenha $(1 - acc_1) \cdot (\alpha|S| + \frac{\alpha|S|acc_1}{1-acc_1}) = \alpha|S|$ exemplos em que M está errado, conforme desejado.

Finalmente, as amostras A_1 e A_2 são reutilizadas para obter `CurrentEstimator`, uma estimativa da acurácia do modelo M (Linhas 7,

9 e 15-16), e o método decide se deseja manter esse classificador (Linhas 17-19). Notamos que `CurrentEstimator` corresponde ao limite inferior do intervalo de 95% de confiança da acurácia esperada *acc* (Seção 5.2 de (MITCHELL, 1997)) e é usado, em vez da acurácia estimada, porque nas primeiras rodadas são utilizados poucos exemplos e, como consequência, a variância da estimativa é alta.

Algoritmo 1: `TCT` (m_0: valor inteiro; L: Learner; α: valor real; P: *pool* de exemplos; T: tempo limite)

1 $S \leftarrow$ *conjunto de* m_0 *exemplos aleatórios de* P
2 `BestEstimator` $\leftarrow -\infty$
3 **enquanto** `ElapsedTime` $\leq T$ **faça**
4 $M \leftarrow$ *modelo treinado pelo learner* L *no conjunto* S
5 *# Construindo o conjunto de exemplos* S *para a próxima rodada*
6 $A_1 \leftarrow$ *conjunto de* $|S|$ *exemplos aleatórios de* P *que ainda não foram selecionados*
7 $acc_1 \leftarrow$ *Classificar*(M, A_1)
8 $A_2 \leftarrow$ *conjunto de* $\alpha|S|acc_1/(1-acc_1)$ *exemplos aleatórios de* P *que ainda não foram selecionados*
9 $acc_2 \leftarrow$ *Classificar*(M, A_2)
10 `Rand` $\leftarrow$ *conjunto de* $(1-\alpha)|S|$ *exemplos aleatórios de* A_1
11 $V \leftarrow$ *lista de exemplos em* $A_2 \cup (A_1 \setminus$ `Rand`$)$ *com os errados (com respeito a* M*) aparecendo antes dos corretos.*
12 `Wrg` $\leftarrow \alpha|S|$ *primeiros exemplos de* V
13 $S \leftarrow S \cup ($`Rand` $\cup$ `Wrg`$)$
14 *# Estimando a acurácia do modelo atual, salve se for o melhor*
15 $acc \leftarrow (acc_1|A_1| + acc_2|A_2|)/(|A_1| + |A_2|)$
16 `CurrentEstimator` $\leftarrow acc - 1.96\sqrt{\frac{acc(1-acc)}{|A_1|+|A_2|}}$
17 **se** `CurrentEstimator` $>$ `BestEstimator` AND `ElapsedTime` $\leq T$ **então**
18 `BestModel` $\leftarrow M$
19 `BestEstimator` $\leftarrow$ `CurrentEstimator`
20 Retorne `BestModel`

Algumas limitações/observações sobre nosso método:

– Se o parâmetro α for muito pequeno, então `TCT` torna-se semelhante à amostragem aleatória pura. Por outro lado, se α for grande, o *learner* é orientado a aprender um modelo que segue uma distribuição que pode ser significativamente diferente da real, o que não está de acordo com o Princípio 3. De fato, apresentamos um exemplo na Seção I.1 em que usar um α grande é problemático. Além disso, um grande valor de α pode ter um impacto negativo no tempo de execução devido às Linhas 8 e 9. Os

resultados de nossos experimentos sugerem que $\alpha = 0.2$ funciona bem na prática, e também que o método é robusto para variações moderadas desse parâmetro.

– `TCT` é mais adequado para o cenário típico em que o tempo de classificação por exemplo é (muito) menor que o tempo de treinamento por exemplo. Para cenários em que essa propriedade não é válida, `TCT` pode gastar muito tempo classificando exemplos e, portanto, terminar com um modelo treinado em um conjunto relativamente pequeno.

– Assumimos que o *pool* P é grande o suficiente para que possamos sempre amostrar exemplos que não foram considerados até agora (linhas 6 e 8). Na prática, isso não ocorre necessariamente. Nesse caso, quando `TCT` atinge o ponto em que todos os exemplos já foram classificados, ele passa a usar exemplos que ainda não foram adicionados ao conjunto S.

4.5 Estudo Experimental

Em nosso primeiro conjunto de experimentos, comparamos o algoritmo `TCT` com dois outros *teachers*: `Double` e `OSCT` (definido no Capítulo 3). Eles são brevemente descritos abaixo.

– `Double` é um *teacher* que a cada rodada i envia para o *learner* $m_0 2^i$ novos exemplos selecionados aleatoriamente, em que m_0 é o número de exemplos usados para treinar o primeiro modelo. Em seguida, o *learner* retorna um modelo treinado em todos os exemplos recebidos até o momento e, então, inicia-se a rodada $i + 1$. O modelo retornado por `Double` é o último construído dentro do limite de tempo dado;

– `OSCT` é o *teacher* inspirado em (DASGUPTA et al., 2019) e refinado na primeira etapa da nossa pesquisa (CICALESE et al., 2020). Ele mantém um peso para cada exemplo no conjunto de treinamento e, a cada rodada, recebe um novo modelo (hipótese) do *learner* e o utiliza para classificar todos os exemplos do *dataset*. Em seguida, dobra repetidamente os pesos dos exemplos errados até que sua soma exceda 1. Nesse ponto, `OSCT` amostra $O(\log n)$ exemplos seguindo uma distribuição induzida por esses pesos, em que n é uma estimativa do número de hipóteses eficazes na classe do *learner*. O *learner* recebe essa amostra de exemplos e a usa para atualizar seu modelo atual.

`Double` pode ser visto como uma adaptação de amostragem aleatória para a tarefa TCL e, assim, entendemos que é um *baseline* muito natural.

`OSCT` é um método recente para ensinar *black-box learners* que se concentra em minimizar o tamanho do *teaching set*. Em nossos experimentos empregamos a implementação discutida no Capítulo 3 (CICALESE et al., 2020).

Também comparamos `TCT` com o treinamento baseado em *Stochastic Gradient Descent* (`SGD`), uma estratégia amplamente utilizada para treinamento *online*. Para isso, utilizamos a classe `SGDClassifier` da biblioteca `Scikit-Learn`, com seus parâmetros padrões.

Para comparar `TCT` com `Double` e `OSCT`, consideramos 5 *learners*: `LGBM`, `Random Forest`, `Decision Tree`, `SVM` e `Logistic Regression`. Em nossa comparação com `SGD`, consideramos apenas os dois últimos *learners*, pois não é claro como treinar (diretamente) os outros via `SGD`.

`LGBM` pertence à família dos métodos *Gradient Boosting* e é muito popular em plataformas de competições como o Kaggle. `Random Forest` também é amplamente utilizada. Selecionamos `Árvores de Decisão` com pequena profundidade como representantes de métodos interpretáveis, enquanto `SVM` e `Logistic Regression` foram selecionados como representantes de classificadores lineares.

Nossos *learners* foram implementados usando Python, versão 3.8.5, com as bibliotecas numpy (1.20.1), pandas (1.2.2), lightgbm (3.1.1), scikit-learn (0.24.1), scipy (1.6.1). Usamos os parâmetros padrões para todos os *learners*, exceto para `Decision Trees`, em que definimos $min\_samples\_split = 30$ e $max\_depth = 5$ para construir árvores interpretáveis; e para `Random Forest`, em que definimos $min\_samples\_split = 30$ para evitar um tempo de execução muito longo e, consequentemente, uma limitação em nosso estudo experimental. Para obter classificadores `SVM` e `Logistic Regression` via `SGD`, definimos o parâmetro `loss` da classe `SGDClassifier` para `hinge` e `log`, respectivamente.

Consideramos 20 *datasets* em nossos experimentos, cujas principais características são mostradas na Tabela 4.1. Os experimentos da Seção 4.5.1-4.5.3 foram executados usando um processador Core i9-7900X 3.3GHz, com 128GB RAM DDR4 e Windows 10, enquanto os da Seção 4.5.4 foram executados com as seguintes configurações: Core i7-4790 3.60GHz, 32GB RAM DDR3, Ubuntu 20.04.3 LTS. Para cronometragem usamos o método `timeit.default_timet` da biblioteca `timeit`. Nosso código pode ser encontrado em <https://github.com/sfilhofreitas/TimeConstainedLearning>. Mais detalhes sobre os *datasets* e os *learners* podem ser encontrados em H.

Para cada combinação de *dataset* $\mathcal{D}$ e Learner $\mathcal{L}$, calculamos o tempo $t_{\mathcal{D},\mathcal{L}}$ necessário para $\mathcal{L}$ construir um modelo de classificação treinando em todo o *dataset* $\mathcal{D}$ (média de 4 execuções) e consideramos os pares $(\mathcal{D}, \mathcal{L})$ para os quais

Tabela 4.1: *Datasets*. m: tamanho do conjunto de treino, d: número de atributos; k: número de classes

Dataset	**m**	**d**	**k**
vehicle_sensIT	68969	100	2
MiniBooNE	91044	50	2
SantanderCustomer	140000	200	2
BNG_spambase	699993	171	2
BNG_spectf_test	700000	44	2
Diabetes130US	71236	2518	3
BNG_wine	700000	13	3
jannis	58613	54	4
BNG_eucalyptus	700000	95	5
BNG_satimage	700000	36	6
covtype	406708	54	7
volkert	40817	180	10
cifar_10	42000	3072	10
mnist	60000	784	10
BNG_mfeat_fourier	700000	76	10
poker_hand	1000000	85	10
Sensorless_drive	40956	48	11
BNG_letter_5000_1	700000	16	26
GTSRB-HueHist	36287	256	43
aloi	75600	128	1000

$t_{\mathcal{D},\mathcal{L}}$ é de pelo menos 10 segundos. Definimos $Valid(\mathcal{L}) := \{\mathcal{D} \mid t_{\mathcal{D},\mathcal{L}} > 10\}$.

Para cada combinação válida $(\mathcal{D}, \mathcal{L})$, executamos cada um dos *teachers* 4 vezes, com 4 *seeds* diferentes, para construir um modelo de classificação dentro do limite de tempo $t_{\mathcal{D},\mathcal{L}}$. A análise reportada nesta seção emprega a média dessas 4 execuções. Durante o processo de treinamento, sempre que era obtido um novo modelo, avaliávamos o mesmo no conjunto de teste (pausando o tempo), o que nos permitiu computar as curvas de evolução que serão apresentadas posteriormente. Para todos os experimentos, definimos m_0 – o tamanho do primeiro conjunto de exemplos enviado ao aluno – como $0,5\%$ do tamanho *dataset* completo. Na prática, m_0 deve ser definido como o maior valor para o qual estamos confiantes de que um conjunto de treinamento com m_0 exemplos rotulados pode ser treinado dentro do limite de tempo.

4.5.1 Comparações com `OSCT` e `Double`

A Figura 4.1 mostra 5 imagens, cada uma correspondendo a uma comparação entre `TCT` ($\alpha = 0.2$) e `OSCT` para cada um dos *learners*. O eixo horizontal corresponde ao limite de tempo normalizado $t \in [0, 1]$ (como uma fração de $t_{\mathcal{D},\mathcal{L}}$), e o eixo vertical corresponde à acurácia (média) nos conjuntos de teste. Mais precisamente, para um *teacher* $\mathcal{T}$ e um *learner* $\mathcal{L}$, o ponto associado ao tempo normalizado $t \in [0, 1]$ é dado por

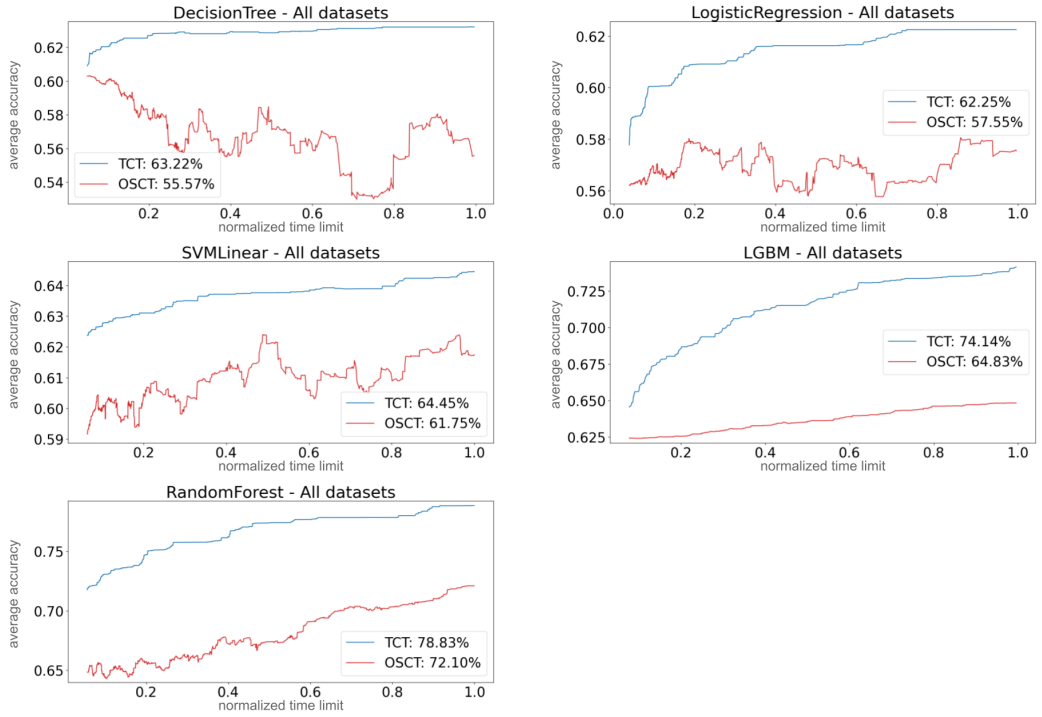

Figura 4.1: Acurácias médias no conjunto de teste ao longo do tempo normalizado para `TCT` e `OSCT`. Os números próximos aos rótulos dos *teachers* são suas acurácias médias no final do tempo limite normalizado ($t = 1$). O palpite inicial de `OSCT` para n é 2.

$$\sum_{\mathcal{D} \in Valid(\mathcal{L})} \frac{acc(\mathcal{T}, \mathcal{L}, \mathcal{D}, t \cdot t_{\mathcal{D},\mathcal{L}})}{|Valid(\mathcal{L})|}, \tag{4-1}$$

em que $acc(\mathcal{T}, \mathcal{L}, \mathcal{D}, t \cdot t_{\mathcal{D},\mathcal{L}})$ é a acurácia média, calculada sobre 4 execuções, do modelo obtido pelo *teacher* $\mathcal{T}$, com o *learner* $\mathcal{L}$, para o *dataset* $\mathcal{D}$ quando o limite de tempo para treinamento é $t \cdot t_{\mathcal{D},\mathcal{L}}$.

Como exemplo, considere um *dataset* em que o treinamento de um SVM com todos os exemplos levou 1000 segundos. Então, a Figura 4.1 "SVMLinear" no eixo x igual a 0,6, por exemplo, mostra a acurácia dos algoritmos `TCT` e `OSCT` para $0{,}6 \cdot 1000 = 600$ segundos de tempo de execução (na verdade, essa figura mostra a média da acurácia em todos *datasets*).

Observamos que `TCT` apresenta uma grande vantagem em comparação com `OSCT`. Uma das razões é que `OSCT` não consegue construir modelos em grandes conjuntos de treinamento, uma vez que ele adiciona apenas alguns exemplos por rodada e gasta um tempo não negligenciável classificando exemplos. Tentamos variações de `OSCT` para melhorar sua acurácia (ver Apêndice H.3), mas os resultados não mudaram significativamente.

A Figura 4.2 mostra uma comparação semelhante em que é usado `Double`, em vez de `OSCT`. Observamos que `TCT` supera `Double` para todos os *learners*,

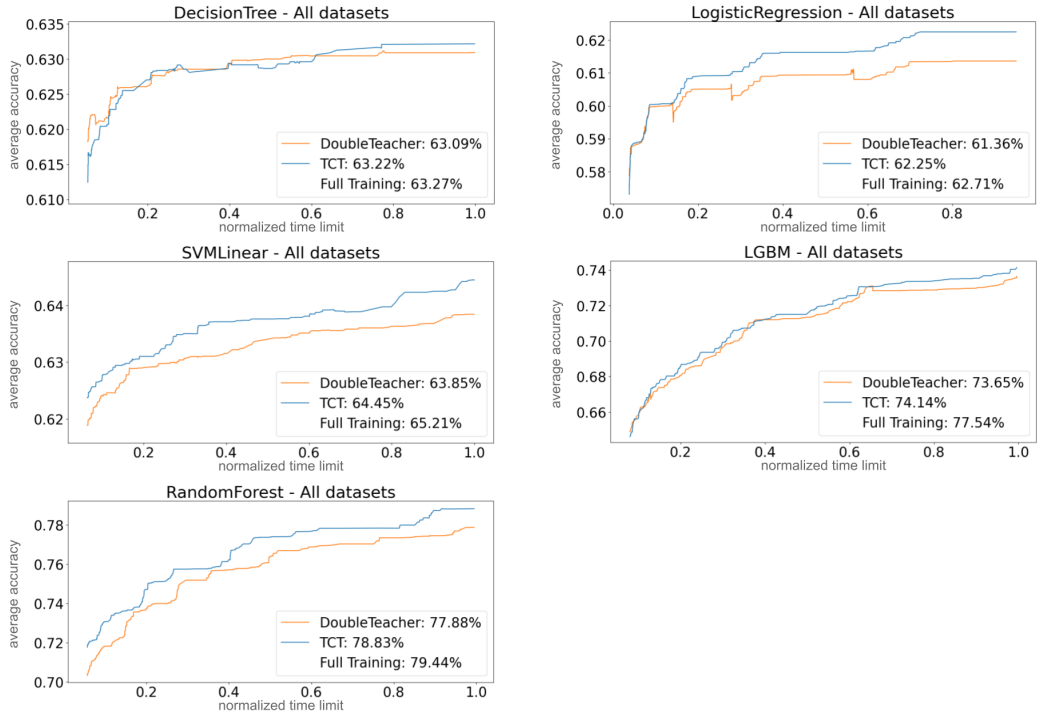

Figura 4.2: Acurácias médias no conjunto de teste ao longo do tempo normalizado para `TCT`, `Double` e `Full`. Os números próximos aos rótulos dos *teachers* são suas acurácias médias no final do tempo limite normalizado ($t = 1$).

exceto `Decision Trees`, em que suas performances são muito semelhantes. Também notamos que, no limite de tempo final $t_{\mathcal{D},\mathcal{L}}$, para a maioria dos *learners*, tanto `TCT` quanto `Double` são capazes de atingir acurácia comparável à do treinamento usando todo o *dataset* (rótulo `Full Training`). A acurácia associada ao treinamento em todo o conjunto de dados pode ser pensada como a acurácia que pode ser alcançada por um *teacher* "oráculo" para `TCL` que sabe antecipadamente o tamanho do maior *batch* de exemplos (aleatórios) que podem ser treinados dentro de um determinado limite de tempo ($t_{\mathcal{D},\mathcal{L}}$ nesse caso) e envia tal conjunto para o *learner*.

A Tabela 4.2 mostra outras estatísticas relevantes no limite de tempo final $t_{\mathcal{D},\mathcal{L}}$. A coluna dupla `TCT` vs `Double` (resp. `TCT` vs `Full`) fornece, para cada *learner*, o número de *datasets* em que o classificador construído por `TCT` supera o de `Double` (resp. `Full`) com confiança de 95% (Seção 5.5 de (MITCHELL, 1997)). Por exemplo, para `LGBM`, `TCT` superou `Double` 9 vezes e foi superado apenas uma vez. Observamos uma clara vantagem de `TCT` sobre `Double` para `LGBM`, `Random Forest` e `SVM`. Para `Logistic Regression` e `Árvores de Decisão` estes algoritmos têm desempenho similar. Talvez surpreendentemente, `TCT` é ainda competitivo contra o treinamento com o conjunto de dados completo para `Random Forest` e `SVM` e, portanto, também competitivo contra o

Tabela 4.2: Estatísticas Adicionais Relevantes.

		TCT vs Double		TCT vs Full		% do Dataset	
Learner	# Datasets	Win	Loss	Win	Loss	Double	TCT
LGBM	15	9	1	6	7	24,1	17,1
Random Forest	20	13	1	4	4	45,0	45,5
SVM	13	7	2	5	5	39,0	34,6
Log. Regression	13	2	2	1	8	29,2	23,3
Decision Tree	7	2	3	2	4	33,4	25,0
Overall	68	33	9	18	28		

teacher "oráculo" (mencionado anteriormente). A coluna dupla "% do *Dataset*" dá a média do número de exemplos, relativo ao tamanho do conjunto de treinamento completo, utilizados pelos modelos construídos por cada par (*teacher*, *learner*). Observamos que TCT constrói modelos mais precisos que Double, apesar de usar 15% menos exemplos (média simples sobre os diferentes *learners*).

Destacamos que a Tabela H.1 do Apêndice H.4 mostra todas as acurácias médias que foram utilizadas para obter as estatísticas da Tabela 4.2.

4.5.2 Comparações com SGD

Comparamos TCT e SGD para treinar SVM e Logistic Regression. Para isso, utilizamos o módulo SGDClassifier da biblioteca sklearn com suas configurações padrões. Em cada iteração, o *learner* recebe um conjunto (mini-batch) aleatório de exemplos rotulados e utiliza SGD, via método parcial_fit, para atualizar o modelo de classificação. Isso é repetido enquanto não atingirmos o limite de tempo (possivelmente com várias passagens no conjunto de treinamento).

Para escolher o tamanho do mini-batch, consideramos todas as possibilidades no conjunto $\{64, 128, 256, 512\}$ e reportamos os tamanhos que obtiveram os melhores resultados. Mais precisamente, 256 para *hinge loss* (SVM) e 512 para *log loss* (Logistic Regression).

TCT teve um desempenho muito melhor: para Logistic Regression, com 95% confiança estatística, superou SGD em 11 de 13 *datasets* e foi pior em apenas 1; para SVM, também de 13 *datasets*, foi melhor em 10 e pior em 2. A Figura 4.3 mostra a acurácia média ao longo do tempo no conjunto de teste.

Notamos que a vantagem de TCT pode ter a ver com o fato de que as classes Scikit-Learn que ele usa para treinar Logistic Regression e SVM empregam técnicas de otimização e possuem hiperparâmetros diferentes dos

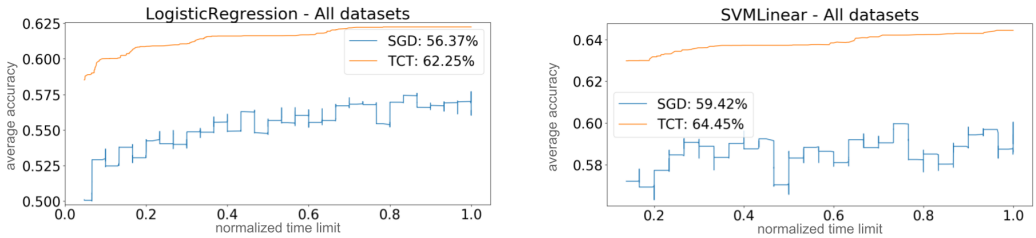

Figura 4.3: Acurácias médias no conjunto de teste ao longo do tempo normalizado para `TCT`, `SGD`. Os números próximos aos rótulos são suas acurácias médias no final do tempo limite normalizado ($t = 1$).

de `SGD`. Dito isso, a principal informação revelada por nossos experimentos é que `TCT` obtém resultados muito melhores do que usar uma alternativa muito natural para um praticante, isto é, treinar por meio da implementação de `SGD` (da biblioteca `SkLearn`) com seus parâmetros padrões. Isso confirma o apelo prático da implementação de um módulo para Aprendizado com Restrição de Tempo que se baseie em nosso método proposto.

4.5.3 Sensibilidade do Hiperparâmetro α

Por fim, realizamos alguns experimentos para entender o impacto do parâmetro α que controla o número de exemplos errados alocados a cada rodada. A Tabela 4.3 mostra o número de vitórias e derrotas de `TCT` sobre `Double` para diferentes valores de α. Em geral, quanto maior o valor de α, maior o número de pares $(\mathcal{L}, \mathcal{D})$, dado pela coluna `Total`, para os quais existe uma diferença com 95% de confiança entre a acurácia de `TCT` e a de `Double`. Isso não é surpreendente, pois quanto menor o valor de α, maior a interseção entre os conjuntos de treinamento empregados por `TCT` e `Double`. O resultado mais interessante é a deterioração da acurácia média com o crescimento de α, o que está de acordo com o Princípio 3. Esses experimentos sugerem que `TCT` é robusto com relação à escolha de α: pode-se defini-lo com segurança no intervalo $[0.05, 0.3]$ e esperar ganhos consistentes.

4.5.4 Selecionando Exemplos via Active Learning

Também avaliamos a possibilidade de substituir os exemplos errados em nossa estratégia por exemplos selecionados via *uncertainty sampling*, uma estratégia clássica de *Active Learning*.

Mais precisamente, consideramos a seguinte variação de `TCT`, denominada $\texttt{TCT}_{AL}$. No início de cada rodada, $\texttt{TCT}_{AL}$ usa o conjunto de treinamento atual

Tabela 4.3: Sensibilidade de α: número de vitórias (coluna Win) e derrotas (coluna Loss) do método `TCT` sobre `Double`, e acurácia média de `TCT`.

α	Win	Loss	Total	Acurácia Média
$0,05$	24	3	27	69,9
$0,1$	31	4	35	70,2
$0,2$	33	9	42	70,3
$0,3$	37	12	49	70,4
$0,45$	32	21	53	70,0
$0,6$	25	35	60	69,5
$0,9$	19	38	57	68,6

para construir um novo modelo. Em seguida, esse modelo é empregado para classificar os exemplos de um conjunto S, contendo $2i$ exemplos aleatórios, em que i é o número de exemplos empregados pelo último modelo treinado. Então, $\texttt{TCT}_{AL}$ seleciona um conjunto $S' \subset S$ contendo i exemplos e o adiciona ao conjunto de treinamento atual, o que aciona o início de uma nova rodada. O conjunto S' é construído escolhendo os $\alpha \cdot i$ exemplos mais incertos de S e $(1-\alpha)\cdot i$ exemplos aleatórios adicionais de S, em que a incerteza de um exemplo é dado pela diferença entre as probabilidades (atribuídas pelo *learner*) das duas classes mais prováveis. Em nossos experimentos usamos $\alpha = 0.2$ tanto para `TCT` quanto para $\texttt{TCT}_{AL}$.

Os resultados são apresentados na Tabela 4.4. A versão "padrão" de `TCT` é claramente melhor para `Random Forest`, tem alguma vantagem para `LGBM` e é pior para `Log. Regression` e `Decision Tree`. Não temos resultados para `SVM` porque as probabilidades necessárias para a estratégia não estão diretamente disponíveis. A figura 4.4 mostra a acurácia normalizada de `TCT` e $\texttt{TCT}_{AL}$ ao longo do tempo. Veja mais detalhes em H.5.

É interessante notar que, apesar de $\texttt{TCT}_{AL}$ usar informações adicionais de *feedback* do *learner* (ou seja, sua incerteza), `TCT` original ainda é competitivo com $\texttt{TCT}_{AL}$. Como `TCT` é capaz de lidar diretamente com qualquer *learner* (ao contrário de $\texttt{TCT}_{AL}$), isso é o que acreditamos ser de maior impacto prático.

4.6 Análise Teórica

Para complementar nosso trabalho, apresentamos resultados teóricos que fornecem uma visão sobre o funcionamento e a qualidade de `TCT`. Como veremos, usar `TCT` com um grande α (fração de "exemplos errados") pode realmente ter um desempenho ruim, como esperado; no entanto, mostraremos que, desde que α não seja muito grande, `TCT` nunca é muito pior do que

Tabela 4.4: Comparação entre `TCT` e sua variação que seleciona exemplos via *Active Learning*.

	# Datasets	Win	Loss
`LGBM`	15	7	4
`Random Forest`	20	13	1
`Log. Regression`	15	0	4
`Decision Tree`	8	1	5

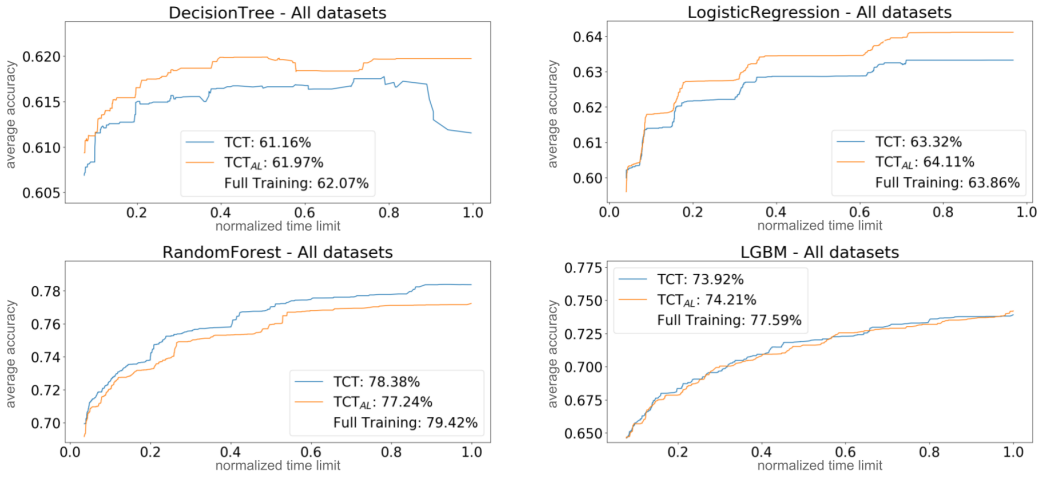

Figura 4.4: Médias das acurácias nos conjuntos de teste ao longo do tempo normalizado para TCT e TCT$_{AL}$. Os números próximos aos rótulos dos algoritmos são suas acurácias médias no final do tempo normalizado $t = 1$.

a amostragem aleatória e, talvez surpreendentemente, em algumas situações é quase exponencialmente mais rápido obter o mesmo erro esperado. Para realizar a análise, consideramos o seguinte cenário:

(i) O algoritmo de treinamento tem acesso a quantos exemplos rotulados, da distribuição desconhecida μ, ele quiser;

(ii) O tempo de treinamento do *learner* $\mathcal{L}$ é no mínimo linear e no máximo polinomial no número de exemplos. Mais precisamente, treinar $\mathcal{L}$ com m exemplos leva tempo $m^k f(m)$, em que $k \geq 1$ e f é uma função sublinear não decrescente;

(iii) O *learner* é um *Empirical Risk Minimizer* (ERM), ou seja, retorna uma hipótese h em sua classe de hipóteses $\mathcal{H}$ que comete o menor número de erros no conjunto de exemplos S que ele recebe;

(iv) O tempo para amostrar um exemplo e classificá-lo com o modelo de classificação atual do *learner* $\mathcal{L}$ é insignificante.

A suposição (i) pode ser aproximada por ter um enorme conjunto de dados D amostrado de μ e é exatamente o que motiva nossa pesquisa, uma vez que para *datasets* menores o Aprendizado com Restrição de Tempo não é particularmente relevante. A segunda suposição também é mínima, dado que a maioria dos métodos de aprendizado usados têm complexidade de tempo polinomial, mas não são sublineares. A suposição (iii) é uma suposição padrão empregada para realizar análises teóricas.

Com relação ao último item, ele é motivado pela situação bastante comum em que o tempo de classificação é muito pequeno em relação ao tempo de treinamento (por exemplo, árvores de decisão e SVM's). Na verdade, poderíamos ter substituído a hipótese (iv) por uma mais fraca, mas isso comprometeria a clareza da apresentação sem alterar nossos principais *insights*.

Para entender a acurácia dos modelos obtidos por `TCT`, analisamos uma versão simplificada do algoritmo denotado por `TCTbase`: como em `TCT`, em cada rodada `TCTbase` envia ao *learner* uma $(1-\alpha)$ fração de exemplos da distribuição original e uma fração α de exemplos em que o modelo atual do *learner* falha. O pseudocódigo do `TCTbase` é apresentado a seguir.

Algoritmo 2: `TCTbase` (α: valor real; T: limite de tempo)

Inicie com um conjunto aleatório S_0 com um único exemplo rotulado amostrado a partir de μ.
Para cada rodada i (iniciando com $i = 1$):

1. Execute o *learner* nos exemplos S_i e receba uma hipótese h_i
2. Obtenha $(1-\alpha)\,2^i$ exemplos rotulados não enviesados a partir de μ.

   Então, repetidamente amostre de μ até obter $\alpha\,2^i$ exemplos rotulados (x, y) em que h_i esteja errada, isto é, $h_i(x) \neq y$.
3. Defina S_{i+1} como S_i mais esses novos 2^i exemplos

Retorne a última hipótese h_i encontrada dentro do limite de tempo T

Notamos que, devido à hipótese (iv), o passo número 2 do algoritmo incorre em tempo de execução insignificante. No entanto, se h_i tiver um pequeno erro, essa suposição se torna inverossímil, pois precisaríamos amostrar um grande número de exemplos de μ para obter $\alpha 2^i$ exemplos errados. Esse problema pode ser corrigido parando o algoritmo assim que uma hipótese com erro no máximo ε (por exemplo, $< 1\%$) for obtida. Isso ocasiona apenas um adicional de $+\varepsilon$ nos limites dos Teoremas 4.1 e 4.2 apresentados mais adiante nesta seção.

Em nossos resultados, comparamos `TCTbase` com o *teacher* "oráculo" `TBatch` que recebe um limite de tempo T e simplesmente envia para o *learner*, em uma única rodada, o maior número de exemplos aleatórios a partir μ para os quais o *learner* consegue treinar dentro do tempo T. Novamente, esse competidor tem a vantagem injusta de saber de antemão o tamanho do *batch* de exemplos que deve enviar a um determinado *learner*; esse é o *teacher* que foi associado ao treinamento com todo o conjunto de dados (`Full`) na seção experimental (veja a Seção 4.5.1 para detalhes).

Antes de apresentar nossos resultados, relembramos brevemente algumas definições do aprendizado estatístico. Para uma distribuição desconhecida μ sobre exemplos rotulados $\mathcal{X} \times \mathcal{Y}$, o erro real de um classificador h é

$$\mathrm{err}(h) := \Pr_{(X,Y)\sim\mu}(h(X) \neq Y).$$

Dado um conjunto de amostras $S = ((X_1, Y_1), \ldots, (X_m, Y_m))$ de μ, o erro amostral (também conhecido como erro empírico) de h é

$$\mathrm{err}_S(h) := \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}(h(X_i) \neq Y_i).$$

Seja $\mathcal{H}$ o Conjunto de hipóteses do *learner*. No cenário *realizável*, todos os rótulos das amostras são dados por um $h^* \in \mathcal{H}$, ou seja, $y = h^*(x)$ para cada (x, y) amostrado a partir de μ. Se o cenário considerado não for realizável, ele é chamado de *agnóstico*.

Análise de desempenho. Conforme discutido no Princípio 3, usado para projetar `TCT` (Seção 4.4), existe a preocupação de que, ao incluir "exemplos errados", enviesemos a distribuição dos exemplos enviados ao *learner* e comprometamos a acurácia real da hipótese aprendida. De fato, construímos um pequeno exemplo no qual `TCTbase` com um grande valor de α (fração de "exemplos errados") é significativamente pior do que `TBatch`; ver Apêndice I.1 para detalhes.

Em contrapartida, provamos que mesmo no pior caso, `TCTbase` retorna uma hipótese com a mesma acurácia que `TBatch` desde que a porcentagem α de exemplos errados não seja tão grande e seja dado um limite de tempo um pouco maior (novamente, isso inclui crucialmente o tempo total consumido pelo *learner* durante as execuções desses algoritmos).

Para deixar mais claro, começamos com o caso realizável. Observe que neste caso qualquer *learner* ERM treinado por `TBatch` retorna alguma hipótese h com erro amostral zero. Seja m_T o número de exemplos que `TBatch` envia ao *learner* dentro do limite de tempo T e seja $\varepsilon_T = \varepsilon_T(\mathcal{H}, \mu, \delta)$ o menor valor

tal que

$$\Pr\left(\ \exists h \in \mathcal{H} \text{ tal que } \mathrm{err}_S(h) = 0 \text{ e } \mathrm{err}(h) > \varepsilon_T\right) < \delta,$$

em que a probabilidade é tomada sobre um conjunto S de m_T exemplos iid amostrados de acordo com μ. Em palavras, ε_T é a melhor garantia de erro comprovável para `TBatch`, com probabilidade $1-\delta$, quando o limite de tempo é T.

Teorema 4.1 *Dado $\delta \in (0,1)$ e um limite de tempo T, seja ϵ_T definido como acima. Sob as hipóteses (i)-(iv), no cenário realizável, com probabilidade de pelo menos $1-\delta$,* `TCTbase` *retorna no tempo no máximo $T \cdot 2(\frac{2}{1-\alpha})^{k+1}$ um classificador com erro no máximo ε_T.*

Prova. Novamente, seja m_T o número de amostras enviadas por `TBatch` quando o limite de tempo for T. Além disso, seja $\hat{i}$ a primeira rodada em que `TCTbase` envia pelo menos $\frac{1}{1-\alpha}m_T$ amostras, ou seja, $\frac{1}{1-\alpha}m_T \leq 2^{\hat{i}} \leq \frac{2}{1-\alpha}m_T$. Devido às hipóteses (ii) e (iv), o tempo que `TCTbase` leva para terminar a rodada $\hat{i}$ é no máximo

$$\begin{aligned}\sum_{i=0}^{\hat{i}} \left((2^i)^k \cdot f(2^i)\right) &\leq f(2^{\hat{i}})\frac{(2^{\hat{i}+1})^k}{2^k-1} \leq 2f(2^{\hat{i}})2^{k\hat{i}} \qquad (4\text{-}2)\\ &\leq 2f\left(\frac{2}{1-\alpha}m_T\right)\left(\frac{2}{1-\alpha}m_T\right)^k \leq 2\left(\frac{2}{1-\alpha}\right)^{k+1} T,\end{aligned}$$

em que a última desigualdade é assegurada por causa da sublinearidade de f e porque $(m_T)^k \cdot f(m_T) \leq T$, por definição de m_T.

Seja S o conjunto de amostras enviadas por `TCTbase` ao *learner* na rodada $\hat{i}$, e seja h a hipótese retornada. A escolha de $\hat{i}$ garante a existência de um subconjunto U de S contendo $(1-\alpha)|S| \geq m_T$ amostras que foram amostradas sem viés a partir da distribuição μ. Como estamos no caso realizável, temos que $err_U(h) = 0$ e, por definição de ϵ_T, a probabilidade de h ter erro real no máximo ϵ_T é de no mínimo $1-\delta$. ■

Observe que quando α é pequeno, o adicional de tempo é de aproximadamente 2^{k+2}, que normalmente não é grande, dado que a maioria dos *learners* usados são razoavelmente eficientes (ou seja, k é pequeno).

Um resultado semelhante também vale para o caso agnóstico. Como ele usa as mesmas ideias, nós o apresentamos em I.2.

Uma aceleração quase exponencial. É importante ressaltar que não apenas `TCTbase` sempre leva um tempo semelhante ao de `TBatch`, mas também mostramos que em alguns casos `TCTbase` é quase exponencialmente mais rápido.

Consideramos o problema clássico de aprender uma função de *threshold* na linha real $\mathbb{R}$ (portanto, os classificadores são do tipo $h(x) = 1$ se $x \geq v$ e $h(x) = -1$ if $x < v$, para $v \in \mathbb{R}$), no caso realizável. Esse é o exemplo canônico em que *Active Learning* fornece uma melhoria exponencial na complexidade de amostragem em comparação com o aprendizado PAC padrão (DASGUPTA, 2009).

O próximo teorema mostra que `TCTbase` atinge uma aceleração quase exponencial para esse problema em comparação com `TBatch`. Como o `TCTbase` não foi adaptado para esse problema, esse resultado atesta sua capacidade de "descartar" hipóteses indesejadas automaticamente. Lembre-se de que $2^{O(\sqrt{\log x})}$ é assintoticamente menor que x^c para qualquer constante $c > 0$.

Teorema 4.2 (Melhoria em relação a `TBatch`) *Considere $\varepsilon, \delta \in (0, 1)$. Seja $T_{\texttt{TBatch}}$ o menor limite de tempo que garante que* `TBatch` *retorne uma hipótese com erro no máximo ε com probabilidade de no mínimo $1-\delta$ para todas as instâncias realizáveis do problema de aprendizado de uma função threshold na linha real, e defina $T_{\texttt{TCTbase}}$ analogamente. Então,*

$$T_{\texttt{TCTbase}} \leq 2^{c \cdot \sqrt{\log T_{\texttt{TBatch}}}},$$

em que c é uma constante que depende de k, α, δ.

Prova. Basta provar que

$$T_{\texttt{TBatch}} \geq \left(\Omega(\tfrac{1}{\varepsilon} \log \tfrac{1}{\delta})\right)^k$$

e

$$T_{\texttt{TCTbase}} \leq \left(2^{O(\sqrt{\log 1/\varepsilon})} \cdot \log \tfrac{1}{\delta} \cdot \left(\tfrac{1}{\alpha}\right)^{O(1)}\right)^{k+1}.$$

Para o limite inferior em $T_{\texttt{TBatch}}$, o limite inferior padrão de complexidade amostral para o aprendizado estatístico de funções de *thresholds* (por exemplo, Teorema 5.3 de (ANTHONY; BARTLETT, 1999)) diz que existe uma instância realizável em que $\Omega(\frac{1}{\varepsilon} \log \frac{1}{\delta})$ amostras são requeridas pelo *learner* para obter erro no máximo ε com probabilidade de pelo menos $1 - \delta$. Assim, dada a hipótese (ii), o limite de tempo precisa ser pelo menos

$$T_{\texttt{TBatch}} \geq \left(\Omega(\tfrac{1}{\varepsilon} \log \tfrac{1}{\delta})\right)^k \cdot f\left(\Omega(\tfrac{1}{\varepsilon} \log \tfrac{1}{\delta})\right) \geq \left(\Omega(\tfrac{1}{\varepsilon} \log \tfrac{1}{\delta})\right)^k$$

para permitir que o *learner* treine com todas essas amostras.

Agora, vamos estabelecer um limite superior para $T_{\texttt{TCTbase}}$. Primeiro verificamos quantos exemplos e rodadas são necessários para atingir o erro ε com probabilidade de pelo menos $1 - \delta$. A ideia, em alto nível, é a seguinte:

na rodada i, com boa probabilidade, os exemplos selecionados por `TCTbase` reduzem o μ-peso do "intervalo de incerteza" para a localização do *threshold* correto v^* por um fator de $\frac{1}{\alpha 2^{i/4}}$. Aplicando isso em todas as rodadas temos que, em cerca de $\log 1/\alpha + \sqrt{\log 1/\varepsilon}$ rodadas, o intervalo de incerteza tem um peso de no máximo ε e, portanto, o *learner* produzirá uma hipótese com erro no máximo ε. Isso rende um total de aproximadamente $2^{\log \frac{1}{\alpha}+\sqrt{\log 1/\varepsilon}} = \frac{1}{\alpha}2^{\sqrt{\log 1/\varepsilon}}$ amostras usadas e tempo de execução $\approx (\frac{1}{\alpha}2^{\sqrt{\log 1/\varepsilon}})^k$. Mas como uma redução tão grande no intervalo de incerteza não é garantida para todas as rodadas e cenários, e os exemplos selecionados em uma rodada dependem do histórico até aquele ponto, precisaremos empregar argumentos de concentração de *martingale* para garantir que um número suficiente dessas rodadas são realmente "boas".

Para tornar esse argumento preciso, lembre-se que S_i é o conjunto de exemplos que `TCTbase` envia ao *learner* na rodada i, e h_i é a hipótese retornada. Novamente usamos a notação de que $U_i \subseteq S_i$ é o conjunto de amostras até o início da rodada i que foram amostrada sem viés da distribuição μ, e $W_i \subseteq S_i$ é o conjunto "exemplos errados" amostrados até agora. Usamos μ_x para denotar a distribuição marginal de μ em $\mathcal{X} = \mathbb{R}$.

Seja $v^* \in \mathbb{R}$ o *threshold* correto para a instância especificada. Seja $I_i \subseteq \mathbb{R}$ o intervalo máximo contendo v^* que não contém nenhum dos exemplos S_i (a "região de incerteza" neste momento). Defina $E_i \subseteq \mathbb{R}$ como os pontos nos quais a classificação de h_i está incorreta. Observe que essa região é um intervalo que está "à esquerda" ou "à direita" de v^* (dependendo se o *threshold* usado em h_i está à esquerda ou à direita de v^*). Observe também que E_i está contido em I_i: como estamos em uma instância realizável, a amostra mais à direita em S_i à esquerda de v^* (que é o ponto inicial do intervalo aberto I_i) força o *learner* ERM a classificar todos os pontos antes dele corretamente como -1, e de forma semelhante aos pontos após o final do intervalo aberto I_i.

Definimos o "peso esquerdo" $\mathrm{Lw}(I)$ do intervalo I como sendo a quantidade de μ_x-massa na parte do intervalo que está à esquerda de v^*, ou seja, $\mathrm{Lw}(I) = \mu_x(I \cap (-\infty, v^*])$. Da mesma forma, defina o "peso direito" $\mathrm{Rw}(I) = \mu_x(I \cap [v^*, \infty))$. A seguinte proposição é a base da idéia de "busca binária automática" e diz que com boa probabilidade em cada rodada reduzimos significativamente o peso esquerdo/direito da região de incerteza.

Proposição 4.6.1 *Para cada realização das amostras sorteadas antes da rodada i em que a região de erro E_i está à esquerda de v^*, com probabilidade de pelo menos $1 - e^{-2^{3i/4}}$ (com relação às amostras amostradas na rodada i) temos*

$$\mathrm{Lw}(I_{i+1}) \leq \frac{\mathrm{Lw}(I_i)}{\alpha 2^{i/4}}.$$

Da mesma forma, se a região de erro E_i estiver à direita de v^*, com probabilidade de pelo menos $1 - e^{-2^{3i/4}}$ temos

$$\mathrm{Rw}(I_{i+1}) \leq \frac{\mathrm{Rw}(I_i)}{\alpha 2^{i/4}}.$$

Prova. Provamos apenas a primeira afirmação, pois a segunda é análoga. Fixe uma realização das amostras até o início da rodada i em que E_i está à esquerda de v^*. Por construção, na rodada i o algoritmo seleciona $\alpha 2^i$ amostras da distribuição μ_x condicionadas a estarem no conjunto E_i, chameas de $X_1, \ldots, X_{\alpha 2^i}$. Seja $\bar{v}$ o ponto tal que o intervalo $[\bar{v}, v^*]$ tenha μ_x-medida $\frac{1}{\alpha 2^{i/4}} \cdot \mu_x(E_i)$. A probabilidade de que nenhuma das amostras X_i caia nesse intervalo é $\left(1 - \frac{1}{\alpha 2^{i/4}}\right)^{\alpha 2^i} \leq e^{-2^{3i/4}}$. Então, uma dessas amostras cai em $[\bar{v}, v^*]$ com probabilidade de pelo menos $1 - e^{-2^{3i/4}}$. Quando isso acontece, o próximo conjunto de incerteza I_{i+1} começa no/depois do ponto $\bar{v}$ e, portanto, seu peso esquerdo satisfaz

$$\mathrm{Lw}(I_{i+1}) \leq \mu_x([\bar{v}, v^*]) = \frac{1}{\alpha 2^{i/4}}\, \mu_x(E_i) \leq \frac{1}{\alpha 2^{i/4}}\, \mathrm{Lw}(I_i),$$

em que a última desigualdade se dá porque $E_i \subseteq I_i$ e, portanto (já que E_i está à esquerda de v^*), $\mu_x(E_i) \leq \mu_x(I_i \cap (-\infty, v^*]) = \mathrm{Lw}(I_i)$. Isso dá o resultado desejado. ■

Seja $R := cst \cdot (\log \frac{1}{\alpha} + \sqrt{\log \frac{1}{\varepsilon}} + 1) + \log\log \frac{1}{\delta}$, para uma constante suficientemente grande cst. Usando a afirmação acima, mostramos que com probabilidade de pelo menos $1 - \delta$, o conjunto de erros E_{R+1} na rodada $R+1$ tem μ_x-medida no máximo ε, ou seja, h_{R+1} tem erro no máximo ε. Seja B_i o indicador do evento ruim de que a redução de peso determinada pela proposição anterior não ocorreu na rodada i. Então $\mathbb{E}[B_i \mid B_1, \ldots, B_{i-1}] \leq e^{-2^{3i/4}}$. Além disso, usando a concentração de martingale, temos o seguinte *tail bound* (deferimos a prova para Apêndice I.3).

Proposição 4.6.2

$$\Pr\left(\sum_{i=2R/3}^{R} B_i \geq \frac{R}{6}\right) \leq e^{-2^{R/2}} \leq \delta,$$

Basta então mostrar que sempre que $\sum_{i=2R/3}^{R} B_i < \frac{R}{6}$, temos $\mu_x(E_R) \leq \varepsilon$. Portanto, fixe um cenário que satisfaça $\sum_{i=2R/3}^{R} B_i < \frac{R}{6}$ e suponha, por contradição, que $\mu_x(E_R) > \varepsilon$. Defina os subconjuntos G, L, R das rodadas $\{1, \ldots, R\}$ da seguinte forma: G é o conjunto de índices "bons" i tal que $B_i = 0$, L o conjunto de índices no qual E_i está à esquerda de v^*, e R o conjunto de índices em que E_i está à direita de v^*.

Então, $L \cap G$ são as rodadas em que a redução determinada pela Proposição 4.6.1 aconteceu no peso esquerdo de I_i, e $R \cap G$ em que aconteceu em seu peso direito. Além disso, os conjuntos $I_1, I_2, \ldots$ são monotonicamente decrescentes, portanto, mesmo para as rodadas fora do conjunto "bom" G, os pesos esquerdo e direito de I_i apenas diminuem com o tempo. Então,

$$\mathrm{Lw}(I_{R+1}) \;\leq\; \mathrm{Lw}(\mathbb{R}) \cdot \prod_{i \in L \cap G} \frac{1}{\alpha 2^{i/4}} \;\leq\; \left(\frac{1}{\alpha}\right)^{R} \cdot \frac{1}{2^{\sum_{i \in L \cap G} i/4}}.$$

Combinando com o fato que $\mathrm{Lw}(I_{R+1}) \geq \mathrm{Lw}(E_{R+1}) \geq \mu_x(E_{R+1}) > \varepsilon$, e tirando logs, isso implica

$$\sum_{i \in L \cap G} i \;\leq\; 4R \log \frac{1}{\alpha} + 4 \log \frac{1}{\varepsilon}.$$

A mesma desigualdade vale com L substituído por R, e somando essas desigualdades temos

$$\sum_{i \in (L \cup R) \cap G} i \;\leq\; 8R \log \frac{1}{\alpha} + 8 \log \frac{1}{\varepsilon}. \tag{4-3}$$

Pela hipótese do cenário, sabemos que pelo menos $\frac{R}{3} - \frac{R}{6} = \frac{R}{6}$ das rodadas no intervalo $\{\frac{2R}{3}, \ldots, R\}$ estão no conjunto bom G, ou seja, $|G| \geq \frac{R}{6}$. Então

$$\sum_{i \in (L \cup R) \cap G} i \;=\; \sum_{i \in G} i \;\geq\; \sum_{i=1}^{|G|} i \geq \sum_{i=1}^{R/6} i \;\geq\; \frac{R^2}{72}.$$

Porém, como $R \geq cst \cdot (\log \frac{1}{\alpha} + \sqrt{\log \frac{1}{\varepsilon}})$ para uma constante suficientemente grande cst, isso contradiz a desigualdade (4-3) ($cst = 8 \cdot 72$ é suficiente, mas as constantes não foram otimizadas).

Então, com probabilidade no mínimo $1 - \delta$, pela rodada R o algoritmo obtém uma classificação com erro no máximo ε. Devido às hipóteses (ii) e (iv), o mesmo desenvolvimento como na desigualdade (4-2) mostra que essa rodada termina no tempo

$$\sum_{i=1}^{R} \left((2^i)^k \cdot f(2^i) \right) \leq 2 f(2^R) 2^{Rk} \leq 2^{R(k+1)+1}.$$

Usando o valor de R, esse tempo é no máximo

$$\left(2^{O(\sqrt{\log 1/\varepsilon})} \cdot \log \tfrac{1}{\delta} \cdot \left(\tfrac{1}{\alpha}\right)^{O(1)} \right)^{k+1},$$

que então conclui a prova. ■

4.7 Conclusões

Apresentamos a tarefa de Aprendizado com Restrição de Tempo e o algoritmo `TCT` para lidar com ela. Nosso algoritmo se baseia em ideias metodologicamente sólidas, é apoiado por resultados teóricos e, mais importante, experimentos envolvendo 20 conjuntos de dados, 5 algoritmos de aprendizado diferentes e outros três algoritmos *baselines* sugerem que ele é uma boa escolha para a tarefa de Aprendizado com Restrição de Tempo. Devido à sua simplicidade e generalidade, o `TCT` poderia ser facilmente implementado como um *wrapper* em bibliotecas de aprendizado de máquina para lidar com o aprendizado com restrição de tempo.

Como trabalho futuro, seria interessante investigar maneiras de misturar exemplos selecionados pelo `TCT` com estratégias de *Active Learning* (AL). Embora isso possa reduzir a generalidade do algoritmo (já que a maioria das estratégias de AL depende de propriedades adicionais dos algoritmos de aprendizado), parece ser uma direção promissora, conforme indicado pelos experimentos apresentados na Seção 4.5.4.

5
Considerações Finais

Neste trabalho, abordamos o problema de aprendizado em um cenário com restrição de recursos e propomos soluções via paradigma de *Machine Teaching*. Primeiramente, obtemos resultados teóricos abordando o problema de *Machine Teaching*[1] como ele é tradicionalmente tratado na literatura, em que o objetivo é minimizar o tamanho do *teaching set* (Capítulo 3). No Capítulo 4, tratamos esse paradigma de forma mais aplicada, na qual introduzimos o problema de Aprendizado com Restrição de Tempo (TCL), que pode ser visto como uma formulação alternativa para o problema de *Machine Teaching* (Seção 2.3.1) em que o objetivo é encontrar o *teaching set* para o qual o *learner* retorne o melhor classificador possível e não extrapole o limite de tempo.

Para o problema de *Machine Teaching*, consideramos o cenário no qual o *teacher* tem conhecimento limitado sobre o *learner* e, para o caso realizável, propomos o algoritmo $\texttt{OSCT}_{base}$ que converge com alta probabilidade para a hipótese alvo h^* enviando $O(\mathcal{TS}\ \log m\ \log n)$ exemplos, sendo $\mathcal{TS}$ a quantidade mínima necessária, m o tamanho do *dataset* e $n = |\mathcal{H}|$. Em particular, para o caso realizável, nosso resultado é similar ao apresentado em (DASGUPTA et al., 2019), porém, a análise de (DASGUPTA et al., 2019) é baseada no conhecimento de um limite superior para n e, diferente da nossa análise, Dasgupta et al. (2019) não tratam algumas sutilezas relevantes decorrentes da interdependência entre teacher e learner. Também usamos nosso algoritmo $\texttt{OSCT}_{base}$ como base para aprimorar nossos resultados, construir novos algoritmos e tratar algumas variações do problema. Em particular, consideramos o cenário com *black-box learners* não adversários e mostramos que podemos obter melhores resultados para o tipo de *learner* que se move para a próxima hipótese de maneira suave, preferindo hipóteses que são mais próximas da hipótese atual. Para esse tipo de *learner*, quando a proximidade é medida em termos de distância de Hamming, apresentamos um algoritmo de ensino que com alta probabilidade envia $O(\mathcal{TS} \log n(\log err_1 + \log\log n))$ exemplos, em que err_1 é o número de erros da primeira hipótese fornecida pelo *learner*.

Além disso, ainda para o problema de *Machine Teaching*, generaliza-

[1]Relembrando: a área de *Machine Teaching* estuda o problema de encontrar um *teaching set* ótimo D. Desse modo, nos referimos a esse objetivo como o problema de *Machine Teaching*. Mais especificamente, utilizamos esse termo como sinônimo para o problema de minimizar o tamanho do *teaching set*.

mos o limite acima para o caso não realizável para o qual propusemos um algoritmo que garante que, com alta probabilidade, o *learner* converge para a hipótese $\tilde{h}$, que é a mais próxima de h^* em sua classe de hipótese, após receber $O(\mathcal{TS}_k \log m \log(m+n))$ exemplos, em que $\mathcal{TS}_k$ é um limite inferior no número de exemplos necessários para essa tarefa. Para finalizar, realizamos experimentos com 12 *datasets* reais para comparar nosso *teacher* OSCT em relação a um *teacher* que seleciona exemplos de forma aleatória. Nesses experimentos, pudemos observar que OSCT utilizou significativamente menos exemplos do que o *teacher* randomizado (que seleciona exemplos aleatoriamente) para ambos atingirem uma mesma acurácia (por exemplo, um fator de 3 para atingir 99% da acurácia obtida pelo *learner* treinando em todo o conjunto de treino). Sendo assim, pudemos constatar o bom desempenho de utilizar *Machine Teaching* em cenários que necessitamos obter um bom classificador treinando em um subconjunto reduzido do *dataset*.

Apesar desses bons resultados mesmo em um cenário mais realista (sem fortes suposições sobre o *learner*), observamos que, assim como nosso método, os métodos convencionais de *Machine Teaching* ignoram um fator que pode ser importante em um cenário de aprendizado em larga escala de dados e com recursos limitados: o tempo que leva para construir o *teaching set*. Por esse motivo e pelo potencial de aplicação, resolvemos dar continuidade em nossa pesquisa introduzindo e abordando um cenário prático de Aprendizado com Restrição de Tempo (TCL).

Para o problema de TCL, propusemos o algoritmo Time Constrained Teacher (TCT), projetado com base em princípios/ideias bem estabelecidos da Teoria da Aprendizagem Computacional e *Machine Teaching*. Destacamos como ponto positivo a simplicidade do nosso algoritmo, que emprega apenas um único parâmetro simples de definir, robusto e que não necessita de nenhuma informação sobre o *learner*. Realizamos experimentos computacionais envolvendo 20 *datasets*, 5 diferentes algoritmos de aprendizado e 3 *baselines*, que sugerem que nosso método é uma boa escolha para resolver o problema. Também implementamos e disponibilizamos um *framework* Python no qual o usuário pode usar nosso método `TCT` integrado com algoritmos de aprendizado disponíveis na biblioteca scikit-learn (ver Apêndice A).

Embora o foco dessa segunda etapa da pesquisa fosse tratar o problema de Aprendizado com Restrição de Tempo de forma prática, também demonstramos que uma versão simplificada do nosso algoritmo tem garantias teóricas. Sob suposições razoáveis, mostramos que o tempo que nosso algoritmo leva para atingir uma certa acurácia nunca é muito maior que o tempo gasto pelo

melhor *teacher* randomizado (oráculo[2]), mas pode ser quase exponencialmente mais rápido.

Como linha de pesquisa futura, seria interessante investigar formas de mesclar exemplos selecionados por nosso método `TCT` com estratégias de *Active Learning* (AL). Conforme indicado pelos experimentos apresentados na Seção 4.5.4, essa parece ser uma direção promissora. Contudo, isso pode reduzir a generalidade do algoritmo pelo fato que a maioria das estratégias de AL depende de propriedades adicionais dos algoritmos de aprendizado.

[2]Relembrando: o *teacher* oráculo sabe a quantidade máxima de exemplos aleatórios que o *learner* pode treinar sem estourar o tempo limite.

6
Referências bibliográficas

ALON, N. et al. The online set cover problem. **SIAM J. Comput**, v. 39, n. 2, p. 361–370, 2009. Citado 3 vezes nas páginas 31, 34 e 35.

ANGLUIN, D.; DOHRN, T. The power of random counterexamples. **Theor. Comput. Sci.**, v. 808, p. 2–13, 2020. Citado na página 32.

ANTHONY, M.; BARTLETT, P. L. **Neural Network Learning: Theoretical Foundations**. [S.l.]: Cambridge University Press, 1999. Citado 3 vezes nas páginas 15, 24 e 69.

ANôNIMO; DASGUPTA, S. Anytown, MN, USA: [s.n.], 2020. Comunicação Privada: discussão da prova do Lema 2. Citado na página 86.

BALBACH, F. J.; ZEUGMANN, T. Teaching randomized learners with feedback. **Inf. Comput**, v. 209, n. 3, p. 296–319, 2011. Citado na página 32.

BOTTOU, L.; BOUSQUET, O. The tradeoffs of large scale learning. **Advances in neural information processing systems**, v. 20, 2007. Citado na página 52.

BUCHBINDER, N.; NAOR, J. Online primal-dual algorithms for covering and packing. **Mathematics of Operations Research**, INFORMS, v. 34, n. 2, p. 270–286, 2009. Citado 5 vezes nas páginas 40, 92, 93, 94 e 95.

CHEN, Y. et al. Understanding the role of adaptivity in machine teaching: The case of version space learners. In: BENGIO, S. et al. (Ed.). **Advances in Neural Information Processing Systems 31**. [S.l.: s.n.], 2018. p. 1476–1486. Citado 3 vezes nas páginas 25, 32 e 52.

CICALESE, F. et al. Teaching with limited information on the learner's behaviour. In: **Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event**. [S.l.]: PMLR, 2020. (Proceedings of Machine Learning Research, v. 119), p. 2016–2026. Citado 9 vezes nas páginas 13, 29, 50, 51, 52, 53, 54, 57 e 58.

DASGUPTA, S. The two faces of active learning. In: GAVALDÀ, R. et al. (Ed.). **Algorithmic Learning Theory**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 1–1. ISBN 978-3-642-04414-4. Citado 2 vezes nas páginas 55 e 69.

DASGUPTA, S. et al. Teaching a black-box learner. In: CHAUDHURI, K.; SALAKHUTDINOV, R. (Ed.). **Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA**. [S.l.]: PMLR, 2019. (Proceedings of Machine Learning Research, v. 97), p. 1547–1555. Citado 16 vezes nas páginas 5, 7, 8, 25, 29, 31, 33, 50, 51, 52, 53, 54, 57, 74, 86 e 87.

DEVIDZE, R. et al. Understanding the power and limitations of teaching with imperfect knowledge. In: BESSIERE, C. (Ed.). **Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020**. [S.l.]: ijcai.org, 2020. p. 2647–2654. Citado na página 50.

DU, C. X. L. J. Active teaching for inductive learners. In: **Proceedings of the Eleventh SIAM International Conference on Data Mining, SDM 2011, April 28-30, 2011, Mesa, Arizona, USA**. [S.l.]: SIAM / Omnipress, 2011. p. 851–861. Citado 2 vezes nas páginas 52 e 54.

FREEDMAN, D. A. On tail probabilities for martingales. **Annals of Probability**, v. 3, p. 100–118, 1975. Citado 2 vezes nas páginas 99 e 119.

FREITAS, S. et al. Time-constrained learning. **Pattern Recognition**, v. 142, p. 109672, 2023. ISSN 0031-3203. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0031320323003734>. Citado 2 vezes nas páginas 13 e 49.

GAO, Z. et al. Preference-based teaching. **J. Mach. Learn. Res**, v. 18, p. 31:1–31:32, 2017. Disponível em: <http://jmlr.org/papers/v18/16-460.html>. Citado na página 32.

GOLDMAN, S. A.; KEARNS, M. J. On the complexity of teaching. **J. Comput. Syst. Sci**, v. 50, n. 1, p. 20–31, 1995. Citado 6 vezes nas páginas 24, 29, 30, 31, 50 e 52.

JOHNS, E.; AODHA, O. M.; BROSTOW, G. J. Becoming the expert-interactive multi-class machine teaching. In: **proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2015. p. 2616–2624. Citado na página 25.

KIRKPATRICK, D.; SIMON, H. U.; ZILLES, S. Optimal collusion-free teaching. In: GARIVIER, A.; KALE, S. (Ed.). **Proceedings of the 30th International Conference on Algorithmic Learning Theory**. Chicago, Illinois: PMLR, 2019. (Proceedings of Machine Learning Research, v. 98), p. 506–528. Disponível em: <http://proceedings.mlr.press/v98/kirkpatrick19a.html>. Citado na página 32.

KORMAN, S. **On the Use of Randomization in the Online Set Cover Problem**. Tese ( ) — Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Israel, 2004. Citado 4 vezes nas páginas 32, 43, 102 e 103.

KORMAN, S. On the use of randomization in the online set cover problem. **Weizmann Institute of Science**, v. 2, 2004. Citado na página 34.

LIU, W. et al. Iterative machine teaching. In: PMLR. **International Conference on Machine Learning**. [S.l.], 2017. p. 2149–2158. Citado 5 vezes nas páginas 10, 25, 26, 27 e 52.

LIU, W. et al. Towards black-box iterative machine teaching. In: DY, J. G.; 0001, A. K. (Ed.). **ICML**. [S.l.]: PMLR, 2018. (Proceedings of Machine

Learning Research, v. 80), p. 3147–3155. Citado 5 vezes nas páginas 25, 29, 50, 52 e 53.

MA, Y. et al. Teacher improves learning by selecting a training subset. In: STORKEY, A. J.; PÉREZ-CRUZ, F. (Ed.). **International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain**. [S.l.]: PMLR, 2018. (Proceedings of Machine Learning Research, v. 84), p. 1366–1375. Citado na página 52.

MANSOURI, F. et al. Preference-based batch and sequential teaching: Towards a unified view of models. In: **Advances in Neural Information Processing Systems 32**. [S.l.]: Curran Associates, Inc., 2019. p. 9195–9205. Citado na página 32.

MEI, S.; ZHU, X. Using machine teaching to identify optimal training-set attacks on machine learners. In: **Twenty-Ninth AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2015. Citado na página 25.

MELO, F. S.; GUERRA, C.; LOPES, M. Interactive optimal teaching with unknown learners. In: LANG, J. (Ed.). **Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden**. [S.l.]: ijcai.org, 2018. p. 2567–2573. Citado na página 50.

MITCHELL, T. **Machine Learning**. [S.l.]: McGraw-Hill Education, 1997. Citado 3 vezes nas páginas 56, 61 e 113.

MITZENMACHER, M.; UPFAL, E. **Probability and Computing: Randomized Algorithms and Probabilistic Analysis**. [S.l.]: Cambridge University Press, 2005. I–XVI, 1–352 p. ISBN 978-0-521-83540-4; 9780511813603. Citado 2 vezes nas páginas 105 e 106.

POURKAMALI-ANARAKI, F.; BENNETTE, W. D. Adaptive data compression for classification problems. **IEEE Access**, IEEE, v. 9, p. 157654–157669, 2021. Citado na página 52.

RAFFERTY, A. N. et al. Faster teaching via pomdp planning. **Cognitive science**, Wiley Online Library, v. 40, n. 6, p. 1290–1332, 2016. Citado na página 25.

RAZ; SAFRA. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In: **STOC: ACM Symposium on Theory of Computing (STOC)**. [S.l.: s.n.], 1997. Citado 3 vezes nas páginas 32, 44 e 104.

SCHAPIRE, R. E. A brief introduction to boosting. In: DEAN, T. (Ed.). **Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages**. [S.l.]: Morgan Kaufmann, 1999. p. 1401–1406. Citado na página 54.

SETTLES, B. **Active Learning Literature Survey**. [S.l.], 2009. Citado 3 vezes nas páginas 26, 27 e 53.

SEUNG, H. S.; OPPER, M.; SOMPOLINSKY, H. Query by committee. In: **Proceedings of the fifth annual workshop on Computational learning theory**. [S.l.: s.n.], 1992. p. 287–294. Citado na página 54.

SHINOHARA, A. Teachability in computational learning. **New Generation Comput**, v. 8, n. 4, p. 337–347, 1991. Citado 5 vezes nas páginas 24, 29, 31, 50 e 52.

SHWARTZ, S. S.; DAVID, S. B. **Understanding machine learning: From theory to algorithms**. [S.l.]: Cambridge university press, 2014. Citado 10 vezes nas páginas 10, 15, 17, 18, 19, 20, 21, 22, 23 e 24.

SINGLA, A. et al. Near-optimally teaching the crowd to classify. In: **ICML**. [S.l.]: JMLR.org, 2014. (JMLR Workshop and Conference Proceedings, v. 32), p. 154–162. Citado 2 vezes nas páginas 32 e 52.

VALIANT, L. G. A theory of the learnable. **Communications of the ACM**, ACM New York, NY, USA, v. 27, n. 11, p. 1134–1142, 1984. Citado na página 27.

VICTOR, H.; GINÉ, E. **Decoupling: From Dependence to Independence**. [S.l.]: Springer New York, 1999. Citado na página 31.

ZHOU, Y.; NELAKURTHI, A. R.; HE, J. Unlearn what you have learned: Adaptive crowd teaching with exponentially decayed memory learners. In: GUO, Y.; FAROOQ, F. (Ed.). **KDD**. [S.l.]: ACM, 2018. p. 2817–2826. Citado na página 25.

ZHU, X. Machine teaching: An inverse problem to machine learning and an approach toward optimal education. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2015. v. 29, n. 1. Citado na página 24.

ZHU, X. et al. An overview of machine teaching. **CoRR**, abs/1801.05927, 2018. Citado 3 vezes nas páginas 25, 28 e 50.

ZILLES, S. et al. Models of cooperative teaching and learning. **J. Mach. Learn. Res**, v. 12, p. 349–384, 2011. Disponível em: <http://portal.acm.org/citation.cfm?id=1953059>. Citado na página 32.

A
TCLtask: Um Framework para Aprendizado com Restrição de Tempo

Neste capítulo apresentamos o *framework* `TCLtask`, uma contribuição adicional para o problema de Aprendizado com Restrição de Tempo apresentado no Capítulo 4. A partir desse *framework*, desenvolvido em Python3, é possível utilizar facilmente nosso *teacher* (`TCT`), proposto na Seção 4, bem como sua versão de *Active Learning* e o *baseline* `Double` que seleciona exemplos aleatoriamente.

Na Seção A.1 são dadas as instruções de instalação da nossa ferramenta. Já a Seção A.2 fornece um passo a passo de como utilizar `TCLtask` a partir de exemplos práticos.

A.1
Instalação

Para instalar a biblioteca Python `TCLtask`, siga os seguintes passos:

- Baixe o arquivo de instalação da biblioteca Python disponível em <https://github.com/sfilhofreitas/TCLlibrary/tree/main/dist>
- Abra um terminal de linha de comando e navegue até o diretório onde o arquivo de instalação foi salvo.
- Use o comando pip para instalar a biblioteca[1]. O comando pode variar de acordo com o nome do arquivo baixado. Se o arquivo tiver o nome "TCLibrary-0.0.1-py3-none-any.whl", por exemplo, use o seguinte comando: pip install TCLibrary-0.0.1-py3-none-any.whl
- Certifique-se de substituir "TCLibrary-0.0.1-py3-none-any.whl" pelo nome do arquivo baixado.

Aguarde enquanto o pip instala a biblioteca Python. Após a instalação, você poderá importar a biblioteca em seus projetos Python normalmente.

O código desta ferramenta está disponível em repositório remoto no seguinte endereço: <https://github.com/sfilhofreitas/TCLlibrary/>. Em caso de dúvida, de sugestões ou de encontrar algum problema durante a instalação/utilização, entre em contato com o desenvolvedor.

[1]Recomendamos a instalação em um ambiente virtual para evitar possíveis problemas de dependências de pacotes.

A.2 Utilizando `TCLtask` Passo a Passo

Nesta seção fazemos um tutorial com exemplos práticos de como utilizar nosso *framework* Python. O primeiro passo é importar o *framework* e a biblioteca sklearn[2], que oferece diversos modelos de classificação que podem ser utilizados como *learners*:

```
1 import TCLtask as tcl
2 import sklearn as skl
```

Posteriormente, é necessário escolher o *teacher* desejado. `TCLtask` conta com três algoritmos: (i) nosso algoritmo `TCT`; (ii) o *teacher* `Double` que, de forma iterativa, dobra o tamanho do conjunto de treino escolhendo os novos exemplos aleatoriamente; e (iii) a versão de `TCT` que utiliza *active learning* (AL) para selecionar exemplos. A priori, escolhemos nosso método `TCT` para construir nosso primeiro exemplo:

```
3 TCT = tcl.Teachers.TCT(seed=0, frac_start=0.005, alpha=0.2)
```

O parâmetro *seed* define uma semente de aleatoriedade do *teacher*. Já o parâmetro *frac_start*, define o tamanho do conjunto de exemplos utilizados para treinar o primeiro modelo. Vale destacar que, para esse primeiro conjunto, os exemplos são escolhidos de maneira aleatória. Por fim, *alpha* (não aplicável apenas para `Double`) define a razão entre exemplos aleatórios e não aleatórios fornecidos ao *learner* a cada iteração. Mais precisamente, a cada iteração, `TCT` dobra o tamanho do conjunto de treino atual S selecionando (aproximadamente) $alpha \cdot |S|$ exemplos em que o modelo atual erra e $(1 - alpha)|S|$ exemplos aleatórios. O mesmo vale para a versão de `TCT` que utiliza AL, porém, em vez de exemplos errados, são priorizados os exemplos de maiores incertezas[3].

Uma vez escolhido o *teacher*, deve-se escolher o modelo de classificação que se deseja utilizar. Como já ressaltamos, `TCLtask` é compatível com os modelos oferecidos/padronizados pela biblioteca sklearn. Podemos utilizar, por exemplo, uma árvore de decisão como a que usamos em nossos experimentos computacionais:

```
4 L = skl.tree.DecisionTreeClassifier(random_state=0,
                                      min_samples_split=30,
```

[2]https://scikit-learn.org/stable/

[3]Consideramos a incerteza de um exemplo como a diferença das probabilidades (atribuídas pelo *learner*) para as duas classes mais prováveis.

max_depth = 5)

Para seguir com nosso tutorial, adotamos o *dataset* mnist que pode ser obtido e processado facilmente via código Python, conforme ilustrado na Figura A.3. Sobre os dados, é necessário que a matriz de atributos seja de dimensões (m, k) – em que m é o número de exemplos e k o número de atributos – e que o vetor de rótulos pertença ao intervalo [0, c-1], em que c é o número de classes. Em particular, para esse conjunto de dados, o tempo necessário para treinar L utilizando todos os exemplos de treino (em uma única rodada) é aproximadamente 5 segundos.

O último passo é chamar a função de ensino responsável pelo processo de interação entre *teacher* e *learner*:

```
5 teach_result = tcl.teach(T = TCT, L = L,
                           X = train_X, X_labels = train_y,
                           time_limit = 2)
```

A função *tcl.teach* recebe como parâmetro um *teacher*, um *learner*, uma matriz X com os dados de treinamento (apenas atributos), um vetor X_labels com os rótulos dos dados de treinamento e um limite de tempo (em segundos).

Agora podemos utilizar o modelo treinado e retornado pela função *tcl.teach* para classificar os exemplos do conjunto de teste do mnist e obter sua acurácia nesse conjunto:

```
6 pred_y = teach_result.model.predict(test_X)
7 acc = (pred_y == test_y).sum()/test_y.size
8 print('Acurácia obtida: ', acc)
Out [8]: Acurácia obtida: 0.6825
```

Nas Figuras A.1 e A.2, mostramos como realizar o mesmo teste utilizando, respectivamente, os *teachers* `Double` e `ALTCT`.

Como podemos notar na Figura A.2, para utilizar `ALTCT` é necessário especificar o parâmetro *send_probabilities = True* na função *tcl.teach*. Esse parâmetro é específico para AL e determina que, para cada exemplo e classificado no processo de ensino, o *learner* também informe a probabilidade de e pertencer a cada classe[4].

Além de disponibilizar o modelo retornado (teach_result.model), o objeto teach_result salva todos os modelos treinados (teach_result.models) e nos permite utilizá-los como um comitê de classificadores:

[4]É necessário que o *learner* disponha do método *predict_proba*.

```
1  import TCLtask as tcl
2  import sklearn as skl
3  Double = tcl.Teachers.DoubleTeacher(seed=0, frac_start=0.005)
4  L = skl.tree.DecisionTreeClassifier(random_state=0,
                                       min_samples_split=30,
                                       max_depth = 5)
5  teach_result = tcl.teach(Double, L,
                            train_X, train_y,
                            time_limit=2)
6  pred_y = teach_result.model.predict(test_X)
7  acc = (pred_y == test_y).sum()/test_y.size
8  print('Acurácia obtida: ', acc)
Out [8]: Acurácia obtida: 0.6651
```

Figura A.1: Exemplo utilizando o *teacher* `Double`.

```
1  import TCLtask as tcl
2  import sklearn as skl
3  ALTCT = tcl.Teachers.ALTCT(seed=0, frac_start=0.005, alpha=0.2)
4  L = skl.tree.DecisionTreeClassifier(random_state=0,
                                       min_samples_split=30,
                                       max_depth = 5)
5  teach_result = tcl.teach(ALTCT, L,
                            train_X, train_y,
                            time_limit=2,
                            send_probabilities=True)
6  pred_y = teach_result.model.predict(test_X)
7  acc = (pred_y == test_y).sum()/test_y.size
8  print('Acurácia obtida: ', acc)
Out [8]: Acurácia obtida: 0.6712
```

Figura A.2: Exemplo utilizando o *teacher* `ALTCT`.

```
9  pred_y = teach_result.predict_by_committee(test_X)
10 acc = (pred_y == test_y).sum()/test_y.size
11 print('Acurácia obtida: ', acc)
Out [11]: Acurácia obtida: 0.7649
```

Novamente estamos considerando o caso em que utilizamos nosso método `TCT`. Como podemos perceber nas linhas 8 e 11, a acurácia no conjunto de teste

que era de 68,25% quando utilizamos apenas o modelo retornado, passa a ser de 76,49% quando classificamos utilizando um comitê composto por todos os modelos treinados. Para os exemplos em que utilizamos os *teachers* `Double` e `ALTCT`, obtemos, respectivamente, acurácias 70,79% e 74,26%.

```
1  from sklearn import preprocessing
2  from keras.datasets import mnist
3  import numpy as np
4  (train_X, train_y), (test_X, test_y) = mnist.load_data()
5  train_X = np.reshape(train_X, (-1, 784))
6  test_X = np.reshape(test_X, (-1, 784))
7  scaler = preprocessing.StandardScaler()
8  scaler.fit(train_X)
9  scaler.transform(train_X, copy = False)
10 scaler.transform(test_X, copy = False)
11 le = preprocessing.LabelEncoder()
12 le.fit(train_y)
13 train_y = le.transform(train_y)
14 test_y = le.transform(test_y)
```

Figura A.3: Exemplo de como carregar e realizar o pré-processamento do *dataset* mnist via código Python.

B
Detalhes Sobre a Prova do Lema 5 de (DASGUPTA et al., 2019)

O algoritmo de ensino de (DASGUPTA et al., 2019) é semelhante ao algoritmo $\texttt{OSCT}_{\text{base}}$ apresentado na Seção 3.3, sendo a única diferença na etapa de amostragem: o primeiro envia o exemplo x para o *learner* quando seu peso cruza um limite aleatório T_x que é amostrado no início do algoritmo a partir de uma distribuição exponencial com taxa $\lambda = \ln(N/\delta)$, em que N é um limite superior em $|\mathcal{H}|$ e o parâmetro δ é a probabilidade de falha do algoritmo.

O Lema 5 de (DASGUPTA et al., 2019) afirma que a probabilidade de falha de seu algoritmo é no máximo δ. A prova fixa uma hipótese h e então considera o primeiro ponto no tempo em que seu peso é pelo menos 1. Então, a prova afirma que

$$Pr[h \text{ não ser coberto} \mid \text{peso de } h \geq 1] = \prod_{x \in \text{erros}(h)} Pr[w(x) \leq T_x] = \quad \text{(B-1)}$$

$$\prod_{x \in \text{erros}(h)} exp(-\lambda w(x)) \leq exp(-\lambda) = \frac{\delta}{N}. \quad \text{(B-2)}$$

Finalmente, a prova toma o *union bound* sobre todas as hipóteses para obter o limite δ.

Primeiro, não ficou claro se os pesos $w(x)$ são determinísticos ou estocásticos nas equações acima. Além disso, em ambos os casos também não está claro como a primeira igualdade de (B-1) poderia ser obtida. Em particular, se os pesos são determinísticos, o que faz mais sentido dadas as seguintes igualdades, (B-1) parece ignorar o fato de que o condicionamento (peso de $h \geq 1$) afeta a probabilidade de distribuição do limiares e, aparentemente, também está assumindo uma independência entre eventos que não parece ser válida.

Conforme gentilmente explicado em (ANôNIMO; DASGUPTA, 2020), para fins de análise, deve-se levar em conta que a probabilidade de não enviar o exemplo x na rodada t é dada por

$$Pr[T_x > W_x^t | T_x > W_x^{t-1}] = Pr[T_x > W_x^t - W_x^{t-1}] = Pr[T_x > D_x^t] = exp(-\lambda \cdot D_x^t).$$

Embora esse *insight* seja realmente muito útil para esclarecer como o algoritmo pode ser analisado, ele não ajuda a superar o problema – discutido logo após o Lema 3.3.1 – da falta de independência entre as rodadas.

Nesse sentido, usando esse *insight* e nosso lema técnico relativo a Ber-

noulli adaptado (Lema 3.3.2), explicamos como derivar uma prova formal para o Lema 5 de (DASGUPTA et al., 2019).

Uma demonstração semelhante à do Lema 3.3.3 pode ser usada para estabelecer o Lema 5, com a única diferença em como o limite em $Z^t = Pr[X^t = 0|\mathcal{F}_{t-1}]$, dado pela desigualdade (3-2), é derivado.

No que se segue, provamos que

$$Pr[X^t = 0|\mathcal{F}_{t-1}] \leq exp(-\lambda \cdot D^t(h)).$$

Primeiro, note que

$$Z^t = Pr[X^t = 0|\mathcal{F}_{t-1}] = Pr[W_x^t < T_x \text{ para todo } x \in \text{erros}(h)|\mathcal{F}_{t-1}.]$$

Lembre-se que nenhum exemplo em erros(h) é enviado durante o histórico $\mathcal{F}_{t-1}$. Assim, se um ponto $\mathbf{t} = (T_1, \ldots, T_m)$ no espaço de probabilidade dos limites leva ao histórico $\mathcal{F}_{t-1}$, então $T_x > W_x^{t-1}$ para todo $x \in$ erros(h), em que W_x^{t-1} é o peso do exemplo x até o final do histórico $\mathcal{F}_{t-1}$. Além disso, assumindo um *learner* determinístico, cada ponto $\mathbf{t}' = (T'_1, \ldots, T'_m)$ satisfaz simultaneamente:

- $T'_x > W_x^{t-1}$ para todo $x \in$ erros(h)
- $T'_x = T_x$ para todo $x \notin$ erros(h)

resultando no mesmo histórico. Portanto,

$$Pr[W_x^t < T_x \text{ para todo } x \in \text{erros}(h)|\mathcal{F}_{t-1}] =$$
$$Pr[W_x^t < T_x \text{ para todo } x \in \text{erros}(h)|W_x^{t-1} < T_x \text{ para todo } x \in \text{erros}(h)] \leq$$
$$\prod_{x \in \text{erros}(h)} exp(-\lambda \cdot (W_x^t - W_x^{t-1})) = exp(-\lambda \cdot D^t(h))$$

C
Melhoria na Garantia com Base na Qualidade das Hipóteses

Seja $\texttt{OSCT}_{\text{base}}(\texttt{N}, \omega)$ o algoritmo de *teacher* da Figura 3.1, tornando suas entradas explícitas. Para obter a garantia do Teorema 3.3, refinamos esse algoritmo base em alguns passos.

Primeiro, Lemas 3.3.1 e 3.3.3 na verdade fornecem a seguinte garantia para $\texttt{OSCT}_{\text{base}}(\texttt{N}, \omega)$, mesmo no caso em que $\mathcal{H}$ não contém uma hipótese correta, ou seja, $h^* \notin \mathcal{H}$. Observe que neste caso a interação entre *learner* e $\texttt{OSCT}_{\text{base}}(\texttt{N}, \omega)$ para devido a um dos dois eventos:

1. $\texttt{OSCT}_{\text{base}}(\texttt{N}, \omega)$ retorna FALHA.
2. O *learner* deixa de enviar hipóteses, pois não há hipóteses $\mathcal{H}$ consistentes com os exemplos fornecidos pelo *teacher*.

Em contraste, quando há uma hipótese correta em $\mathcal{H}$, a interação para quando $\texttt{OSCT}_{\text{base}}(\texttt{N}, \omega)$ retorna FALHA ou uma hipótese correta.

Teorema C.1 *Considere uma classe de hipóteses $\mathcal{H}$ e exemplos $\mathcal{X}$, possivelmente sem nenhuma hipótese em $\mathcal{H}$ que esteja correta em todos os exemplos. Suponha que todas as hipóteses em $\mathcal{H}$ tenham no máximo ω erros. Então, o algoritmo de teacher $\texttt{OSCT}_{base}(\texttt{N}, \omega)$ sempre envia no máximo*

$$\mathcal{TS}(\mathcal{H}) \cdot O(\log \omega \log N)$$

exemplos. Além disso, se $N \geq |\mathcal{H}|$, então com probabilidade de pelo menos $1 - \frac{1}{N}$:

- *Se houver um classificador em $\mathcal{H}$ que esteja correto em todos os exemplos $\mathcal{X}$, $\texttt{OSCT}_{base}(\texttt{N}, \omega)$ retorna um desses classificadores.*
- *Caso contrário, a interação entre o learner e $\texttt{OSCT}_{base}(\texttt{N}, \omega)$ para porque o primeiro parou de enviar hipóteses.*

Algoritmo agnóstico ao número de hipóteses. O próximo passo na construção de nosso algoritmo final é converter o algoritmo $\texttt{OSCT}_{\text{base}}(\texttt{N}, \omega)$ em um que não precise de uma estimativa explícita N do número de hipóteses em $\mathcal{H}$; isso é feito usando uma abordagem simples de "adivinhar e dobrar". Na verdade, será conveniente fornecer uma "estimativa inicial" N^0 para garantir que o algoritmo falhe com probabilidade de no máximo $\frac{1}{N^0}$ (em vez de $\frac{1}{|\mathcal{H}|}$), independentemente do tamanho de $\mathcal{H}$.

Algoritmo `OSCT`

Input: Exemplos $\mathcal{X}$, estimativa inicial $N^0 \geq 1$ do número de hipóteses, peso inicial $\omega \geq 0$

Seja i_0 o menor inteiro i tal que $2^{2^i} \geq N^0$

Para fases $i = i_0, i_0 + 1, \ldots$

- Interaja com o *learner* executando $\texttt{OSCT}_{\text{base}}(2^{2^i}, \omega)$ até que a interação pare
- Se $\texttt{OSCT}_{\text{base}}(2^{2^i}, \omega)$ retornar FALHA, passa para a próxima fase
- Se $\texttt{OSCT}_{\text{base}}(2^{2^i}, \omega)$ retornar um classificador correto, então simplesmente **retorne** ele
- Se a interação parar porque o *learner* não enviou nenhuma hipótese adicional, **retorne** NENHUMA HIPÓTESE CORRETA

Figura C.1: Algoritmo `OSCT`

Teorema C.2 *Considere as suposições do Teorema C.1. Seja* $N^0 \geq 1$ *e seja* $\bar{n} := \max\{|\mathcal{H}|, N^0\}$. *Existe um evento* G *que ocorre com probabilidade pelo menos* $1 - \frac{1}{\bar{n}}$ *tal que, sob* G*, o algoritmo de teacher* $\texttt{OSCT}(N^0, w)$ *envia no máximo*

$$\mathcal{TS}(\mathcal{H}) \cdot O(\log w \log \bar{n})$$

exemplos. Além disso, se houver uma hipótese em $\mathcal{H}$ *que esteja correta em todos os exemplos* $\mathcal{X}$*, sob* G *o algoritmo retornará uma dessas hipóteses.*

Prova. Seja $\bar{i}$ o menor inteiro i tal que $2^{2^i} \geq \bar{n}$, e seja G o evento que `OSCT` para na fase $\bar{i}$. Se `OSCT` atingir a fase $\bar{i}$, ele executa $\texttt{OSCT}_{\text{base}}(\texttt{N}, \omega)$ com $N \geq \bar{n} \geq |\mathcal{H}|$ e, portanto, a garantia do Teorema C.1 está assegurada. Neste caso, com probabilidade de pelo menos $1 - \frac{1}{\bar{n}}$ o algoritmo $\texttt{OSCT}(N^0, w)$ pára nesta fase $\bar{i}$, seja por retornar uma hipótese correta em $\mathcal{H}$ (se existir), ou então retornando NENHUMA HIPÓTESE CORRETA. Assim, o evento G de parar pela fase $\bar{i}$ ocorre com probabilidade de pelo menos $1 - \frac{1}{\bar{n}}$.

Para limitar o número de exemplos enviados por `OSCT` sob o evento G usamos o Teorema C.1 em cada fase $i \leq \bar{i}$ para obter

$$\sum_{i \leq \bar{i}} \mathcal{TS}(\mathcal{H}) \cdot O(\log w \log 2^{2^i}) = O(\mathcal{TS}(\mathcal{H}) \log w) \sum_{i \leq \bar{i}} 2^i \leq O(\mathcal{TS}(\mathcal{H}) \log w \log \bar{n}),$$

como desejado.

Finalmente, quando há uma hipótese correta em $\mathcal{H}$ o algoritmo não retorna NENHUMA HIPÓTESE CORRETA e portanto, sob G, retorna uma hipótese correta (pela fase $\bar{i}$). Isso conclui a prova. ■

Algoritmo aprimorado. Agora podemos finalmente apresentar o algoritmo que fornece o Teorema 3.3. Assumimos, sem perda de generalidade, que $\log\log m$ é um número inteiro, caso contrário, simplesmente preenchemos a entrada com exemplos fictícios adicionais. Defina os intervalos $B_0 = [1, 2)$ e $B_i = [2^{2^{i-1}}, 2^{2^i})$ para $i = 1, \ldots, \log\log m$.

Algoritmo $\mathcal{A}_{\text{err}}$
Entrada: Exemplos $\mathcal{X}$

1. Defina os valores $N_i^0 := 10(\log\log m - i + 1)^2$
2. Inicializar algoritmos $\mathcal{A}_0 := \texttt{OSCT}(N_0^0, 2)$ e $\mathcal{A}_i := \texttt{OSCT}(N_i^0, 2^{2^i})$ para $i = 1, \ldots, \log\log m$
3. Para cada intervalo de tempo:
   - Receba a hipótese h do *learner*. Se h não tiver erros, **retorne**
   - Seja $\bar{i}$ tal que $|\text{erros}(h)| \in B_{\bar{i}}$
   - Envie a hipótese h para $\mathcal{A}_{\bar{i}}$, e receba um conjunto de exemplos em $\mathcal{X}$ (Se receber FALHA, **retorne** FALHA)
   - Envie o conjunto de exemplos para o *learner*

Figura C.2: Algoritmo $\mathcal{A}_{\text{err}}$

Mostramos que o algoritmo da Figura C.2 satisfaz as garantias do Teorema 3.3.

Prova.[Prova do Teorema 3.3] Seja E o evento ao qual o limite do Teorema C.2 é assegurado para todos os algoritmos $\mathcal{A}_0, \ldots, \mathcal{A}_{\log\log m}$. Pelo *union bound*, a probabilidade de E não valer é no máximo

$$\sum_{i=0}^{\log\log m} \frac{1}{N_i^0} = \frac{1}{10} \cdot \sum_{j=1}^{\log\log m+1} \frac{1}{j^2} \leq \frac{1}{10} \cdot \sum_{j=1}^{\infty} \frac{1}{j^2} = \frac{1}{10} \cdot \frac{\pi^2}{6} \leq \frac{1}{5}.$$

Assim, basta mostrar que sempre que E é assegurado, o algoritmo $\mathcal{A}_{\text{err}}$ (Figura C.2) retorna uma hipótese correta e o número de exemplos enviados é limitado superiormente pela quantidade desejada.

Para tanto, seja $\mathcal{H}_i$ o conjunto de hipóteses em $\mathcal{H}$ cujo número de erros pertence ao intervalo B_i e seja T_i o conjunto de passos de tempo em que uma hipótese foi enviada ao algoritmo $\mathcal{A}_i$. Restringindo-se às rodadas em T_i, temos uma execução de $\mathcal{A}_i$ sobre uma sequência adaptativa de hipóteses em $\mathcal{H}_i$. Por definição, nenhum dos $\mathcal{H}_i$'s contém um classificador correto. Portanto, sob o evento E, o Teorema C.2 garante que a execução de cada $\mathcal{A}_i$ pare após enviar no máximo $O(\mathcal{TS}(\mathcal{H}_i) \log 2^{2^i} \log \bar{n}_i)$ exemplos, porque o *learner* não envia mais hipóteses de $\mathcal{H}_i$ (em que $\bar{n}_i := \max\{|\mathcal{H}_i|, N_i^0\}$). Seja $I \subseteq \{0, \ldots, \log\log m\}$ o conjunto de i's tal que $\mathcal{H}_i$ não seja vazio, e observe que para índices $i \notin I$ o

algoritmo $\mathcal{A}_i$ nunca é invocado. Assim, sob E, após no máximo

$$\sum_{i \in I} O(\mathcal{TS}(\mathcal{H}_i) \cdot 2^i \log \bar{n}_i) \tag{C-1}$$

exemplos, o *learner* deve enviar uma hipótese em $\mathcal{H} \setminus \bigcap_i \mathcal{H}_i$, ou seja, uma hipótese correta, caso em que o algoritmo $\mathcal{A}_{\text{err}}$ a retornará.

Para concluir a prova, precisamos apenas realizar algumas manipulações algébricas para expandir o limite superior (C-1). Para simplificar a notação, seja $n_i := |\mathcal{H}_i|$. Como a função $x \mapsto \log(x+1)$ é subaditiva[1] sobre os reais não negativos, temos isso para todo $a \geq 0$ e $b \geq 1$

$$\log(\max\{a, b\}) \leq \log(a + (b-1) + 1) \overset{subadd}{\leq} \log(a+1) + \log b.$$

Aplicando isso a $\log \bar{n}_i$ e usando a monotonicidade do *teaching set*, obtemos que $\mathcal{TS}(\mathcal{H}_i) \leq \mathcal{TS}(\mathcal{H})$ e, portanto, o número de exemplos sob E dado por (C-1) é no máximo

$$O(\mathcal{TS}(\mathcal{H})) \cdot \left[\sum_{i \in I} 2^i \log(n_i + 1) \; + \; \sum_{i \in I} 2^i \log N_i^0 \right]. \tag{C-2}$$

Expandindo o segundo somatório temos (definindo $\ell := \log \log m$ para simplificar a notação)

$$\sum_{i \in I} 2^i \log N_i^0 \leq \sum_{i=0}^{\ell} 2^{i+1} \log\left(10(\ell - i + 1)^2\right) = 2^{\ell+2} \sum_{i=0}^{\ell} \frac{\log\left(10(\ell - i + 1)^2\right)}{2^{\ell - i + 1}} \leq O(2^{\ell+2}) = O(\log m),$$

em que a última desigualdade segue do fato de que a série $\sum_{x=1}^{\infty} \frac{\log(10x^2)}{2^x}$ converge para uma constante. Usando essa observação em (C-2), obtemos

$$\#\text{exemplos enviados sob } E \;=\; O(\mathcal{TS}(\mathcal{H})) \cdot \left(\log m + \sum_{i \in I} 2^i \log(n_i + 1) \right).$$

Isso conclui a prova do Teorema 3.3. ■

[1] Uma função subaditiva é uma função $f : A \mapsto B$ tal que $\forall x, y \in A, f(x+y) \leq f(x) + f(y)$.

D
Ensino no Caso não Realizável

Nesta seção apresentamos um algoritmo de *teacher* para o caso em que a melhor hipótese em $\mathcal{H}$ tem $k > 0$ erros e provamos o Teorema 3.4. Esse algoritmo faz uso do algoritmo proposto em (BUCHBINDER; NAOR, 2009) para o problema *Online Fractional Generalized Set Cover*, que tratamos logo adiante.

D.1
Online Fractional Generalized Set Cover

A versão *off-line* desse problema é encontrar uma solução ótima para o seguinte problema de cobertura

$$\begin{aligned} \min \ & \sum_{e\in[m]} w_e \\ \text{s.t. } & \langle a^t, w\rangle \geq b_t \quad \forall t = 1,\ldots,\ell \\ & w \in [0,1]^m, \end{aligned} \tag{D-1}$$

em que os vetores a^t têm entradas 0/1. Isso pode ser interpretado da seguinte forma: existe um *ground set* de ℓ elementos e m conjuntos (a^t_i é 1 se o conjunto i contém o elemento t), e o objetivo é encontrar uma pequena seleção fracionária w dos conjuntos de modo a cobrir cada elemento t pelo menos b_t vezes.

Na versão *online* do problema as restrições $\langle a^1, w\rangle \geq b_1$, $\langle a^2, w\rangle \geq b_2$, $\ldots$, $\langle a^\ell, w\rangle \geq b_\ell$ são reveladas de uma por uma, e o algoritmo precisa manter uma sequência de soluções fracionais viáveis $w^1 \leq w^2 \leq \ldots \leq w^\ell$ que é ponto a ponto não decrescente (ou seja, o algoritmo não pode desmarcar itens). Mais precisamente, no início do jogo o algoritmo não tem informação, e na rodada t o adversário revela a restrição $\langle a^t, w\rangle \geq b_t$. Usando apenas as informações vistas até agora, o algoritmo precisa encontrar uma solução $w^t \in [0,1]^m$ que seja ponto a ponto $w^t \geq w^{t-1}$ e satisfaça essa nova restrição (a monotonicidade garante que as restrições anteriores também sejam satisfeitas). O objetivo é minimizar o tamanho da solução na última rodada, ou seja, $\sum_e w^\ell_e$.

Várias variações e generalizações deste problema foram estudadas, mas o importante (para nós) é que Buchbinder e Naor (2009) deram uma $O(\log m)$-aproximação para esse problema, ou seja, o custo $\sum_e \bar{w}^\ell_e$ da sequência retornada $(\bar{w}^t)_{t\in[\ell]}$ é sempre no máximo $O(\log m)$ vezes o custo da solução ótima w^* para o problema *offline* (D-1).

D.2
Algoritmo e Análise

A ideia principal por trás do algoritmo é a equivalência entre *k-extended teaching sets* e soluções inteiras[1] para (D-1). Para ver isso, lembre-se que k é o número mínimo de erros de uma hipótese em $\mathcal{H}$, e seja $\mathcal{H}_{bad} \subseteq \mathcal{H}$ o conjunto das hipóteses com mais de k erros. Lembre-se que a seleção $\mathcal{X}'$ de exemplos é um *k-extended teaching set* se toda hipótese $h \in \mathcal{H}_{bad}$ erra pelo menos $k+1$ exemplos do conjunto $\mathcal{X}'$.

Mas se considerarmos $a(h) \in \{0,1\}^{\mathcal{X}}$ como o vetor indicador dos exemplos em que a hipótese h está errada, então os *k-extended teaching sets* correspondem precisamente a 0/1 soluções $w \in \{0,1\}^m$ do problema (D-1) com restrições $\langle a(h), w\rangle \geq k+1$ para todos $h \in \mathcal{H}_{bad}$.

Como o *teacher* não conhece as hipóteses $\mathcal{H}$, a ideia é:

1. Construa a versão online da instância adicionando as restrições $\langle a(H_t), w\rangle \geq k+1$ uma a uma conforme as hipóteses são recebidas do *learner*;

2. Use o algoritmo de (BUCHBINDER; NAOR, 2009) para calcular as seleções fracionárias $W^1, W^2, \ldots \in [0,1]^{\mathcal{X}}$ de exemplos para cada intervalo de tempo;

3. Use W^t como ponderação para amostrar os exemplos, que são então enviados para o *learner*.

O elemento adicional é que o número mínimo de erros k não é conhecido a priori. Para lidar com isso, o algoritmo mantém um limite inferior κ de k, inicializado em 0. Por definição nosso *learner* sempre retorna uma hipótese que comete o menor número possível de erros sobre os exemplos enviados até então. Portanto, se ele enviar uma hipótese H_t com err erros nos exemplos enviados até agora, então sabemos que err também é um limite inferior em k, e é usado para atualizar κ.

O algoritmo final é apresentado na Figura D.1. Usaremos $\widetilde{W}^t \in \{0,1\}^m$ para denotar o vetor de incidência dos exemplos enviados ao final do tempo t, daí $\langle a(h), \widetilde{W}^t\rangle$ é o número de exemplos já enviados até então em que h está errado.

Análise principal. Começamos a coletar algumas observações simples sobre o algoritmo.

[1]Solução em que apenas valores inteiros são atribuídos às variáveis de decisão.

Entrada: Exemplos $\mathcal{X}$, número de de hipóteses do *learner* $n = |\mathcal{H}|$, estimativa N de n

Inicialize o limite inferior $\kappa = 0$ de k (e deixe $W^0 = 0$)

Para cada iteração de tempo $t = 1, 2, \ldots$

1. Receba a hipótese H_t do *learner*
2. **(Atualização do limite inferior)** Defina κ como o máximo entre seu valor atual e o número de exemplos enviados até agora em que H_t está errado, ou seja, $\kappa \leftarrow \max\{\kappa, \langle a(H_t), \widetilde{W}^{t-1}\rangle\}$
3. Se o número total de erros de H_t for κ (que é $\leq k$), **retorne** H_t
4. **(Atualização dos pesos)** Envie a restrição $\langle a(H_t), w\rangle \geq \kappa + 1$ para o algoritmo de *fractional generalized set cover* de (BUCHBINDER; NAOR, 2009), e receba dele uma solução fracionária atualizada W^t
5. **(Amostragem)** Seja $D_e^t = W_e^t - W_e^{t-1}$. Para cada exemplo e, lance uma moeda com viés $\min\{D_e^t \cdot 10 \log(Nm), 1\}$, e envie este exemplo para o *learner* em caso de cara
6. Seja $\widetilde{W}^t = \{0,1\}^{\mathcal{X}}$ o vetor indicador dos exemplos enviados até agora
7. Se $\langle a(H_t), \widetilde{W}^t\rangle < \kappa + 1$, **retorne** FALHA

Figura D.1: Algoritmo de *teacher* para o ensino no caso não realizável.

Lema D.2.1 *Independentemente da estimativa N, o algoritmo da Figura D.1 satisfaz:*

1. *κ é sempre no máximo k;*
2. *Se o algoritmo retornar no Passo 3, ele retornará um classificador com o número mínimo de erros totais k;*
3. *O algoritmo termina em no máximo $(k+1)n + 1$ passos de tempo.*

Prova. O primeiro item já foi discutido anteriormente, e o segundo item segue diretamente do primeiro.

Para provar o terceiro item, chame de *fase* a sequência de passos de tempo que teve o mesmo κ após a atualização no Passo 2. Visto que κ aumenta em pelo menos 1 quando iniciamos uma nova fase, existem no máximo $k + 1$ fases. Para obter o limite superior desejado no número de passos de tempo, basta argumentar que, se uma hipótese h aparecer duas vezes na mesma fase, o algoritmo retornará nessa segunda aparição. Para isso, considere um tempo t em uma fase $\bar{\kappa}$. Por causa da etapa de atualização de κ, no início do tempo t a hipótese H_t está errada em no máximo $\bar{\kappa}$ exemplos enviados até agora, ou seja, $\langle a(H_t), \widetilde{W}^{t-1}\rangle \leq \bar{\kappa}$. Porém, ao final do tempo t, após o

envio dos novos exemplos, ou este número aumenta para pelo menos $\bar{\kappa}+1$ (i.e., $\langle a(H_t), \widetilde{W}^t\rangle \geq \bar{\kappa}+1$), ou o algoritmo retorna. De qualquer forma, isso significa que essa mesma hipótese H_t não pode aparecer em uma etapa de tempo posterior na mesma fase. Isso conclui a prova. ■

Para concluir a análise precisamos mostrar que com alta probabilidade o algoritmo de fato retorna no Passo 3 e também envia um número pequeno de exemplos. Isso requer a desigualdade de concentração técnica do Lema D.3.1, que lida com todas as correlações no processo. Para melhorar a legibilidade, adiamos a declaração e prova deste lema para a Seção D.3.

Lema D.2.2 *Independentemente da estimativa N, com probabilidade de pelo menos $1-\frac{1}{2m^2N}$, o número de exemplos enviados pelo algoritmo é $O(\mathcal{TS}_k \log m \cdot \log Nm)$.*

Prova. Seja τ uma variável aleatória denotando o último passo de tempo do algoritmo. Primeiro afirmamos que $\sum_e W_e^\tau \leq O(\mathcal{TS}_k \log m)$, ou seja, o número total de exemplos fracionados selecionados é no máximo $O(\mathcal{TS}_k \log m)$. Para ver isso, fixe qualquer cenário e observe que a instância do problema *Online Fractional Generalized Set Cover* enviada ao algoritmo de (BUCHBINDER; NAOR, 2009) é

$$\begin{aligned} \min \ & \sum_{e\in\mathcal{X}} w_e \\ \text{s.t. } & \langle a(H_t), w\rangle \geq \kappa_t + 1 \quad \forall t = 1,\ldots,\tau \\ & w \in [0,1]^{\mathcal{X}}, \end{aligned} \tag{D-2}$$

em que κ_t denota o valor de κ na Etapa 4 no tempo t. Como do lema anterior todos os κ_t's são no máximo k, vemos que qualquer *k-extended teaching set* dá uma solução viável (inteira) para esse problema e, portanto, seu valor ótimo é no máximo $\mathcal{TS}_k$. A garantia do algoritmo de (BUCHBINDER; NAOR, 2009) garante que sua solução satisfaz

$$\sum_e W_e^\tau \leq [\text{valor ótimo de (D-2)}] \cdot O(\log m) \leq O(\mathcal{TS}_k \log m),$$

comprovando a afirmação.

Agora limitamos superiormente o número de exemplos enviados pelo algoritmo, ou seja, $\sum_e \widetilde{W}_e^\tau$. Seja $\mathcal{F}_t$ o histórico do algoritmo até o tempo t; mais formalmente, $\mathcal{F}$ é a σ-álgebra gerada por $H_1,\ldots,H_{t+1}$ e $\widetilde{W}^1,\ldots,\widetilde{W}^t$.[2]

[2]O último termo H_{t+1} é necessário apenas quando o *learner* não é uma função determinística dos exemplos que recebe.

Seja $\widetilde{D}_e^t := \widetilde{W}_e^t - \widetilde{W}_e^{t-1}$ o indicador dos exemplos enviados no tempo t, então $\sum_e \widetilde{D}_e^t$ conta o número de exemplos enviados no momento t. O número total "condicionalmente esperado" de exemplos enviados pelo algoritmo é

$$\begin{aligned}
\sum_{t\le\tau}\sum_e \mathbb{E}[\widetilde{D}_e^t \mid \mathcal{F}_{t-1}] &\le (10\log Nm)\sum_{t\le\tau}\sum_e D_e^t && \text{(pela etapa de amostragem [item 5])}\\
&\le (10\log Nm)\sum_e W_e^\tau && \text{(pela definição de } D_e^t\text{)}\\
&\le O(\mathcal{TS}_k \log m \cdot \log Nm). && \text{(pela afirmação anterior)}
\end{aligned}$$

Então, por concentração de medida, devemos ter que o número real de exemplos enviados $\sum_e \widetilde{W}_e^\tau = \sum_{t\le\tau}\sum_e \widetilde{D}_e^t$ está próximo dessa expectativa condicional. Como existem correlações entre os vetores $\widetilde{D}^t$'s (e também o tempo de parada τ) não podemos aplicar diretamente desigualdades de concentração padrões. Mas empregando a parte 1 do Lema D.3.1, temos (definindo $\beta := cst \cdot \mathcal{TS}_k \log m \log Nm$ para uma constante suficientemente grande cst)

$$\Pr\left(\sum_e \widetilde{W}_e^\tau \ge 2\beta\right) = \Pr\left(\sum_{t\le\tau}\sum_e \widetilde{D}_e^t \ge 2\beta \text{ e } \sum_{t\le\tau}\sum_e \mathbb{E}[\widetilde{D}_e^t \mid \mathcal{F}_{t-1}] \le \beta\right) \le e^{-\frac{3}{14}\beta} \le \frac{1}{2m^2N}.$$

Isso dá um limite superior no número de exemplos enviados pelo algoritmo, conforme desejado. ■

Lema D.2.3 *Se $N \ge \max\{n, 4\}$, o algoritmo retorna* Falha *com probabilidade de no máximo $\frac{1}{2m}$.*

Prova. Conforme definido no lema anterior, seja τ o último passo de tempo do algoritmo, κ_τ o valor de κ nesse momento e $\mathcal{F}_t$ a σ-álgebra gerada pelo histórico $H_1, \ldots, H_{t+1}$ e $\widetilde{W}^1, \ldots, \widetilde{W}^t$. Lembre-se que $\langle a(h), W^t\rangle = \sum_{e\in\text{erros}(h)} W_e^t$ é a massa fracionária dos exemplos no tempo t em que a hipótese h está errada, e $\langle a(h), \widetilde{W}^t\rangle = \sum_{e\in\text{erros}(h)} \widetilde{W}_e^t$ é o número de exemplos realmente enviados no tempo t em que h está errada.

Quando o algoritmo retorna Falha temos $\sum_{e\in\text{erros}(H_\tau)} \widetilde{W}_e^\tau \le \kappa_\tau$ (um número muito pequeno de exemplos errados enviado) mas no Passo 4 sempre temos $\sum_{e\in\text{erros}(H_\tau)} W_e^\tau \ge \kappa_\tau + 1$ (uma boa massa de exemplos errados); assim, basta mostrar que

$$\Pr\left(\sum_{e\in\text{erros}(H_\tau)} \widetilde{W}_e^\tau \le \kappa_\tau \quad \text{e} \quad \sum_{e\in\text{erros}(H_\tau)} W_e^\tau \ge \kappa_\tau + 1\right) \le \frac{1}{2m}.$$

Tomando um *union bound* sobre todos os valores possíveis $h \in \mathcal{H}$ para H_τ, basta provar que para cada h

$$\Pr\left(\sum_{e\in\text{erros}(h)} \widetilde{W}_e^\tau \leq \kappa_\tau \quad \text{e} \sum_{e\in\text{erros}(h)} W_e^\tau \geq \kappa_\tau + 1\right) \leq \frac{1}{2mn}. \tag{D-3}$$

Para provar esse limite novamente, a ideia é provar uma desigualdade similar para cada valor fixo $L \in \{0,\ldots,k\}$ de κ_τ usando uma desigualdade de concentração e então tomar um *union bound* sobre todos os L's; como no lema anterior, para a concentração usaremos a decomposição em incrementos $W_e^\tau = \sum_{t\leq\tau} D_e^t$, e similarmente para $\widetilde{W}_e^\tau$. Infelizmente, devido ao truncamento $\min\{\cdot,1\}$ na etapa de amostragem, precisamos trabalhar com as vaiáveis aleatórias Z_e^t e $\widetilde{Z}_e^t$, que são versões modificadas de D_e^t e $\widetilde{D}_e^t$.

Portanto, para cada exemplo $e \in \text{erros}(h)$, seja τ_e a primeira vez em que o truncamento ocorre na etapa de amostragem para e, ou seja, é o primeiro tempo t em que $D_e^t > \frac{1}{10\log Nm}$ (e defina $\tau_e = \tau$ quando não ocorrer truncamento). Então Z_e^t é a sequência $D_e^1, D_e^2, \ldots$ parada logo antes de τ_e, ou seja, $Z_e^t = D_e^t \cdot \mathbf{1}(t < \tau_e)$. Definimos $\widetilde{Z}_e^t$ da mesma forma, mas com respeito a $\widetilde{D}_e^t$.

A próxima proposição relaciona essas variáveis com o evento de interesse (D-3).

Proposição D.2.1 *Para todo $e \in erros(h)$ temos:*

1. $\sum_t \widetilde{Z}_e^t \leq \sum_{t\leq\tau} \widetilde{D}_e^t - 1$
2. $\sum_t Z_e^t \geq \sum_{t\leq\tau} D_e^t - 1$
3. *Para todo t, $\mathbb{E}[\widetilde{Z}_e^t \mid \mathcal{F}_{t-1}] = (10\log Nm)\, Z_e^t$.*

Em particular, o lado esquerdo de (D-3) *é limitado superiormente por*

$$\Pr\left[\sum_{e\in erros(h)}\sum_t \widetilde{Z}_e^t \leq \kappa_\tau - |erros(h)| \quad e \sum_{e\in erros(h)}\sum_t \mathbb{E}[\widetilde{Z}_e^t \mid \mathcal{F}_{t-1}] \geq (10\log Nm)\Big(\kappa_\tau - |erros(h)| + 1\Big)\right]$$

Prova. Pela construção do esquema de amostragem temos que quando ocorre o truncamento o exemplo é enviado, ou seja, $\widetilde{D}_e^{\tau_e} = 1$, e portanto,

$$\sum_{t\leq\tau} \widetilde{D}_e^t \geq \sum_{t<\tau_e} \widetilde{D}_e^t + 1 = \sum_{t<\tau_e} \widetilde{Z}_e^t + 1,$$

provando a primeira parte da proposição. Para a segunda parte, como o peso total W_e^τ é no máximo 1,

$$\sum_{t\leq\tau} D_e^t \leq \sum_{t<\tau_e} D_e^t + 1 = \sum_{t<\tau_e} Z_e^t + 1.$$

Para a terceira parte, como o evento $\mathbf{1}(t < \tau_e)$ é determinado pelo histórico $\mathcal{F}_{t-1}$, condicionado a tal histórico temos

$$\begin{aligned}\mathbb{E}[\widetilde{Z}_e^t \mid \mathcal{F}_{t-1}] &= \mathbf{1}(t < \tau_e) \cdot \mathbb{E}[\widetilde{D}_e^t \mid \mathcal{F}_{t-1}] \\ &= \mathbf{1}(t < \tau_e)(10 \log Nm) D_e^t \\ &= (10 \log Nm) Z_e^t,\end{aligned}$$

sendo a segunda igualdade válida porque, dado qualquer histórico, se o truncamento acontecer no tempo t, ambos os lados são iguais a 0; e se isso não acontecer, então, pelo procedimento de amostragem: $\mathbb{E}[\widetilde{D}_e^t \mid \mathcal{F}_{t-1}] = (10 \log Nm) D_e^t$. Isso conclui a prova. ■

Agora, para um inteiro L, defina o evento

$$E_L := \left[\sum_{e \in \text{erros}(h)} \sum_t \widetilde{Z}_e^t \leq L \quad \text{e} \quad \sum_{e \in \text{erros}(h)} \sum_t \mathbb{E}[\widetilde{Z}_e^t \mid \mathcal{F}_{t-1}] \geq (10 \log Nm)\Big(L{+}1\Big) \right].$$

Para $L \geq 0$, parte 2 da desigualdade de concentração do Lema D.3.1 mostra que $\Pr(E_L) \leq \frac{1}{4Nm^2}$. Como o evento nunca ocorre para $L < 0$ devido à não negatividade dos $\widetilde{Z}_e^t$'s, pelo *union bound* obtemos que a probabilidade no final da Proposição D.2.1 pode ser limitado superiormente por $\sum_{L \in \{-n, \ldots, k\}} \Pr(E_L) \leq \frac{k+1}{4Nm^2} \leq \frac{1}{2mn}$, em que a última desigualdade usa o fato $k + 1 \leq 2k \leq 2m$ e a suposição $N \geq n$. Isso prova a desigualdade (D-3) e conclui a prova do lema. ■

Recapitulando. Se o número de hipóteses n for conhecido (e pelo menos 4), podemos executar o algoritmo da Figura D.1 com $N = n$ e tomar um *union bound* sobre os Lemas D.2.2 e D.2.3 para obter que, com probabilidade de pelo menos $1 - \frac{1}{m}$, o algoritmo retorna uma hipótese de erro mínimo e envia no máximo $O(\mathcal{TS}_k \log m \cdot \log mn)$ exemplos.

Para remover a suposição de que n é conhecido, execute o algoritmo com estimativas crescentes $N = N^i$ com $N^i := \frac{1}{m} 2^{2^i}$ para $i = \lceil \log \log 4m \rceil, \ldots, \lceil \log \log nm \rceil$. A probabilidade de qualquer uma dessas execuções enviar mais exemplos do que o limite do Lema D.2.2 é de no máximo $\frac{1}{2m} \sum_{i \geq 1} \frac{1}{2^{2^i}} \leq \frac{1}{2m}$. Assim, com probabilidade de pelo menos $1 - \frac{1}{2m}$ essas execuções enviam no máximo no máximo

$$\sum_{i=1}^{\lceil \log \log nm \rceil} O(\mathcal{TS}_k \log m \cdot \log mN_i) = O(\mathcal{TS}_k \log m) \sum_{i=1}^{\lceil \log \log nm \rceil} 2^i \leq O(\mathcal{TS}_k \log m \cdot \log mn)$$

exemplos. Além disso, do Lema D.2.3, a última dessas execuções não falha (portanto, retorna uma hipótese de erro mínimo) com probabilidade de pelo

menos $1 - \frac{1}{2m}$. Portanto, concluímos a prova do Teorema 3.4 fazendo um *union bound* sobre essas garantias.

D.3 Lema Técnico

A seguinte desigualdade de concentração pode ser considerada como uma extensão da desigualdade de Bernstein para o caso de martingale que é adequado para o nosso uso.

Lema D.3.1 *Considere uma filtragem $\mathcal{F}_1 \subseteq \mathcal{F}_2 \subseteq \ldots$, e uma sequência adaptada de vetores aleatórios $X^1, \ldots, X^T \in \{0,1\}^m$ que podem ser correlacionados, mas que satisfaça o seguinte: condicionadas a $\mathcal{F}_{t-1}$, as coordenadas de X^t são independentes: para todo $x \in \{0,1\}^n$,*

$$\Pr(X^t = x \mid \mathcal{F}_{t-1}) = \prod_i \Pr(X_i^t = x_i \mid \mathcal{F}_{t-1}).$$

Seja $Y^t = \sum_i X_i^t$. Então, para qualquer tempo de parada τ adaptado para $\{\mathcal{F}_t\}_t$, $\alpha > 0$, e $\lambda \geq \alpha + 1$

$$\Pr\left(\sum_{t \leq \tau} Y^t \geq \alpha + 2\lambda \ e \ \sum_{t \leq \tau} \mathbb{E}[Y^t \mid \mathcal{F}_{t-1}] \leq \alpha + \lambda\right) \leq e^{-\frac{3}{14}\lambda},$$

e

$$\Pr\left(\sum_{t \leq \tau} Y^t \leq \alpha \ e \ \sum_{t \leq \tau} \mathbb{E}[Y^t \mid \mathcal{F}_{t-1}] \geq \alpha + \lambda\right) \leq e^{-\frac{3}{14}\lambda}.$$

Vamos precisar da Desigualdade de Freedman.

Lema D.3.2 (Desigualdade de Freedman (FREEDMAN, 1975)) *Considere uma filtragem $\mathcal{F}_1 \subseteq \mathcal{F}_2 \subseteq \ldots$ e uma sequência de diferenças de martingale adaptada $M_1, \ldots, M_T$ tomando valores em $[-1, 1]$. Seja $V = \sum_t \mathbb{E}[M_t^2 \mid \mathcal{F}_{t-1}]$ a variância previsível. Então, para todo $\lambda, v > 0$,*

$$\Pr\left(\sum_t M_t \leq -\lambda \ e \ V \leq v\right) \leq \exp\left(-\frac{\lambda^2}{2(v + \lambda/3)}\right),$$

e o mesmo limite superior vale para o evento $[\sum_t M_t \geq \lambda \ e \ V \leq v]$.

Prova.[Prova do Lema D.3.1] Primeiro provamos o lema para $m = 1$, caso em que temos $Y^t = X_1^t$. Para simplificar a notação, deixe $\mathbb{E}_{t-1}[\cdot]$ denotar a expectativa condicional $\mathbb{E}[\cdot \mid \mathcal{F}_{t-1}]$ ($\mathbb{E}_0[\cdot] := E[\cdot]$). A ideia é aplicar a Desigualdade de Freedman à sequência $Y^t - \mathbb{E}_{t-1}Y^t$, mas para controlar a variação previsível precisamos definir a parada do processo.

Seja τ' o menor t tal que $\mathbb{E}_0 Y^1 + \ldots + \mathbb{E}_{t-1} Y^t \geq \alpha + \lambda$ (e $\tau' = \infty$ se não houver tal t). Defina o processo truncado $\overline{Y}^t := Y^t \cdot \mathbf{1}(t \leq \tau) \cdot \mathbf{1}(t \leq \tau')$. Afirmamos que esse processo truncado limita superiormente o original, ou seja,

$$\Pr\left(\sum_{t\leq\tau} Y^t \geq \alpha + 2\lambda \text{ e } \sum_{t\leq\tau} \mathbb{E}_{t-1} Y^t \leq \alpha + \lambda\right) \leq \Pr\left(\sum_{t} \overline{Y}^t \geq \alpha + 2\lambda \text{ e } \sum_{t} \mathbb{E}_{t-1} \overline{Y}^t \leq \alpha + \lambda\right) \tag{D-4}$$

$$\Pr\left(\sum_{t\leq\tau} Y^t \leq \alpha \text{ e } \sum_{t\leq\tau} \mathbb{E}_{t-1} Y^t \geq \alpha + \lambda\right) \leq \Pr\left(\sum_{t} \overline{Y}^t \leq \alpha \text{ e } \sum_{t} \mathbb{E}_{t-1} \overline{Y}^t \geq \alpha + \lambda\right). \tag{D-5}$$

Para justificar a primeira desigualdade, observe que para cada cenário que satisfaz o lado esquerdo, o truncamento τ' não entra em ação e, portanto, o cenário satisfaz o lado direito. Para a segunda desigualdade, uma vez que $\sum_t \overline{Y}^t \leq \sum_{t\leq\tau} Y^t$, para cada cenário que satisfaça o lado esquerdo temos $\sum_t \overline{Y}^t \leq \alpha$; além disso, para cada cenário no lado esquerdo, temos $\sum_t \mathbb{E}_{t-1} \overline{Y}^t \geq \alpha + \lambda$: já que nesse caso $\sum_{t\leq\tau} \mathbb{E}_{t-1} Y^t \geq \alpha + \lambda$ temos $\tau' \leq \tau < \infty$ e, portanto, $\sum_t \overline{Y}^t = \sum_{t\leq\tau'} \overline{Y}^t \geq \alpha + \lambda$, a última desigualdade por definição de τ'. Então todo cenário que pertence ao lado esquerdo também pertence ao lado direito, provando assim (D-5).

Agora defina a sequência de diferença de martingale $M_t := \overline{Y}^t - \mathbb{E}_{t-1} \overline{Y}^t$ e observe que

$$\Pr\left(\sum_{t} \overline{Y}^t \geq \alpha + 2\lambda \text{ e } \sum_{t} \mathbb{E}_{t-1} \overline{Y}^t \leq \alpha + \lambda\right) \leq \Pr\left(\sum_{t} M_t \geq \lambda\right) \tag{D-6}$$

$$\Pr\left(\sum_{t} \overline{Y}^t \leq \alpha \text{ e } \sum_{t} \mathbb{E}_{t-1} \overline{Y}^t \geq \alpha + \lambda\right) \leq \Pr\left(\sum_{t} M_t \leq -\lambda\right). \tag{D-7}$$

Além disso, por causa do truncamento, podemos limitar a variância previsível desse martingale: em nosso caso atual $m = 1$ temos $|\overline{Y}^t| \leq 1$ e, portanto,

$$\mathbb{E}_{t-1} M_t^2 = \mathrm{Var}(\overline{Y}^t \mid \mathcal{F}_{t-1}) \leq \mathbb{E}_{t-1} (\overline{Y}^t)^2 \leq \mathbf{1}(t \leq \tau') \cdot \mathbb{E}_{t-1} Y^t.$$

Desse modo, usando a definição do tempo de parada τ' e o fato $\mathbb{E}_{t-1} Y^t \leq 1$, a variância previsível é $V := \sum_t \mathbb{E}_{t-1} M_t^2 \leq \sum_{t\leq\tau'} \mathbb{E}_{t-1} Y^t \leq \alpha + \lambda + 1$.

Finalmente, aplicando o Lema D.3.2 a $(M_t)_t$ com $v = \alpha + \lambda + 1$ obtemos

$$\Pr\left(\sum_{t} M_t \geq \lambda \text{ e } V \geq \alpha + \lambda + 1\right) \leq \exp\left(-\frac{\lambda^2}{2(\alpha + \lambda + 1 + \lambda/3)}\right) \leq e^{-\frac{3}{14}\lambda},$$

em que a última desigualdade usa a suposição $\lambda \geq \alpha + 1$. O mesmo limite também vale para $\sum_t M_t \leq -\lambda$. Encadeando esses limites com as desigualdades (D-4)-(D-7). Isso conclui a prova para o caso $m = 1$.

Para o caso $m > 1$ simplesmente revelamos as variáveis aleatórias

$X_1^t, X_2^t, \ldots$ uma a uma, e aplicamos o resultado para o caso $m = 1$ a essa sequência. Mais precisamente, indexamos o tempo usando (t, i) e a ordem lexicográfica $(1, 1), \ldots, (1, n), (2, 1), \ldots$, e usamos $prec(t, i)$ para denotar o predecessor de (t, i), e definimos $\widetilde{Y}^{t,i} := X_i^t$. Seja $\widetilde{\mathcal{F}}_{t,m} := \mathcal{F}_t$. Seja $\widetilde{\mathcal{F}}_{t,i}$, para $i < m$, a σ-álgebra gerada por $\mathcal{F}_{t-1}$ e $X_1^t, \ldots, X_i^t$. As principais propriedades dessa filtragem são: 1) $\widetilde{Y}^{t,i}$ é $\widetilde{\mathcal{F}}_{t,i}$-mensurável; 2) pela independência condicional das variáveis leatórias $(X_1^t, \ldots, X_n^t)|_{\mathcal{F}_{t-1}}$, para $i < m$ condicionada em $\widetilde{\mathcal{F}}_{prec(t,i)}$ é o mesmo que condicionar em $\mathcal{F}_{t-1}$, ou mais precisamente

$$\mathbb{E}[\widetilde{Y}^{t,i} \mid \widetilde{\mathcal{F}}_{prec(t,i)}] = \mathbb{E}[\widetilde{Y}^{t,i} \mid \ F_{t-1}] = \mathbb{E}_{t-1} X_i^t. \tag{D-8}$$

Observe que $\sum_{(t,i)\preceq(\tau,m)} \widetilde{Y}^{t,i} = \sum_{t\leq\tau} \sum_{i\leq m} X_i^t = \sum_{t\leq\tau} Y^t$, e usando (D-8) temos $\sum_{(t,i)\preceq(\tau,m)} \mathbb{E}[\widetilde{Y}^{t,i} \mid \widetilde{\mathcal{F}}_{prec(t,i)}] = \sum_{t\leq\tau} \sum_{i\leq m} \mathbb{E}_{t-1} X_i^t = \sum_{t\leq\tau} \mathbb{E}_{t-1} Y^t$. Portanto, aplicando o lema atual para o caso $m = 1$ a $(\widetilde{Y}^{t,i})_{t,i}$ e o tempo de parada (τ, m) prova o lema por completo. Isso conclui a prova. ■

E
Resultado de Inaproximabilidade em Algoritmos Aleatórios para Ensinar um *Black Box Learner*

Nesta seção reformulamos o limite inferior da competitividade de qualquer algoritmo de tempo polinomial para *online set cover* de (KORMAN, 2004a) em termos de aproximação do *teaching set* para um *teacher* randomizado (que seleciona exemplos de forma aleatória) lidando com um *black-box learner*. Isso associado ao resultado na seção 3.4.1 mostra uma separação entre a eficiência de ensino alcançável com um *learner* adversário em comparação com a eficiência de ensino alcançável com um *learner* de transição suave.

Sejam $\mathcal{I} = \{a_1, \ldots, a_N\}$ e $\mathcal{S} = \{S_1, \ldots, S_M\}$, respectivamente, o conjunto de itens e a família de conjuntos em uma instância para o problema (*offline*) de *Set Cover*, ou seja, $S_i \subseteq \mathcal{I}$ para cada $i = 1, \ldots, M$.

Para $i = 1, \ldots, N-1$, sejam $\mathcal{I}^{(i)}, \mathcal{S}^{(i)}$ cópias distintas de $\mathcal{I}$ e $\mathcal{S}$, ou seja, temos que

- para cada $i \neq i'$, $\mathcal{I}^{(i)} \cap \mathcal{I}^{(i')} = \emptyset$;
- para cada i, $\mathcal{S}^{(i)} \in 2^{\mathcal{I}^{(i)}}$;
- para cada i, ℓ, t, temos que $a_\ell^{(i)} \in S_t^{(i)}$ se e somente se (sse) $a_\ell \in S_t$;

portanto, em particular, para cada i, ℓ, j, t é válido que $a_\ell^{(i)} \in S_t^{(j)}$ sse $i = j$ e $a_\ell \in S_t$.

A seguir, sem perda de generalidade, vamos assumir que N é uma potência de 2.

Seja $\mathcal{X} = \{E_\ell^{(j)} \mid j = \frac{N}{2}, \ldots, N-1,\ \ell = 1, \ldots, M\} \cup \{\tilde{E}\}$ o conjunto de exemplos da nossa instância, no qual $\tilde{E}$ é um exemplo especial cujo papel será esclarecido em breve.

Para cada $i = 1, \ldots, N-1$ e $t = 1, \ldots, N$ definimos uma hipótese $h_t^{(i)} : \mathcal{X} \mapsto \{0, 1\}$ sendo $h_t^{(i)}(\tilde{E}) = 0$ e $h_t^{(i)}(E_\ell^{(j)}) = 1$ se e somente se $a_t \in S_\ell$ e $i \in \{j, \lfloor j/2 \rfloor, \ldots, 1\}$.

Definimos então duas hipóteses adicionais $h^* : \mathcal{X} \mapsto \{0, 1\}$ e $\tilde{h} : \mathcal{X} \mapsto \{0, 1\}$:

$$\begin{aligned} h^*(E) &= 0, \text{ para cada } E \in \mathcal{X}, \\ \tilde{h}(E) &= 1, \text{ sse } E = \tilde{E}. \end{aligned}$$

Observe que $\tilde{h}$ difere de h^* em apenas um exemplo, ou seja, $\tilde{E}$ é o único exemplo que *cobre* $\tilde{h}$. Equivalentemente, com relação à análise do *learner* de transição suave, $\tilde{h}$ está à distância 1 de h^*.

Agora definimos $N/2$ classes de hipóteses. Para isso, considere uma árvore binária com nós $N-1$, que são rotulados $1, \ldots, N-1$, partindo da raiz para as folhas e da esquerda para a direita em cada nível, de modo que a raiz seja o nó 1, seu filho esquerdo é o nó 2, o filho direito é o nó 3, e assim por diante, com as folhas, que da esquerda para a direita são os nós $\frac{N}{2}, \frac{N}{2}+1, \ldots, N-1$. Pense em cada nó j como contendo a sequência de hipóteses $(h_1^{(j)}, \ldots, h_N^{(j)})$. A classe de hipóteses $\mathcal{H}_i$, para $i = N/2, \ldots, N-1$ contém as hipóteses h^* e $\tilde{h}$, e também toda hipótese que pertence a algum nó (sequência) no caminho da raiz até a folha i.

Seja $\mathcal{U} = \cup_i \mathcal{H}_i$. A instância de *machine teaching* é definida pelo conjunto de exemplos $\mathcal{X}$ e o universo de hipóteses $\mathcal{U}$, em que, sem o conhecimento do *teacher*, a classe de hipóteses do *learner* está contida.unknowingly to the teacher, the learner's hypothesis class lies. Em particular, a classe de hipóteses do *learner* será um dos conjuntos $\mathcal{H}_i$.

Para explicar o comportamento do *learner*, precisamos definir uma sequência ρ_i associada à classe $\mathcal{H}_i$. Essa sequência começa com $\tilde{h}$ e então é obtida concatenando as sequências no caminho da raiz até a folha i da árvore. Finalmente, h^* é anexado à sequência ρ_i.

Considere agora um *learner* (sendo o adversário) que com probabilidade uniforme escolhe $i \in \{N/2, N/2+1, \ldots, N-1\}$ –equivalentemente tem a classe de hipóteses $\mathcal{H}_i$– e dá ao algoritmo de ensino hipóteses consistentes seguindo a sequência ρ_i, ignorando aquelas inconsistentes com exemplos recebidos. Observe que, em qualquer caso, a primeira hipótese apresentada será $\tilde{h}$ e qualquer algoritmo de ensino deve cobrí-la com $\tilde{E}$. Portanto, podemos assumir que as primeiras interações são $\tilde{h}, \tilde{E}, h_1^{(1)}$.

O ponto chave aqui é que todo algoritmo de ensino deve usar $\tilde{E}$ e, portanto, se comporta exatamente como se a instância não contivesse $\tilde{h}$ e $\tilde{E}$. Essa última instância é a instânca que pode ser provada como tendo um limite inferior de $\mathcal{TS} \log m \log n$ usando o argumento de (KORMAN, 2004a): (1) Todo algoritmo determinístico em expectativa sobre as possíveis $N/2$ escolhas da sequência ρ_i, gastará pelo menos $\frac{\log N}{2} k$ exemplos, em que k é igual ao tamanho do *teaching set* de tamanho mínimo, ou seja, o tamanho da solução ótima para a instância off-line do problema de *Set Cover*, pois, sem conhecer a escolha de ρ_i, o exemplo escolhido pelo algoritmo para cobrir alguma nova hipótese

tem probabilidade 1/2 de cobrir alguma hipótese apresentada posteriormente; (2) Um algoritmo aleatório, visto como uma distribuição sobre algoritmos determinísticos, em média sobre as $N/2$ sequências de entrada possíveis, usa menos de $\frac{\log N}{2}k$ exemplos com probabilidade menor que 2/3. Aproveitando essas duas observações e a redução de SAT para Set Cover de (RAZ; SAFRA, 1997) obtém-se o resultado:

Teorema E.1 *Existe uma constante $d > 0$ tal que é assegurado: Se existe um algoritmo randomizado de tempo polinomial para ensinar um black-box learner que garante usar menos que $d \cdot \mathcal{TS} \log n \log m$ exemplos em cada instância em que o conjunto de exemplos tem tamanho m, a classe de hipóteses do learner tem tamanho n, então $NP \subset BPP$.*

Isso mostra que mesmo que o *learner* comece com a hipótese $\tilde{h}$ próxima da hipótese alvo h^*, não há como o algoritmo de *teacher* construir um *teaching set* de tamanho $o(\mathcal{TS} \log m \log n)$.

F
Modelo de *Learner* Aleatório

F.1
Prova do Lema 3.4.2

Para cada exemplo $e \in \mathcal{X}$, seja c_e (resp. $\tilde{c}_e$) o número de hipóteses de $\mathcal{H}_t$ (resp. $\tilde{\mathcal{H}}_t$) cobertas por e.

Seja e^* um exemplo que cobre o maior número de hipóteses em $\mathcal{H}_t$ de modo que $c_{e^*} = c^*(\mathcal{H}_t)$. Assim, $Pr[c_{\tilde{e}} < \frac{1}{12}c^*(\mathcal{H}_t)]$ pode ser limitada superiormente pela probabilidade de algum exemplo b, com $c_b < \frac{1}{12}c_{e^*}$, cobrir pelo menos o mesmo número de hipóteses em $\tilde{\mathcal{H}}_t$ cobertas por e^*, ou seja, $\tilde{c}_b \geq \tilde{c}_{e^*}$.

Para um exemplo fixo b, com $c_b < \frac{1}{12}c_{e^*}$, seja E_b o evento no qual $\tilde{c}_b \geq \tilde{c}_{e^*}$. Basta mostrar que $Pr[E_b] \leq 2/m^5$, já que podemos tomar o *union bound*, considerando todos os exemplos, para mostrar que a probabilidade de E_b acontecer para algum b é limitada superiormente por $2/m^4$.

Seja z um número real e sejam F^1 e F^2 os eventos em que $\tilde{c}_{e^*} \leq z$ e $\tilde{c}_b \geq z$, respectivamente. Temos que $E_b \subseteq F^1 \cup F^2$. De fato, se E_b acontece e $\tilde{c}_{e^*} \leq z$, então F^1 acontece. Caso contrário, se E_b acontecer e $\tilde{c}_{e^*} > z$, então F^2 acontecerá. Assim, $Pr[E_b] \leq Pr[F^1] + Pr[F^2]$ de modo que seja suficiente obter limites superiores em $Pr[F^1]$ e $Pr[F^2]$. No que segue, usamos $z = \frac{1}{2}\frac{k \cdot c_{e^*}}{|\mathcal{H}_t|}$, em que $k \geq 40\,\mathcal{TS}(\ cH_t, h^*) \ln m$.

Seja $\tilde{\mathcal{H}}_t = (h_1, \ldots, h_k)$ e, para $i = 1, \ldots, k$, seja X_i^e uma variável aleatória definida como

$$X_i^e = \begin{cases} 1 & \text{se } e \in \text{erros}(h_i) \\ 0 & \text{se } e \notin \text{erros}(h_i). \end{cases}$$

Então, $\Pr(X_i^e = 1) = \frac{c_e}{|\mathcal{H}_t|}$. Seja $X^e = \sum_{i=1}^k X_i^e = \tilde{c}_e$ e $\mu_e = k \cdot \frac{c_e}{|\mathcal{H}_t|} = \sum_{i=1}^k E[X_i^e] = E[X^e]$,

Pelo limite de Chernoff (ver, por exemplo, (MITZENMACHER; UPFAL, 2005, (4.5))), temos que, para qualquer $0 < \delta < 1$,

$$\Pr\left(\frac{\tilde{c}_e}{k} \leq (1-\delta)\frac{c_e}{|\mathcal{H}_t|}\right) = \Pr\left(X^e \leq (1-\delta)\mu_e\right) \leq e^{-\frac{\mu_e \delta^2}{2}} = e^{-\frac{c_e}{|\mathcal{H}_t|}\frac{k\delta^2}{2}} \quad \text{(F-1)}$$

De (F-1), com $\delta = 1/2$ e $e = e^*$, obtemos

$$Pr[F^1] = \Pr\left(\frac{\tilde{c}_{e^*}}{k} \leq \frac{1}{2}\frac{c_{e^*}}{|\mathcal{H}_t|}\right) \leq e^{-\frac{c_{e^*}}{|\mathcal{H}_t|}\frac{k}{8}}, \quad \text{(F-2)}$$

Não é difícil ver que deve haver um exemplo que cubra pelo menos $|\mathcal{H}_t|/(\mathcal{TS}(\mathcal{H}_t, h^*))$ hipóteses em $\mathcal{H}_t$, portanto, em particular, $c_{e^*} \geq \frac{|\mathcal{H}_t|}{\mathcal{TS}(\mathcal{H}_t, h^*)}$.

Por causa de $k \geq 40\,\mathcal{TS}(\mathcal{H}_t, h^*)\ln m$, segue que

$$Pr[F^1] = \Pr\left(\frac{\tilde{c}_{e^*}}{k} \leq \frac{1}{2}\frac{c_{e^*}}{|\mathcal{H}_t|}\right) \leq \frac{1}{m^5}, \tag{F-3}$$

ou seja, com alta probabilidade o exemplo e^* também aparecerá como aquele que cobre uma grande fração de hipóteses em $\tilde{\mathcal{H}}_t$.

Para limitar $Pr[F^2]$, contamos com o seguinte limite de Chernoff (ver (MITZENMACHER; UPFAL, 2005, (4.3))). Para qualquer $R \geq 6\mu_e$, temos

$$\Pr(X^e > R) \leq 2^{-R}. \tag{F-4}$$

Supondo que $c_b < \frac{1}{12}c_{e^*}$ temos $\frac{1}{2}\mu_{e^*} > 6\mu_b$. Seja $R = \frac{1}{2}\mu_{e^*}$. Então, por (F-4) temos

$$Pr[F^2] = \Pr\left(\frac{\tilde{c}_b}{k} \geq \frac{1}{2}\frac{c_{e^*}}{|\mathcal{H}_t|}\right) \leq 2^{-\frac{k}{2}\frac{c_{e^*}}{|\mathcal{H}_t|}} \leq \frac{1}{m^5}, \tag{F-5}$$

em que na última desigualdade usamos a hipótese $k \geq 10\mathcal{TS}(\mathcal{H}_t, h^*)\log_2 m$. Isso conclui a prova.

F.2
Prova do Lema 3.4.3

Seja $C(\mathcal{H})$ o número total de exemplos selecionados pelo algoritmo em um conjunto inicial de hipóteses $\mathcal{H}$, de cardinalidade n.

Podemos mostrar por indução na cardinalidade de $\mathcal{H}$ que $C(\mathcal{H}) \leq \frac{1}{\alpha}\ln(|\mathcal{H}|)$. A desigualdade vale para $|\mathcal{H}| = 1$. Podemos focar na etapa de indução.

Seja e o primeiro exemplo enviado pelo algoritmo e seja c_e o número de hipóteses que x cobre em $\mathcal{H}$.

Seja $\mathcal{H}_1$ o conjunto de hipóteses consistentes após o envio do exemplo e. Temos que $\mathcal{H}_1$ tem cardinalidade $n - c_e \leq n - \alpha\, c^*(\mathcal{H})$, e temos que:

$$C(\mathcal{H}) \leq 1 + C(\mathcal{H}_1).$$

Desde $\mathcal{H}_1 \subset \mathcal{H}$ e $c^*(\mathcal{H}) \geq \frac{|\mathcal{H}|}{\mathcal{TS}(\mathcal{H}, h^*)}$, temos o seguinte limite inferior em $\mathcal{TS}(\mathcal{H}, h^*)$

$$\mathcal{TS}(\mathcal{H}, h^*) \geq \max\left\{\frac{|\mathcal{H}|}{c^*(\mathcal{H})}, \mathcal{TS}(\mathcal{H}_1, h^*)\right\}.$$

Portanto, juntando as duas últimas desigualdades temos

$$\begin{aligned}\frac{C(\mathcal{H})}{\mathcal{TS}(\mathcal{H},h^*)} &\leq \frac{c^*(\mathcal{H})}{|\mathcal{H}|}+\frac{C(\mathcal{H}_1)}{\mathcal{TS}(\mathcal{H}_1,h^*)} \leq \frac{1}{\alpha}\frac{c_e}{|H|}+\frac{C(\mathcal{H}_1)}{\mathcal{TS}(\mathcal{H}_1,h^*)} \\ &\leq \frac{1}{\alpha}\frac{c_e}{|\mathcal{H}|}+\frac{1}{\alpha}\ln(|\mathcal{H}_1|)=\frac{1}{\alpha}\frac{c_e}{n}+\frac{1}{\alpha}\ln(n-c_e)\leq\frac{1}{\alpha}\ln(n).\end{aligned}$$

Isso conclui a prova.

G
Experimentos Computacionais: Maiores Detalhes

Damos mais detalhes sobre os experimentos relatados na Seção 3.6, em que comparamos o número de exemplos exigidos por `OSCT` e `NIT` para atingir um determinado nível de acurácia.

Ambiente Computacional. Nossos experimentos foram executados em um processador Intel Core i7-7700HQ com 16G RAM. Usamos o Python 3.7.3 com as seguintes bibliotecas: numpy 1.16.4, pandas 0.23.4, scikit-learn 0.20.1 e lightgbm 2.3.1.

Os códigos e os conjuntos de dados estão disponíveis em <https://github.com/sfilhofreitas/TeachingWithLimitedKnowledge>.

***Datasets* e Pré-Processamento.** Nossos datasets selecionados atendem aos seguintes critérios: não possuem valores faltantes e possuem até 60.000 exemplos rotulados. Este limite de tamanho foi usado para evitar experimentos demorados.

Os conjuntos de dados foram pré-processados para converter atributos categóricos em numéricos: se um atributo categórico assume k valores diferentes, ele é substituído por k atributos binários. Para isso usamos o método `get_dumies` da biblioteca pandas. Isso foi feito porque nossos *learners* não lidam diretamente com atributos categóricos.

Implementações dos *learners*. Usamos as classes `lightgbm.LGBMClassifier` e `sklearn.ensemble.RandomForestClassifier`, com seus parâmetros padrões, para implementar LGBM e Random Forest, respectivamente.

Implementações dos *teachers*. O Algoritmo `OSCT` é explicado e analisado na Seção 3.3 para o cenário em que o *learner* sempre retorna uma hipótese consistente com os exemplos enviados até agora. Porém, na prática não temos essa garantia. Para executar `OSCT` para uma configuração mais geral, usamos a seguinte definição para o conjunto erros(h):

$$\text{erros}(h) = \{x \in \mathcal{X} \mid h \text{ falha em } x \text{ e } x \text{ não foi enviado para o } \textit{learner}\}.$$

em vez da definição original:

$$\text{erros}(h) = \{x \in \mathcal{X} \mid h \text{ falha em } x\}.$$

Com o requisito adicional na definição de erros(h) evitamos enviar o mesmo exemplo mais de uma vez. Note que para o caso realizável $h^* \in \mathcal{H}$, essa

nova definição é equivalente à original já que, nesse caso, as hipóteses recebidas pelo *teacher* não incorrem em erros nos exemplos enviados até então.

O outro *teacher* que consideramos, `NIT`, procura um pequeno conjunto aleatório de exemplos para os quais o *learner*, quando treinado nesse conjunto, atinge uma acurácia maior do que um determinado percentual sobre a acurácia obtida utilizando todo o conjunto de dados em consideração. Para isso ele escolhe uma ordem aleatória dos exemplos do conjunto de dados e então procura o menor valor de ℓ para o qual a amostra composta pelos primeiros ℓ exemplos dessa ordem satisfaz a propriedade desejada. Para evitar cálculos muito caros, o valor de ℓ é aumentado usando um passo igual a $0,5\%$ do tamanho do conjunto de dados.

Resultados. Para levar em conta a aleatoriedade do *teacher*, para cada tripla possível (*teacher*, *learner*, *dataset*) tivemos 30 execuções. Para uma combinação fixa (*learner*,*dataset*), a primeira hipótese disponível para ambos `OSCT` e `NIT`, para sua $i-$ésima execução, é obtida treinando o *learner* com os primeiros 1% dos exemplos da ordem aleatória empregada por `NIT` para sua $i-$ésima execução.

As colunas `Avg Size` das tabelas G.1 e G.2 mostram o número médio de exemplos (em relação ao tamanho do *dataset*) necessários, para cada par *teacher*-*learner*, para obter uma acurácia sobre o *dataset* completo maior que 99% daquela obtida quando o *learner* é treinado/testado em todo o conjunto de dados. A Tabela G.1 apresenta os resultados para *Random Forests* enquanto a Tabela G.2 apresenta os resultados para LGBM. Por exemplo, para o conjunto de dados `mnist`, a combinação `OSCT`-LGBM necessitou em média (ao longo das 30 execuções) ser treinada com ≈ 3.780 (6,3% de 60.000) exemplos rotulados para obter acurácia (com relação ao *dataset* inteiro) maior que 99% do obtido quando o LGBM é treinado e depois testado nos 60.000 exemplos do `mnist`. Por outro lado, para a combinação `NIT`-LGBM, ≈ 37.620 (62,7 % de 60.000) exemplos são necessários em média. Notamos que os resultados apresentados na Tabela 3.1, para o objetivo de 99%, são as médias dos valores mostrados nas colunas `Avg Size` das Tabelas G.1 e G.2.

Podemos observar uma vantagem significativa de `OSCT`: ele tem um desvio padrão menor que `NIT` e, em termos de número de exemplos necessários, superou `NIT` em todas as combinações, exceto (`crédito_cartão`, LGBM). Esses resultados sugerem que o algoritmo `OSCT` é potencialmente uma opção muito competitiva para uma convergência rápida de métodos de classificação.

Dataset	Size	OSCT-Random Forest		NIT-Random Forest	
		Avg. Size (%)	StdDev (%)	Avg. Size (%)	StdDev (%)
avila	10,430	25.5	0.8	74.4	3.1
bank_marketing	41,188	22.4	0.3	89.2	0.7
car	1,728	24	1	77.7	4.8
credit_card	30,000	54	0.4	94	0.4
crowdsourced	10,545	19.5	0.5	82.4	1.8
Electrical_grid	10,000	1.1	0	2.1	0.6
Firm_Teacher	10,796	52.7	0.7	95.3	0.6
HTRU	17,898	7.1	0.2	46.6	2.5
mnist	60000	23.5	0.4	82.1	0.7
mushroom	8,124	1.2	0.1	4.4	1.7
nursery	12,960	13.2	0.5	59.1	2.6
Sensorless_drive_diagnosis	58,509	3	0.1	11.5	1.1
Average		20.6	0.4	59.9	1.7

Tabela G.1: Coluna `Avg Size` mostra o número médio de exemplos (em porcentagem em relação ao tamanho do conjunto de dados) necessários para cada par *teacher-learner* obter uma precisão sobre o conjunto de dados completo maior que 99% disso alcançado quando o *learner* (Random Forest) é treinado/testado no conjunto de dados completo. Cada valor é a média de 30 execuções.

Dataset	Size	OSCT-LGBM		NIT-LGBM	
		Avg. Size (%)	StdDev (%)	Avg. Size (%)	StdDev (%)
avila	10,430	1.1	0	1.1	0
bank_marketing	41,188	12.7	1.4	27.1	2.3
car	1,728	14.3	0.6	47.1	4
credit_card	30,000	26.3	4.3	16.7	1.6
crowdsourced	10,545	8.7	0.2	74.8	2.4
Electrical_grid	10,000	1	0	1	0
Firm_Teacher	10,796	25.7	1.1	64.7	4.2
HTRU	17,898	5.5	0.3	19.5	3.2
mnist	60000	6.3	0.1	62.6	1.2
mushroom	8,124	1.1	0.1	4.5	1.4
nursery	12,960	2.7	0.2	9.7	1
Sensorless_drive_diagnosis	58,509	1.7	0.1	7.5	0.7
Average		8.9	0.7	28	1.8

Tabela G.2: Coluna `Avg Size` mostra o número médio de exemplos (em porcentagem em relação ao tamanho do conjunto de dados) necessários para cada par *teacher-learner* obter uma precisão sobre o conjunto de dados completo maior que 99% disso alcançado quando o *learner* (LGBM) é treinado/testado no conjunto de dados completo. Cada valor é a média de 30 execuções.

H
Estudo Experimental: Detalhes Adicionais

H.1
Transformações dos Datasets

Realizamos algumas transformações nos conjuntos de dados.

- A ordem dos exemplos em cada *dataset* foi embaralhada aleatoriamente.
- Cada *dataset* (com tamanho m) foi dividido em um *conjunto de treinamento* (com tamanho $0,7 \cdot m$) e um *conjunto de teste* (com tamanho $0,3 \cdot m$). A divisão garante que ambos os conjuntos tenham (aproximadamente) a mesma distribuição de classe que o conjunto original[1].
- Em cada *dataset*, os rótulos receberam valores de 0 a $n_{classes} - 1$.
- Cada atributo não numérico com $n_{categorias}$ valores possíveis foram convertidos em $n_{categorias}$ atributo binários, com um deles 1 e todos os outros 0.
- Cada atributo numérica foi normalizado, isto é, para cada atributo, subtraímos seu valor médio e (se não for constante) dividimos por seu desvio padrão[2].

H.2
Fonte dos Datasets

A maioria dos *datasets* foi obtida dos repositórios *OpenML Repository*, *UCI Machine Learning Repository* e *Kaggle*. A maior parte deles tem o tipo de licença "Public Domain". Abaixo cumprimos pedidos adicionais de citações:

- Diabetes130US: Beata Strack, Jonathan P. DeShazo, Chris Gennings, Juan L. Olmo, Sebastian Ventura, Krzysztof J. Cios e John N. Clore, "Impact of HbA1c Measurement on Hospital Readmission Rates: Analysis of 70,000 Clinical Database Patient Records", BioMed Research International, vol. 2014, Artigo ID 781670, 11 páginas, 2014.
- covtype: copyright para Jock A. Blackard e Colorado State University.
- veículo_sensIT: M. Duarte e Y. H. Hu.

[1] O *dataset minist* já é disponibilizado dividido, então os tamanhos relativos do conjunto de treinamento e do conjunto de dados nesse caso não são $0,7$ e $0,3$

[2] Essa transformação é feita com base nos dados do conjunto de treinamento e é aplicado tanto no conjunto de treinamento quanto no conjunto de teste

– MiniBooNE: B. Roe et al., "Boosted Decision Trees, an Alternative to Artificial Neural Networks" <https://arxiv.org/abs/physics/0408124>, Nucl. Instrum. Metanfetamina A543, 577 (2005).

– cifar_10: Alex Krizhevsky (2009) Learning Multiple Layers of Features from Tiny Images, Tech Report.

– GTSRB-HueHist: <https://www.openml.org/d/41990>

– aloi: <http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html>

H.3 TCT × OSCT - Comparações Adicionais

Testamos algumas variações do `OSCT` com o objetivo de melhorar seu desempenho. Em uma primeira tentativa, aumentamos o palpite inicial N sobre o tamanho da classe de hipóteses do *learner*. Ao fazer isso, impedimos que o *learner* envie poucos exemplos nas primeiras rodadas. A Figura H.1 mostra os resultados para $N = 2^{0.005m}$, o que garante que o *learner* envie aproximadamente $0.005m$ exemplos incorretos por rodada antes que a estimativa de N seja atualizada. Observamos que esses novos resultados são muito semelhantes aos apresentados na Figura 4.1, isto é, não houve impacto significativo.

Em outra tentativa, adotamos a mesma abordagem utilizada pelo `TCT` para selecionar o modelo de classificação final: entre os vários modelos construídos pelo `OSCT` dentro do limite de tempo, retornamos aquele com o maior limite inferior para o intervalo de confiança de acurácia (95%). Isso não acarreta custo adicional, pois o `OSCT`, por construção, classifica todos os exemplos do conjunto de treinamento para selecionar os novos exemplos que são enviados ao *learner*. Os resultados desse teste são mostrados na Figura H.2, na qual pode ser observada uma melhora significativa do `OSCT`, em particular com relação à sua estabilidade. Apesar dessa melhora, o `TCT` ainda supera o `OSCT` para todos os *learners* e para cada unidade de tempo (normalizado) $t \in [0, 1]$.

H.4 Tabelas e Gráficos Adicionais

A Tabela H.1 mostra a acurácia média (para 4 execuções) obtida pelos *learners* em cada *dataset* para `TCT` ($\alpha = 0.2$), `Double`, `OSCT` e a acurácia média obtida usando todo o conjunto de dados (coluna "Full"). Os resultados para `OSCT` referem-se à versão do algoritmo que retorna o modelo com a melhor estimativa de acurácia (Figura H.2). A coluna "Tempo Limite"denota o tempo médio em segundos necessário para treinar todo o conjunto de dados. Esse é

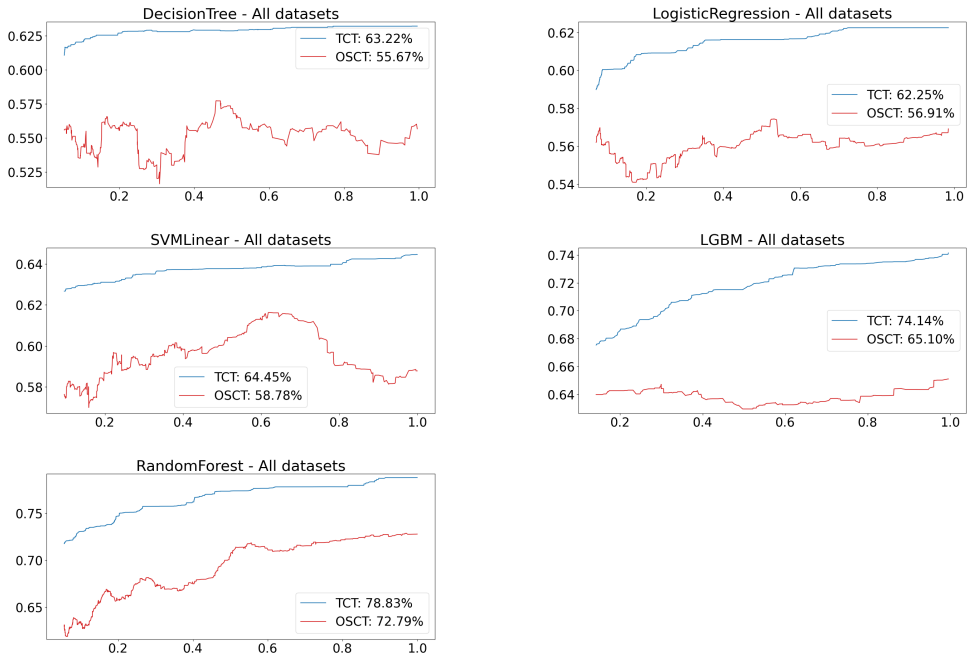

Figura H.1: Acurácias médias no conjunto de teste ao longo do tempo normalizado para `TCT` e `OSCT` (iniciando com $N = 2^{0.005m}$). Os números próximos aos seus rótulos são suas acurácias no final do tempo limite normalizado ($t = 1$).

também o tempo dado como limite para os *teachers*. É importante observar que alguns *learners* têm mais linhas do que outros, porque as combinações $(\mathcal{D}, \mathcal{L})$ em que o limite de tempo é menor que 10 segundos são descartadas. Além disso, casos em que não há relevância estatística entre `Double` e `TCT` (com confiança estatística de 95%) são omitidos.

Nós destacamos em negrito os resultados em que há uma diferença estatística (com 95% de confiança) entre as acurácias de `Double` e `TCT`. Mais especificamente, destacamos em negrito `Double` (`TCT`) para o conjunto de dados $\mathcal{D}$ se a acurácia de `Double` (`TCT`) é maior do que a de `TCT` (`Double`) e

$$|acc_{Dbl} - acc_{\texttt{TCT}}| - 1.645\sqrt{\frac{acc_{Dbl}(1 - acc_{Dbl})}{m_{test}} + \frac{acc_{\texttt{TCT}}(1 - acc_{\texttt{TCT}})}{m_{test}}} > 0,$$

em que m_{test} é o tamanho do conjunto de teste para o *dataset* $\mathcal{D}$. Para calcular o intervalo de confiança, assumimos que os exemplos do conjunto de teste são extraídos independentemente de uma distribuição desconhecida μ (Capítulo 5 de (MITCHELL, 1997)). Nós não consideramos `OSCT` nesta comparação, pois não é competitivo com os outros dois *teachers*.

Em <https://github.com/sfilhofreitas/TimeConstainedLearning/tree/main/experiments/results/graphics_by_dataset> podem ser encontradas imagens que mostram como as acurácias de `TCT` e `Double` evoluem ao longo

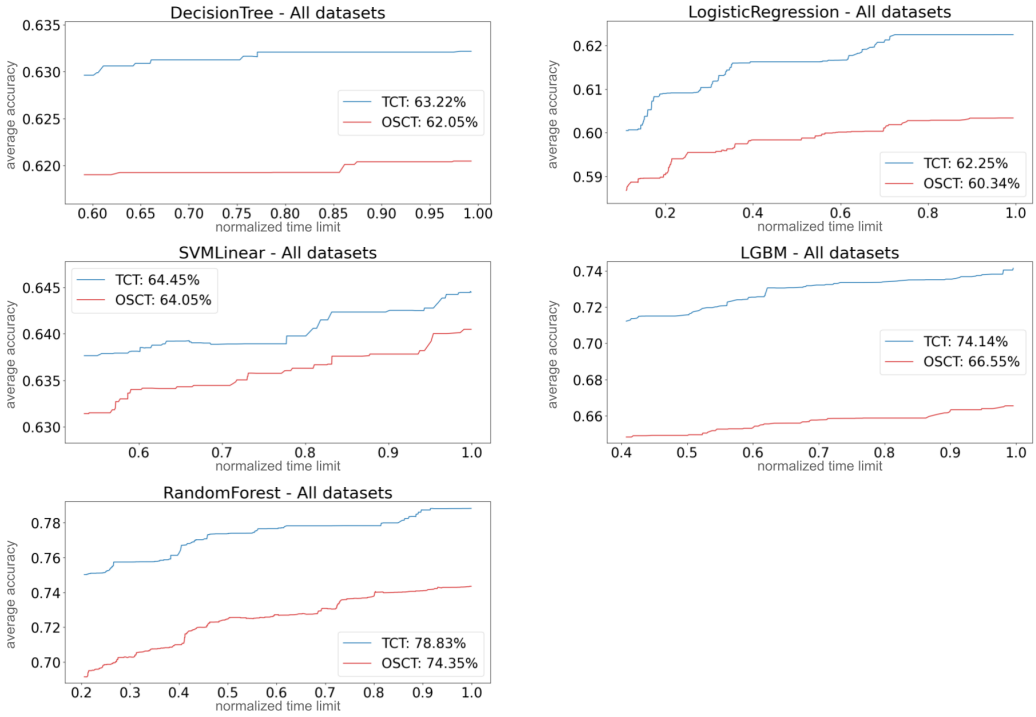

Figura H.2: Acurácias médias no conjunto de teste ao longo do tempo normalizado para `TCT` e `OSCT` que retorna o modelo com a melhor estimativa de acurácia. Os números próximos aos seus rótulos são suas acurácias no final do tempo limite normalizado ($t = 1$).

do tempo normalizado para cada *dataset*.

H.5 Selecting examples via active learning - additional tables

A Tabela H.2 mostra a acurácia média de `TCT` e $\texttt{TCT}_{AL}$ para os diferentes *learners* e *datasets*.

Tabela H.1: Accuracies in the testing sets for TCT, Double, OSCT and Full Training for each dataset.

Learner	Dataset	Tempo Limite	TCT	Double	OSCT	Full
LGBM	BNG_letter_5000_1	198.5	**75.9%**	74.8%	59.5%	76.7%
LGBM	SantanderCustomerSatisfaction	18.6	**91.3%**	90.6%	90.0%	90.7%
LGBM	BNG_wine	19.7	**96.1%**	95.8%	94.2%	96.0%
LGBM	BNG_eucalyptus	49.8	**74.9%**	73.9%	63.0%	74.3%
LGBM	mnist	134.5	**97.4%**	96.0%	92.5%	97.7%
LGBM	covtype	15.5	**86.6%**	85.3%	73.8%	85.3%
LGBM	BNG_satimage	77.2	**92.2%**	91.6%	87.2%	92.1%
LGBM	Sensorless_drive_diagnosis	11.7	**99.3%**	98.8%	93.8%	99.9%
LGBM	GTSRB-HueHist	258.0	38.9%	**39.9%**	33.0%	57.6%
LGBM	BNG_mfeat_fourier	214.6	**93.7%**	92.9%	86.4%	93.4%
Random Forest	BNG_letter_5000_1	231.9	**71.9%**	70.1%	68.5%	72.9%
Random Forest	poker_hand	277.5	**92.2%**	91.8%	92.3%	92.2%
Random Forest	BNG_spectf_test	781.3	**80.0%**	79.0%	78.0%	79.2%
Random Forest	MiniBooNE	59.7	**93.9%**	92.5%	92.6%	93.2%
Random Forest	BNG_eucalyptus	166.2	**69.2%**	67.5%	64.9%	69.1%
Random Forest	covtype	92.7	**93.0%**	88.1%	88.7%	92.4%
Random Forest	cifar_10	123.6	**42.9%**	42.1%	40.8%	45.4%
Random Forest	jannis	36.2	**69.6%**	68.5%	65.5%	70.0%
Random Forest	volkert	18.9	**62.4%**	60.6%	61.1%	64.4%
Random Forest	BNG_satimage	680.2	**89.7%**	88.2%	88.7%	89.0%
Random Forest	Sensorless_drive_diagnosis	13.7	**99.8%**	99.4%	99.7%	99.8%
Random Forest	GTSRB-HueHist	42.2	**47.8%**	43.8%	44.2%	47.3%
Random Forest	BNG_mfeat_fourier	1021.6	**88.9%**	88.4%	87.8%	88.9%
Random Forest	aloi	26.0	81.1%	**84.4%**	32.8%	90.6%
SVM	BNG_letter_5000_1	105.6	**42.8%**	41.3%	42.6%	41.3%
SVM	poker_hand	17.7	48.1%	**50.0%**	47.8%	50.0%
SVM	MiniBooNE	11.6	**90.1%**	89.7%	89.8%	90.1%
SVM	BNG_eucalyptus	80.3	**57.0%**	56.4%	54.5%	56.4%
SVM	mnist	1320.7	**90.4%**	89.0%	89.6%	91.7%
SVM	covtype	123.7	**70.8%**	70.4%	70.0%	70.5%
SVM	BNG_satimage	33.8	**81.6%**	80.7%	81.8%	80.7%
SVM	Sensorless_drive_diagnosis	156.7	**78.9%**	73.7%	89.1%	74.3%
SVM	BNG_mfeat_fourier	71.4	82.7%	**83.1%**	82.1%	83.2%
Decision Tree	SantanderCustomerSatisfaction	17.7	89.0%	**89.7**%	86.9%	89.9%
Decision Tree	BNG_spectf_test	30.0	77.6%	**78.5%**	77.4%	78.5%
Decision Tree	BNG_eucalyptus	12.5	53.8%	**54.2**%	52.7%	54.2%
Decision Tree	BNG_satimage	25.7	**73.8 %**	73.0%	74.0%	73.0%
Decision Tree	BNG_mfeat_fourier	47.5	**66.3%**	64.1%	64.0%	63.9%
Logistic Regression	poker_hand	391.5	48.1%	**50.0%**	47.7%	50.0%
Logistic Regression	covtype	84.0	**68.2%**	67.6%	62.5%	68.8%
Logistic Regression	Sensorless_drive_diagnosis	11.3	**83.9%**	70.0%	81.7%	78.8%
Logistic Regression	BNG_mfeat_fourier	46.1	84.4%	**84.6%**	82.8%	84.8%

Tabela H.2: Accuracies in the testing sets for `TCT` and `TCT_`AL.

Learner	**Dataset**	**Time Limit**	**TCT**	**TCT_**AL
Logistic Regression	BayesianNetworkGenerator_spambase	45.6	66.4%	66.6%
Logistic Regression	BNG_eucalyptus	48.3	57.8%	57.9%
Logistic Regression	BNG_letter_5000_1	45.1	45.6%	**46.0%**
Logistic Regression	BNG_mfeat_fourier	80.5	84.4%	84.5%
Logistic Regression	BNG_satimage	26.3	83.8%	83.8%
Logistic Regression	BNG_spectf_test	10.6	76.9%	**78.4%**
Logistic Regression	cifar_10	938.8	37.1%	37.5%
Logistic Regression	covtype	110.7	68.2%	67.9%
Logistic Regression	Diabetes130US	350.7	58.6%	58.8%
Logistic Regression	GTSRB-HueHist	237.7	26.3%	26.6%
Logistic Regression	jannis	12.9	63.7%	64.2%
Logistic Regression	mnist	266.1	91.6%	91.8%
Logistic Regression	poker_hand	526.5	48.1%	**52.1%**
Logistic Regression	Sensorless_drive_diagnosis	16.9	83.9%	**88.1%**
Logistic Regression	volkert	48.9	57.4%	57.4%
Random Forest	aloi	25.1	**81.1%**	68.7%
Random Forest	BayesianNetworkGenerator_spambase	517.3	66.6%	66.7%
Random Forest	BNG_eucalyptus	194.2	**69.2%**	68.0%
Random Forest	BNG_letter_5000_1	264.6	**71.9%**	70.8%
Random Forest	BNG_mfeat_fourier	889.9	**88.8%**	88.5%
Random Forest	BNG_satimage	864.6	**89.3%**	89.1%
Random Forest	BNG_spectf_test	829.8	**80.0%**	79.3%
Random Forest	BNG_wine	289.5	**95.7%**	95.5%
Random Forest	cifar_10	133.8	42.9%	42.1%
Random Forest	covtype	63.8	**92.9%**	89.5%
Random Forest	Diabetes130US	69.4	58.5%	58.7%
Random Forest	GTSRB-HueHist	35.0	40.0%	39.8%
Random Forest	jannis	35.5	**69.6%**	68.8%
Random Forest	MiniBooNE	62.8	**93.9%**	93.2%
Random Forest	mnist	40.3	96.3%	96.4%
Random Forest	poker_hand	189.8	91.4%	**92.1%**
Random Forest	SantanderCustomerSatisfaction	460.7	**90.5%**	90.0%
Random Forest	Sensorless_drive_diagnosis	14.9	99.8%	99.6%
Random Forest	vehicle_sensIT	77.9	**86.9%**	86.7%
Random Forest	volkert	20.4	**62.4%**	61.3%
LGBM	aloi	982.4	12.6%	**13.4%**
LGBM	BNG_eucalyptus	57.6	**74.9%**	74.2%
LGBM	BNG_letter_5000_1	162.4	**74.1%**	73.5%
LGBM	BNG_mfeat_fourier	279.5	**93.7%**	93.1%
LGBM	BNG_satimage	83.7	**92.2%**	91.5%
LGBM	BNG_spectf_test	18.5	82.6%	82.6%
LGBM	BNG_wine	20.7	95.9%	95.9%
LGBM	cifar_10	977.3	42.6%	42.7%
LGBM	covtype	18.1	**86.6%**	85.8%
LGBM	GTSRB-HueHist	270.7	34.0%	**36.8%**
LGBM	mnist	109.0	96.4%	**96.8%**
LGBM	poker_hand	54.7	73.0%	**77.9%**
LGBM	SantanderCustomerSatisfaction	23.0	**91.3%**	90.9%
LGBM	Sensorless_drive_diagnosis	14.1	**99.3%**	99.0%
LGBM	volkert	36.1	59.7%	59.1%
Decision Tree	BNG_eucalyptus	13.3	53.1%	**54.5%**
Decision Tree	BNG_mfeat_fourier	52.1	62.4%	**64.0%**
Decision Tree	BNG_satimage	22.1	**73.8%**	73.0%
Decision Tree	BNG_spectf_test	27.1	77.6%	**78.5%**
Decision Tree	cifar_10	34.0	24.9%	25.4%
Decision Tree	Diabetes130US	10.8	57.1%	57.1%
Decision Tree	poker_hand	10.6	51.3%	**53.5%**
Decision Tree	SantanderCustomerSatisfaction	18.4	89.0%	**89.8%**

I
Demonstrações da Seção 4.6

I.1
Enviar Muitos "Exemplos Incorretos" é Ruim

Construímos uma instância simples em que ilustramos um caso no qual o algoritmo `TCTbase` é definido com a porcentagem α de "amostras erradas" muito alta e, como resultado, tem uma acurácia muito baixa.

Mais precisamente, essa instância não realizável possui pontos $\mathcal{X} = \{1, 2, \ldots, 6\}$ e uma classe de hipóteses com 4 classificadores $\mathcal{H} = \{h_1, h_2, h_3, \bar{h}\}$ que classificam os pontos como $+1$ da seguinte forma (os pontos restantes são classificados como -1):

$$\begin{aligned} h_1 &: \quad \{1, 2, 3, 4\} \\ h_2 &: \quad \{1, 2, 5, 6\} \\ h_3 &: \quad \{3, 4, 5, 6\} \\ \bar{h} &: \quad \{1, 2, 3, 4, 5, 6\}. \end{aligned}$$

A classificação correta h^* classifica como $+1$ os pontos ímpares $\{1, 3, 5\}$. Finalmente, a distribuição μ define a probabilidade de $\frac{2}{9}$ para cada número ímpar e a probabilidade $\frac{1}{9}$ em cada número par de $\mathcal{X}$.

O melhor classificador em $\mathcal{H}$ é $\bar{h}$, que tem erro $\text{err}(\bar{h}) = \frac{1}{3}$, enquanto todos os outros classificadores têm erro $\text{err}(h_i) = \frac{4}{9}$.

Realizamos experimentos com `TCTbase` em que ele envia $\alpha = 90\%$ amostras "erradas" em cada rodada. Rodamos o algoritmo por 20 rodadas, então na última rodada ele tem um total de $2^{21} - 1 \approx 2.000.000$ amostras (pode haver/há várias cópias da mesma amostra $(x, h^*(x))$). Em mais de 100 tentativas, este algoritmo só encontrou o melhor classificador $\bar{h}$ (em qualquer uma de suas rodadas) 8% das vezes. Em contraste, `TBatch` com 1.000 exemplos encontrou o melhor classificador 100% da vez.

Observe que na última rodada `TCTbase` tem $(1 - \alpha) \cdot (2^{21} - 1) \approx 200.000$ amostras imparciais que é, em ordem de magnitude, um número bem maior que a quantidade usada por `TBatch`, mas ainda assim as "amostras erradas" causaram um desempenho muito ruim.

Observe também que quando α é grande como neste exemplo, o limite do Teorema I.1 é vago, devido ao último termo do erro.

I.2
O Caso Agnóstico

Considerando o caso agnóstico, seja $\varepsilon_T^A = \varepsilon_T^A(\mathcal{H}, \mu, \delta)$ tal que `TBatch` com limite de tempo T retorna uma hipótese com erro real no máximo igual ao do melhor classificador em $\mathcal{H}$ mais ε_T^A com probabilidade pelo menos $1 - \delta$ independentemente do *learner* ERM; mais precisamente, seja S um conjunto de m_T amostras aleatórias de μ e deixe ε_T^A ser o menor valor tal que

$$\Pr\left(\text{argmax}_{h\in\mathcal{H}}|\text{err}(h) - \text{err}_S(h)| > \frac{\varepsilon_T^A}{2}\right) < \delta.$$

Temos, então, a seguinte garantia.

Teorema I.1 *Dado $\delta \in (0,1)$ e um limite de tempo T, seja ε_T^A o erro adicional com probabilidade $(1-\delta)$ do* `TBatch` *conforme definido acima. Sob as hipóteses (i)-(iv), no cenário agnóstico, com probabilidade pelo menos $1-\delta$, o* `TCTbase` *retorna em tempo de no máximo $T \cdot 2(\frac{2}{1-\alpha})^{k+1}$ um classificador h com erro adicional de no máximo $\varepsilon_T^A + \frac{\alpha}{1-\alpha}$, ou seja,*

$$\text{err}(h) \leq \min_{h'\in\mathcal{H}} \text{err}(h') \ + \ \varepsilon_T^A + \frac{\alpha}{1-\alpha}.$$

Prova. Novamente, seja m_T o número de amostras enviadas por `TBatch` quando o limite de tempo for T. Além disso, seja $\hat{i}$ a primeira rodada em que `TCTbase` envia pelo menos $\frac{1}{1-\alpha}m_T$ amostras, ou seja, $\frac{1}{1-\ alpha}m_T \leq 2^{\hat{i}} \leq \frac{2}{1-\alpha}m_T$. Devido às hipóteses (ii) e (iv), a desigualdade (4-2) mostra que `TCTbase` termina a rodada $\hat{i}$ no tempo $2(\frac{2}{1-\alpha})^{k+1}T$.

Seja S o conjunto de amostras enviadas por `TCTbase` ao *learner* ao final da rodada $\hat{i}$, e seja $\hat{h}$ a hipótese retornada. Seja também U o subconjunto das amostras S que foram amostradas sem viés de μ, e $W = S \setminus U$ as restantes.

Como U e W constituem uma fração $(1-\alpha)$ e α de S, respectivamente, temos

$$\text{err}_S(\cdot) = (1-\alpha)\,\text{err}_U(\cdot) + \alpha\,\text{err}_W(\cdot). \tag{I-1}$$

Além disso, por definição de $\hat{i}$ temos $|U| \geq (1-\alpha)|S| \geq m_T$, e assim, usando a definição de ε_T^A, temos que com probabilidade de pelo menos $1-\delta$, para todo $h \in \mathcal{H}$

$$|\text{err}(h) - \text{err}_U(h)| \leq \frac{\varepsilon_T^A}{2};$$

em particular, à luz de (I-1)], para todo $h \in \mathcal{H}$

$$\mathrm{err}_S(h) \leq (1-\alpha)\left[\mathrm{err}(h) + \frac{\varepsilon_T^A}{2}\right] + \alpha \qquad \text{e} \qquad \mathrm{err}_S(h) \geq (1-\alpha)\left[\mathrm{err}(h) - \frac{\varepsilon_T^A}{2}\right].$$

Sob esse evento, o classificador $\hat{h}$ retornado pelo *learner* ERM satisfaz o seguinte limite para cada $h \in \mathcal{H}$:

$$(1-\alpha)\,\mathrm{err}(\hat{h}) \leq \mathrm{err}_S(\hat{h}) + (1-\alpha)\frac{\varepsilon_T^A}{2} \leq \mathrm{err}_S(h) + (1-\alpha)\frac{\varepsilon_T^A}{2} \leq (1-\alpha)\mathrm{err}(h) + (1-\alpha)\varepsilon_T^A + \alpha,$$

e assim tomando um mínimo sobre $h \in \mathcal{H}$ obtemos

$$err(\hat{h}) \leq \min_{h' \in \mathcal{H}} \mathrm{err}(h') + \varepsilon_T^A + \frac{\alpha}{1-\alpha}.$$

Isso conclui a prova. ■

I.3
Prova da Proposição 4.6.2

Para provar esse resultado, precisaremos usar martingales. Lembre-se de que uma sequência de variáveis aleatórias $X_1, \ldots, X_n$ é uma *sequência de diferenças de martingale* se $\mathbb{E}[X_i \mid X_1, \ldots, X_{i-1}] = 0$ para todo i. Precisamos da clássica desigualdade de Freedman para martingales.

Teorema I.2 (Theorem 1.6 of (FREEDMAN, 1975)) *Considere uma sequência de diferenças martingale $X_1, \ldots, X_n$ tal que $X_i \leq 1$, e sua variação quadrática previsível $V := \sum_i \mathbb{E}[X_i^2 \mid X_1, \ldots, X_{i-1}]$. Então, para qualquer $\lambda \geq 0$ e $v > 0$,*

$$\Pr\left(\sum_{i \leq n} X_i \geq \lambda \;\; e \;\; V \leq v\right) \leq \left(\frac{v}{\lambda + v}\right)^{\lambda+v} e^{\lambda}.$$

Prova.[Prova da Proposição 4.6.2] Defina $\widetilde{B}_i := B_i - \mathbb{E}[B_i \mid B_1, \ldots, B_{i-1}]$, de modo que a sequência $\widetilde{B}_{2R/3}, \ldots, \widetilde{B}_R$ é uma sequência de diferenças de martingale. Além disso, como B_i só assume o valor 0 ou 1, temos $|\widetilde{B}_i| \leq 1$, e assim

$$(\widetilde{B}_i)^2 \;\leq\; |\widetilde{B}_i| \;\leq\; B_i + \mathbb{E}[B_i \mid B_1, \ldots, B_{i-1}].$$

Da Proposição 4.6.1, por $i \geq \frac{2R}{3}$ temos

$$\mathbb{E}[B_i \mid B_1, \ldots, B_{i-1}] \leq e^{-2^{3i/4}} \leq e^{-2^{R/2}}$$
$$\text{e assim} \;\; \mathbb{E}\big[\widetilde{B}_i^2 \mid \widetilde{B}_{2R/3}, \ldots, \widetilde{B}_{i-1}\big] \leq 2e^{-2^{R/2}}.$$

Então $v := 2 \cdot \frac{2R}{3} \cdot e^{-2^{R/2}}$ é um limite superior para ambos os deslocamentos introduzidos em $\widetilde{B}_i$ e a variação quadrática previsível com probabilidade 1:

$$\begin{aligned} &\textstyle\sum_{i=2R/3}^{R} \mathbb{E}[B_i \mid B_1, \ldots, B_{i-1}] \leq v \\ \text{e} \quad &\textstyle\sum_{i=2R/3}^{R} \mathbb{E}\big[\widetilde{B}_i^2 \mid \widetilde{B}_{2R/3}, \ldots, \widetilde{B}_{i-1}] \leq v. \end{aligned}$$

Então, a desigualdade de Freedman dá

$$\begin{aligned} \Pr\left(\sum_{i=2R/3}^{R} B_i \geq \frac{R}{6}\right) \leq \Pr\left(\sum_{i=2R/3}^{R} \widetilde{B}_i \geq \frac{R}{6} - v\right) \leq \left(\frac{v}{R/6}\right)^{R/6} e^{R/6} &\leq (8e \cdot e^{-2^{R/2}})^{R/6} \\ &\leq e^{-2^{R/2}}, \end{aligned}$$

em que a última desigualdade usa o fato de que $R \geq 10$ (definindo *cst* grande o suficiente). Como $R \geq \log\log\frac{1}{\delta} + \sqrt{\log\frac{1}{\varepsilon}}$ e por hipótese $\varepsilon \leq \delta$, temos $e^{-2^{R/2}} \leq e^{-2^{\log\log 1/\delta}} \leq \delta$. Isso prova a proposição. ■