Pontifícia Universidade Católica
do Rio de Janeiro

Pedro Lazéra Cardoso

Machine Teaching com Tempo Limitado para Problemas de Regressão

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática da PUC–Rio.

Orientador: Prof. Eduardo Sany Laber

Rio de Janeiro
Setembro 2021

Pontifícia Universidade Católica
do Rio de Janeiro

Pedro Lazéra Cardoso

Machine Teaching com Tempo Limitado para Problemas de Regressão

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

Prof. Eduardo Sany Laber
Orientador
Pontifícia Universidade Católica do Rio de Janeiro

Prof. Marco Serpa Molinaro
Pontifícia Universidade Católica do Rio de Janeiro

Prof. Hélio Côrtes Vieira Lopes
Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 28 de setembro de 2021

Pedro Lazéra Cardoso

Graduado em Engenharia de Produção pela Pontifícia Universidade Católica do Rio de Janeiro (2013).

Ficha Catalográfica

Cardoso, Pedro Lazéra

Machine Teaching com Tempo Limitado para Problemas de Regressão / Pedro Lazéra Cardoso; orientador: Eduardo Sany Laber. — Rio de Janeiro : PUC–Rio, Departamento de Informática, 2021.

v., 57 f: il. ; 29,7 cm

1. Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui referências bibliográficas.

1. Informática – Dissertação. 2. aprendizado de maquina;. 3. inteligência artificial;. 4. problemas de regressão;. 5. machine teaching.. I. Laber, Eduardo Sany. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Agradecimentos

Ao meu orientador, Eduardo Laber, pela ajuda inestimável e pelas sugestões precisas, sempre mantendo o equilíbrio entre guiar e dar liberdade para o aluno seguir os próprios caminhos. Ao Sérgio, pelo companheirismo durante a pesquisa, em especial pela paciência em rodar os diversos experimentos sem os quais este trabalho não seria possível.

Ao Georges e ao João Guilherme, amigos da vida e do mestrado, e à Clarice, minha namorada, pela parceria em todos os momentos.

Por último, aos meus pais, Alexandre e Eliane, e aos meus irmãos, Juliana e Felipe, pelo apoio incondicional.

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

Resumo

Cardoso, Pedro Lazéra; Laber, Eduardo Sany. **Machine Teaching com Tempo Limitado para Problemas de Regressão**. Rio de Janeiro, 2021. 57p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho considera o problema de Regressão com Tempo Limitado. Dados um *dataset*, um algoritmo de aprendizado (*Learner*) a ser treinado e um tempo limitado, não sabemos se será possível treinar o modelo com todo o *dataset* dentro deste tempo. Queremos então elaborar a estratégia que extraia o melhor modelo possível deste algoritmo de aprendizado respeitando o limite de tempo. Uma estratégia consiste em interagir com o *Learner* de duas formas: enviando exemplos para o *Learner* treinar e enviando exemplos para o *Learner* rotular. Nós definimos o que é o problema de Regressão com Tempo Limitado, decompomos o problema de elaborar uma estratégia em subproblemas mais simples e bem definidos, elaboramos uma estratégia natural baseada em escolha aleatória de exemplos e finalmente apresentamos uma estratégia, T_{W+BH}, que supera a estratégia natural em experimentos que realizamos com diversos *datasets* reais.

Palavras–chave

aprendizado de maquina; inteligência artificial; problemas de regressão; machine teaching.

Abstract

Cardoso, Pedro Lazéra; Laber, Eduardo Sany. **Limited Time Machine Teaching for Regression Problems**. Rio de Janeiro, 2021. 57p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

This work considers the Time-Limited Regression problem. Given a dataset, a learning algorithm (*Learner*) to be trained and a limited time, we do not know if it's going to be possible to train the model with the entire dataset within this time constraint. We then want to elaborate the strategy that extracts the best possible model from this learning algorithm respecting the time limit. A strategy consists of a series of interactions with the *Learner*, in two possible ways: sending labeled examples for the *Learner* to train and sending unlabeled examples for the *Learner* to classify. We define what the Time-Limited Regression problem is, we decompose the problem of elaborating a strategy into simpler and more well-defined sub-problems, we elaborate a natural strategy based on random choice of examples and finally we present a strategy, T_{W+BH}, that performs better than the natural strategy in experiments we have done with several real datasets.

Keywords

machine learning; artificial intelligence; regression problems; machine teaching.

Sumário

Lista de Figuras

Lista de Tabelas

1
Introdução

Imagine que queremos treinar uma Árvore de Regressão para rotular posições[1] de tabuleiros de xadrez. Nesse cenário hipotético, nosso *dataset* consiste em mais de um bilhão de partidas jogadas online[2] (das quais extrairemos mais de 50 bilhões de posições, ou exemplos de treinamento) e nosso tempo está restrito a apenas um dia. Apesar de conhecermos um pouco da teoria das Árvores de Regressão, estamos usando uma biblioteca escrita por terceiros e portanto não conhecemos o algoritmo para construção da Árvore ou sua implementação. Considerando a possibilidade de não ser possível rodar o algoritmo em todo o *dataset* em apenas um dia, como devemos proceder?

Uma abordagem possível é estimar o tamanho da maior subamostra (subconjunto de exemplos) que o algoritmo é capaz de treinar e enviar essa exata quantidade (ou uma quantidade um pouco menor, por segurança) de exemplos para serem treinados. Fica evidente que essa linha não é promissora, uma vez que o tempo total de execução depende do *dataset*, da implementação do algoritmo de Árvores de Regressão e da configuração da máquina que vai rodar os experimentos. Não bastasse, é possível que o algoritmo tenha um componente aleatório que vai impactar este tempo de execução.

Uma outra abordagem é escolher uma subamostra $S = S_0$ muito pequena (por exemplo, de tamanho $|S_0| = 1$) e repetir os passos abaixo, *enquanto houver tempo*:

- Treina com a subamostra S
- Aumenta o tamanho da subamostra S

Essa segunda abordagem resolve nossos problemas mais imediatos:

[1]Não é escopo deste trabalho detalhar como uma Árvore de Regressão avaliaria posições de xadrez. No entanto, convém dar alguma satisfação ao leitor mais curioso. Primeiro, uma posição de um tabuleiro de xadrez deve ser entendida como uma fotografia do tabuleiro (acompanhada da indicação de qual jogador tem a vez). Segundo, tipicamente a avaliação de uma posição de xadrez é associada um número real, cujo sinal indica de quem é a vantagem (positivo = "brancas vencendo", negativo = "pretas vencendo") e cuja magnitude (o valor em módulo) indica o tamanho da vantagem. De forma simplificada, uma avaliação $= -1$, por exemplo, indica que as pretas têm um peão de vantagem.

[2]Em 15/dez/2014, o maior sítio de xadrez online do mundo (`chess.com`) celebrou a bilionésima partida jogada. Esse número é certamente dezenas de vezes maior atualmente.

– O algoritmo de aprendizado é uma caixa-preta: não conhecemos a complexidade de tempo do algoritmo, sua implementação ou as configurações da máquina em que rodaremos nosso experimento

– O tempo é um recurso escasso: temos um limite de tempo que talvez impossibilite que rodemos o algoritmo em todo o *dataset*

– Queremos retornar algum modelo: queremos garantir que ao fim do experimento (dentro do limite de tempo), retornaremos *algum* modelo de regressão

A priori, em todos os momentos em que aumentamos nossa subamostra – na escolha da subamostra inicial e a cada vez que ampliamos o tamanho da subamostra –, escolhemos exemplos do *dataset* de forma aleatória[3].

A principal pergunta que este trabalho pretende responder é exatamente essa: dado um algoritmo de aprendizado (cujo funcionamento não conhecemos), um *dataset* e um tempo limitado para treinamento, podemos fazer melhor do que escolher subamostras de forma aleatória e uniforme?

1.1 Definição do Problema

1.1.1 Notação

Antes de enunciar formalmente o problema, vamos definir alguns termos.

– Um exemplo é um vetor x de números reais

– $\mathcal{X}$ é um conjunto de exemplos

– $\mathcal{Y}$ é um conjunto de rótulos possíveis para um exemplo $x \in \mathcal{X}$. Em particular, como estamos tratando de problemas de regressão, assumimos que $\mathcal{Y}$ é um intervalo de $\mathbb{R}$.

– Uma hipótese h é uma função que associa a cada exemplo $x \in \mathcal{X}$ um rótulo $h(x) \in \mathcal{Y}$

– Um *dataset* é um par (X, Y), onde $X = [x_1, \cdots, x_m]$ é um vetor de exemplos com $x_i \in \mathcal{X}$ e $Y = [y_1, \cdots, y_m]$ é um vetor de rótulos, com $y_i \in \mathcal{Y}$

– Um *learner* L é um algoritmo de aprendizado que recebe um *dataset* (X, Y) e retorna uma hipótese h

[3]Abuso de notação: de forma aleatória uniforme, ou seja, cada exemplo tem a mesma chance de ser escolhido

– A *classe de hipóteses* de um *learner* L é o conjunto de hipóteses que L é capaz de produzir

Além disso, duas hipóteses h_1 e h_2 são distintas (dado um universo de exemplos D) se $h_1(x) \neq h_2(x)$ para algum $x \in D$

1.1.2 O problema

Dados um *learner* L, um *dataset* (X, Y) e um limite de tempo t_{lim}, qual é a *estratégia* que retorna a melhor hipótese possível (entre as hipóteses que L pode produzir)?

Uma estratégia consiste numa série de interações com L, ao fim das quais um modelo de aprendizado treinado (uma hipótese) deve ser retornado. Cada interação pode ser de dois tipos: (i) enviar a L um subconjunto de exemplos rotulados para treinar; (ii) enviar a L um subconjunto de exemplos não-rotulados para medir seu erro nessa amostra.

É importante frisar que a estratégia não tem acesso aos parâmetros que definem o modelo. Se L produzir Árvores de Decisão, por exemplo, isto equivale a dizer que a estratégia não tem acesso aos nós ou à altura de qualquer Árvore retornada por L. Em outras palavras, L é uma caixa-preta – tudo que sabemos sobre L é que, ao receber como *input* uma amostra (X', Y'), este retorna uma hipótese h.

1.1.3 O paradigma de Machine Teaching

Machine Teaching estuda a forma mais eficiente de fazer um *Learner* L aprender uma determinada hipótese [6]. Tradicionalmente, a eficiência é associada ao número de exemplos ou o número de iterações necessários para induzir L a aprender (ou a convergir para) uma hipótese, independente do tempo gasto para realizar a tarefa. Neste trabalho, tratamos do problema de *Machine Teaching* com tempo limitado: dado um trio (Learner, *dataset*, t_{lim}), talvez não saibamos se é possível treinar L com todo o *dataset* dentro de t_{lim}. Nesse contexto, a pergunta que queremos responder é a seguinte: qual é a melhor estratégia para treinar L dentro de um determinado tempo? Em resumo, para resolver o problema proposto na seção 1.1.2, trabalhamos com o paradigma de *Iterative Machine Teaching* ([13], [14]).

1.2 Contribuições

Este trabalho faz três contribuições: primeiro, define o problema de *Iterative Machine Teaching* com Tempo Limitado para problemas de Regressão. Apesar de a área de Machine Teaching contar com diversas contribuições sobre induzir *Learners* a convergirem para uma determinada hipótese, não achamos, entre os trabalhos relacionados, um que trate especificamente do tempo como recurso escasso e principal restrição. Segundo, propõe uma estratégia natural para resolver o problema, T_{double}, baseada na ideia do "doubling trick". Terceiro (e principal), apresenta uma nova estratégia, T_{W+BH}, baseada na ideia de dar mais peso a exemplos em que o modelo de aprendizado corrente erra.

Para evidenciar a melhoria que T_{W+BH} apresenta sobre T_{double}, realizamos um conjunto robusto de experimentos, com diversos modelos de aprendizados e *datasets* reais.

1.3 Organização da Dissertação

Esta dissertação está organizada da seguinte forma: o capítulo 2 trata dos trabalhos relacionados ao tema, em especial os trabalhos no campo de *Machine Teaching*; o capítulo 3, cerne da dissertação, apresenta as soluções (ou estratégias) desenvolvidas para o problema; o capítulo 4 exibe diversos experimentos para verificar a qualidade das soluções propostas e traz evidências de que nossa estratégia T_{W+BH} tem desempenho superior à estratégia que seleciona exemplos de forma aleatória uniforme; finalmente, o capítulo 5 apresenta conclusões, limitações e possíveis caminhos que este trabalho pode seguir.

2
Trabalhos Relacionados

Em [8], o autor categoriza a literatura sobre *Machine Teaching* por meio duas dimensões: se o *Learner* sempre formula hipóteses consistentes com todos os exemplos do conjunto de treinamento e o nível de informação que o *Teacher* tem sobre o *Learner*. Neste trabalho, não assumimos a consistência das hipóteses ou qualquer conhecimento do *Teacher* sobre o *Learner*.

Em [13], o paradigma de *Machine Teaching* é introduzido, junto com a ideia de um protocolo de comunicação entre o *Teacher* e o *Learner*, duas abordagens que adotamos neste trabalho. No artigo, no entanto, sempre se assume que T[1] tem algum conhecimento sobre L[2]. Em [14] o autor abandona essa restrição de conhecimento, mas é mais restritivo sobre as interações entre T e L: a cada interação, T deve enviar apenas mais um exemplo rotulado para L atualizar seu modelo de aprendizado.

Independente do que é assumido sobre a consistência do *Learner* ou o conhecimento do *Teacher*, a maioria dos estudos em Machine Teaching procura minimizar a quantidade de exemplos necessários para fazer o *Learner* convergir ([8], [6], [19]), provavelmente muito influenciados pelo trabalho de [10]. Em vez de minimizar o número de exemplos treinados, estamos interessados no melhor modelo que conseguimos fazer L retornar dentro de um tempo limitado.

Em [9] o autor aborda o objetivo de *Machine Teaching* de maneira mais geral, assumindo que o *Teacher* interage com o *Learner* para minizar o *Learning Cost*, que pode ser tanto o número de exemplos utilizados quanto o tempo gasto no processo. Além disso, propõe uma estratégia de seleção de exemplos que escolhe os exemplos em ordem decrescente de erro. Neste trabalho, propomos parametrizar a importância que se dá a escolher exemplos com maior erro de previsão, uma vez que essa abordagem faz com que o conjunto de treino selecionado tenha sua distribuição diferente da distribuição real dos dados.

Stochastic Gradient Descent (SGD) tem sido apontada como uma solução *promissora* para problemas de aprendizado de máquina supervisionado em larga escala — isto é, problemas em que não há tempo para treinar com todos

[1]Lembrete: T é sinônimo de *Teacher* e de estratégia
[2]Lembrete: L é sinônimo de *Learner* e de algoritmo de aprendizado

os exemplos rotulados disponíveis [2]. No contexto geral em que a técnica é aplicada [1], se assume que L é um algoritmo *online* paramétrico, definido por um vetor de parâmetros w, ao qual é possível associar uma função $J(w)$ que calcula a perda (erro esperado, erro de generalização) do modelo definido por w. Além disso, essa função deve ser diferenciável, a não ser por um conjunto de pontos com probabilidade zero de ocorrer. Aqui, por "solução promissora" queremos dizer "solução que faz w convergir rapidamente para um um valor ótimo ou próximo do ótimo". Sob essas premissas (algoritmo de aprendizado online e paramétrico, função de perda diferenciável), podemos enxergar SGD como uma estratégia de seleção de exemplos. No entanto, as estratégias que propomos nessa dissertação são mais gerais, pois fazem menos exigências sobre L. Para L = Árvore de Regressão, um dos algoritmos utilizados na parte experimental desse trabalho, não é óbvio como poderíamos usar a técnica de SGD.

O paradigma de regressão com tempo limitado também tem relação com o paradigma de *Active Learning*. Em *Active Learning*, exemplos não rotulados são abundantes, ao passo que obter exemplos rotulados é custoso. O próprio *learner* L influencia que exemplos serão rotulados. A primeira questão é portanto como elaborar uma estratégia para determinar que exemplos rotular [7] — exemplos esses que serão então usados para treinar L. Temos também um problema de seleção de exemplos. A diferença essencial entre os paradigmas é que, ao simular o contexto de *Active Learning* num problema de regressão com tempo limitado, desprezamos a informação do rótulo correto de cada exemplo na fase de seleção de exemplos. Estamos abrindo mão de uma informação provavelmente e intuitivamente valiosa. Em [12], o autor sugere usar modelos distintos para as fases de selecionar exemplos e de treinar exemplos, para tratar do caso em que usar o modelo principal (o de treino) na fase de seleção de exemplos pode também ser custoso demais. Uma das estratégias propostas nesse artigo, T_{UPF}, é baseada no paradigma de *Active Learning*.

Muitos trabalhos demonstram resultados teóricos sem aplicação prática imediata ao problema tratado nesta dissertação. Ainda que se considere este distanciamento, estes resultados são inspiradores ao confirmarem - mesmo que num contexto simplificado, em que o tempo de processamento não é levado em conta ou em que T tem total conhecimento e ingerência sobre os parâmetros de L - o que tomamos a liberdade de chamar de intuição ou de bom senso. Como exemplo, descrevemos abaixo [5], em que é demonstrado que, quando o objetivo é fazer L convergir mais rápido (treinando com menos exemplos) para uma hipótese h^*, T se beneficia ao interagir (testar L ao longo do processo) com L.

2.1 Understanding the Role of Adaptivity in Machine Teaching: The Case of Version Space Learners

No trabalho, um *Version Space* induzido por uma amostra (subconjunto de exemplos de um *dataset*) Z é o conjunto de hipóteses consistentes com Z. Quando L recebe Z para treinar, L retorna, entre as hipóteses em $VersionSpace(Z)$, uma de suas hipóteses preferidas (não necessariamente de forma aleatória).

A preferência (de L entre duas hipóteses h_1, h_2) é definida por uma função de afinidade $OMEGA : H \times H \rightarrow \mathcal{R}$. A função recebe um par de hipóteses porque a afinidade de L com uma hipótese h depende de que hipótese h_{curr} está atualmente configurada em L. Em outras palavras, dada uma hipótese corrente h_{curr} e duas hipóteses h_1, h_2, temos que L prefere h_1 a h_2 se $OMEGA(h_{curr}, h_1) > OMEGA(h_{curr}, h_2)$.

Um *teacher* T é uma entidade (uma estratégia) que tenta conduzir L a uma hipótese-alvo h^*. Para realizar essa condução, a cada instante t a estratégia T seleciona um exemplo z_t e o envia a L. O ensino termina quando L retorna h^*.

T é (por definição) adaptativo se considerar h_{curr} para escolher z_t, e T é não-adaptativo se ignorar isso.

O trabalho assume que T conhece a função de preferência de L, $OMEGA : H \times H \rightarrow \mathbb{R}$.

Por fim, é demonstrado que para algumas funções de preferência um *teacher* adaptativo é melhor (precisa de menos iterações ou de menos exemplos) do que um *teacher* não-adaptativo. Uma outra maneira de dizer isso é que, em alguns casos, podemos fazer melhor do que escolher exemplos de forma aleatória para fazer L aprender.

3
Estratégias Propostas

O cerne desse trabalho é elaborar estratégias de seleção de exemplos (de um conjunto dado de exemplos, possivelmente grande demais para ser treinado dentro de um limite de tempo) para treinar modelos de regressão. Uma exigência a respeito da estratégia é que esta não dependa de um algoritmo de aprendizado específico, uma vez que o algoritmo será entendido como uma entidade que recebe um subconjunto de exemplos rotulados e retorna um modelo treinado. Em outras palavras, uma das exigências é que a estratégia trate L como uma caixa-preta[1].

Todas as estratégias propostas neste trabalho podem ser resumidas, em alto nível, pelas etapas listadas abaixo:

1. Faz um pré-processamento do *dataset*
2. Escolhe uma amostra inicial
3. Treina L com esta amostra inicial
4. Enquanto houver tempo, repete:
   (a) avalia o modelo corrente (não há treinamento nesta etapa)
   (b) seleciona um novo conjunto de exemplos para treino com base nos resultados da avaliação
   (c) treina L com o novo conjunto de exemplos

Neste esquema, "avaliar" equivale a avaliar a hipótese corrente (o modelo treinado corrente) de L numa amostra do *dataset*.

3.1
Um pseudocódigo para resumir todas as estratégias

Este esquema, em pseudocódigo um pouco menos abstrato, está detalhado abaixo. O significado do *input* segue aquilo definido na seção 1.1.1: T é

[1] Um *black box learner*

uma estratégia de seleção de exemplos (ou *Teacher*), L é um algoritmo de aprendizado, (X, Y) é um *dataset* e t_{lim} é o limite de tempo. Finalmente, a variável $m = |X| = |Y|$ representa o tamanho (a quantidade de exemplos) do *dataset*[2].

[2]Não é o tamanho da subamostra corrente, mas sim o tamanho do *dataset* inteiro

Algorithm 1 Pseudocódigo das estratégias

function TEACH($T, L, \mathbf{X}, \mathbf{Y}, t_{lim}$)
  $h_{selected} \leftarrow NULL$ ▷ model starts as NULL
  $(\mathbf{X}_{train}, \mathbf{Y}_{train}) \leftarrow (NULL, NULL)$ ▷ training sample, initialy empty

  T.START($\mathbf{X}, \mathbf{Y}$) ▷ The teacher can make some precomputations
  $(\mathbf{X}_{train}, \mathbf{Y}_{train}) \leftarrow T$.SELECT_FIRST_EXAMPLES($\mathbf{X}, \mathbf{Y}$) ▷ First batch
  $h_{curr} \leftarrow L$.FIT($\mathbf{X}_{train}, \mathbf{Y}_{train}$)

  while $t_{elapsed} \leq t_{lim}$ **do**
    $h_{selected} \leftarrow T$.CHOOSE_H($h_{selected}, h_{curr}$)
    $(\mathbf{X}_{eval}, \hat{\mathbf{Y}}_{eval}) \leftarrow$ RUN_EVALUATION($T, L, \mathbf{X}, \mathbf{Y}$)
    $(\mathbf{X'}_{train}, \mathbf{Y'}_{train}) \leftarrow T$.SELECT_MORE_EXAMPLES($\mathbf{X}, \mathbf{Y}, \mathbf{X}_{eval}, \hat{\mathbf{Y}}_{eval}$)
    $(\mathbf{X}_{train}, \mathbf{Y}_{train}) \leftarrow (\mathbf{X}_{train} \cup \mathbf{X'}_{train}, \mathbf{Y}_{train} \cup \mathbf{Y'}_{train})$
    L.FIT($\mathbf{X}_{train}, \mathbf{Y}_{train}$)
  end while

  return $h_{selected}$
end

function RUN_EVALUATIONS($T, L, \mathbf{X}, \mathbf{Y}$)
  $\mathbf{X}_{eval} \leftarrow NULL$ ▷ a list of unlabeled examples, initialy empty
  $\hat{\mathbf{Y}}_{eval} \leftarrow NULL$ ▷ the labels of $\mathbf{X}_{eval}$, according to h_{curr}
  while T.KEEP_EVALUATING( ) **do**
    $\mathbf{X}_{eval} \leftarrow \mathbf{X}_{eval} \cup T$.SELECT_EVALUATION_EXAMPLES($\mathbf{X}_{eval}, \hat{\mathbf{Y}}_{eval}$)
    $\hat{\mathbf{Y}}_{eval} \leftarrow L$.PREDICT($\mathbf{X}_{eval}$)
  end while

  return $(\mathbf{X}_{eval}, \hat{\mathbf{Y}}_{eval})$
end

As instruções $L.FIT$ e $L.PREDICT$ indicam respectivamente que o *Learner* irá treinar com um subconjunto de treino e rotular um subconjunto de exemplos (este último sem os rótulos verdadeiros de cada exemplo).

As variáveis em negrito (como $\mathbf{X}$, $\mathbf{Y}_{eval}$) representam coleções, como vetores ou conjuntos (estruturas de dados das quais podemos enumerar objetos).

As variáveis sobrescritas com o sinal "^" — o acento circunflexo, ou chapéu, como em $\hat{\mathbf{Y}}_{eval}$ — representam previsões feitas por algum modelo de aprendizado treinado.

Todas as variáveis globais (variáveis que não estão no *input* da função e que não são inicializadas dentro da função) são listadas no início do pseudocódigo[3].

Finalmente, expressões como

$$[\ \mathbf{X}[i]\ \textbf{for}\ i\ \textbf{in}\ \mathbf{S}\ ]$$

representam um vetor cujo j-ésimo elemento vale $X[\ S[j]\ ]$.

3.2 Uma estratégia em termos de funções

Definir as estratégia de seleção de exemplos abordadas neste trabalho consiste apenas em definir o comportamento (a implementação) de algumas funções:

- START: $(X, Y) \rightarrow NULL$
- SELECT_FIRST_EXAMPLES: $(X, Y) \rightarrow (X', Y')$
- SELECT_MORE_EXAMPLES: $(X_1, Y_1, X_2, Y_2) \rightarrow (X', Y')$
- KEEP_EVALUATING: $() \rightarrow \{true, false\}$
- SELECT_EVALUATION_EXAMPLES: $(X, Y) \rightarrow X'$
- CHOOSE_H: $(h_1, h_2) \rightarrow h$

Estas são todas as funções associadas à T no pseudocódigo geral da seção 3.1. Seguindo uma linha de Orientação a Objeto, são as funções com o prefixo "T."

A função $T.START(\mathbf{X}, \mathbf{Y})$ indica um pré-processamento da estratégia T sobre o *dataset* ($\mathbf{X}$, $\mathbf{Y}$).

A função $T.SELECT\_FIRST\_EXAMPLES(\mathbf{X}, \mathbf{Y})$ indica o primeiro subconjunto de exemplos selecionado pela estratégia T para treinar L. Nesse ponto, ainda não existe hipótese retornada por L (a variável $h_{selected}$ vale $NULL$).

[3]Esse caso só ocorre mais para frente

A função $T.SELECT\_MORE\_EXAMPLES(\mathbf{X}, \mathbf{Y}, \mathbf{X}_{eval}, \hat{\mathbf{Y}}_{eval})$ indica os subconjuntos subsequentes de exemplos (além do primeiro subconjunto) selecionado pela estratégia T para treinar L. Nesse ponto, já existe hipótese retornada por L, e portanto esses subconjuntos posteriores são escolhidos com base nos exemplos disponíveis $(\mathbf{X}, \mathbf{Y})$ e também numa avaliação da hipótese corrente sobre uma amostra do *dataset* $(\mathbf{X}_{eval}, \hat{\mathbf{Y}}_{eval})$.

As funções $T.SELECT\_EVALUATION\_EXAMPLES(\mathbf{X}_{eval}, \hat{\mathbf{Y}}_{eval})$ e $T.KEEP\_EVALUATING(\ )$ indicam a avaliação que T faz sobre a hipótese corrente. Nesse conjunto de interações entre T e L, a estratégia T seleciona exemplos apenas para L rotular (prever).

Finalmente, a função $CHOOSE\_H(h_{selected}, h_{curr})$ indica se T vai trocar ou não a hipótese a ser retornada ao fim do experimento.

3.3 Esclarecendo algumas premissas sobre o aprendizado

A respeito do pseudocódigo geral em 3.1, fazemos alguns esclarecimentos: primeiro, X_{train} e X_{eval} são subsequências de X; segundo, Y_{train} é o vetor de rótulos (verdadeiros)[4] dos exemplos em X_{train}; terceiro, $\hat{Y}_{eval}$ *não* é o vetor de rótulos corretos de X_{eval}, mas sim o vetor de rótulos previstos pelo modelo corrente; quarto, $h_{selected}$ e h_{curr} representam modelos já treinados[5], diferente de L, que é o algoritmo que retorna modelos treinados.

Um aspecto bem mais relevante está implícito no comando abaixo:

$$(\mathbf{X}_{train}, \mathbf{Y}_{train}) \leftarrow (\mathbf{X}_{train} \cup \mathbf{X'}_{train}, \mathbf{Y}_{train} \cup \mathbf{Y'}_{train})$$

Note que estamos assumindo que, a cada iteração, a estratégia *aumenta* a amostra de treino, ou seja, a nova amostra é a união entre a amostra antiga e alguns novos exemplos. A definição do problema não exige isso - as estratégias em geral podem, a cada rodada, selecionar um conjunto de exemplos completamente distinto do conjunto anterior. Sendo ainda mais puristas, a definição do problema sequer exige que existam rodadas de seleção de exemplo.

Finalmente, outro aspecto que segue do pressuposto acima e que deve ser ratificado está no comando

$$L.FIT(\mathbf{X}_{train}, \mathbf{Y}_{train})$$

[4] O gabarito de X_{train}

[5] O termo "modelo de aprendizado" é usado para denotar tanto um modelo treinado (por exemplo, uma árvore de classificação com nós definidos) quanto o algoritmo que monta modelos treinados (por exemplo, o algoritmo que recebe um conjunto de exemplos e retorna uma árvore treinada)

Esse linha implica que o algoritmo de aprendizado não é *online*, ou seja, o modelo corrente não é o modelo anterior modificado pelos novos exemplos selecionados na última rodada - o modelo corrente é o resultado do treinamento de (X_{train}, Y_{train}), e seus parâmetros não dependem de treinamentos anteriores.

Sobre essa premissa, é muito simples adaptar o pseudocódigo (e os códigos) para considerar algoritmos de aprendizado *online*. Basta alterar a instrução $L.FIT(\mathbf{X}_{train}, \mathbf{Y}_{train})$ para chamar $L.FIT$ apenas sobre o conjunto incremental (X'_{train}, Y'_{train}).

3.4 Um paralelo com o aprendizado humano

Para entender a motivação das estratégias propostas nesse trabalho, um paralelo pode ser feito com um professor que quer ensinar um aluno apenas por meio de exemplos.

Primeiro, antes de interagir (por meio de exemplos) com o aluno, o professor recebe a matéria (*dataset* ou conceito) que deve ser ensinada.

Dada uma matéria, o professor escolhe alguns exemplos rotulados e exibe ao aluno.

Em seguida, enquanto ainda restam dias de aula, o professor interage com o aluno da seguinte forma: (i) aplica uma série (de zero ou mais) testes para o aluno e (ii) com base no resultado do aluno nos testes aplicados, escolhe mais um pedaço da matéria (mais exemplos rotulados) para ensinar ao aluno.

Desviando um pouco do foco deste trabalho, é instigante pensar na tarefa de elaborar uma estratégia (um programa de computador) que um aluno (humano) não consiga diferenciar de um professor (humano também). Isto é, uma estratégia que escolhe exemplos de forma semelhante à forma que um professor humano escolheria[6].

3.5 A estratégia Single Batch

A estratégia talvez mais simples de todas é aquela que escolhe como subamostra inicial todo o *dataset*.

[6]Fica a cargo do leitor decidir se é assustador ou animador a semelhança entre o pseudocódigo de uma estratégia e o que um professor-humano de fato faria

Algorithm 2 $T_{SingleBatch}$

function START: **return** $NULL$ ▷ No preprocessing

function SELECT_MORE_EXAMPLES: **return** $NULL$

function KEEP_EVALUATING: **return** $FALSE$

function SELECT_EVALUATION_EXAMPLES: **return** $NULL$

function SELECT_FIRST_EXAMPLES($\mathbf{X}, \mathbf{Y}$)

return $(\mathbf{X}, \mathbf{Y})$ ▷ Select the entire dataset

end

function CHOOSE_H($h_{selected}, h_{curr}$)

return h_{curr}

end

A função $SELECT\_FIRST\_EXAMPLES$ retorna todo o *dataset*. Já as funções relativa a avaliações da hipótese corrente não fazem nenhuma avaliação. Finalmente, a função sobre trocar ou não trocar o valor de $h_{selected}$ sempre seleciona a última hipótese treinada.

O único (enorme) problema desta estratégia é também o motivo pelo qual essa dissertação está sendo escrita: existe a possibilidade de não ser possível treinar com todo o *dataset* dentro do limite de tempo, caso em que $h_{selected} = NULL$.

$T_{SingleBatch}$ irá funcionar como uma base de comparação para as demais estratégias. Com tempo disponível, nossa escolha natural seria treinar com todo o *dataset*. Não havendo tempo, queremos saber quão perto chegamos desse cenário ideal, e a medida de proximidade, a ser detalhada no capítulo 4, será a relação entre (i) o erro da hipótese retornada por $T_{SingleBatch}$ e (ii) o erro da hipótese retornada pela estratégia cuja qualidade queremos medir.

3.6 A estratégia Doubling Trick

Para remediar o problema de $T_{SingleBatch}$, a estratégia T_{double} (é como chamaremos "Doubling Trick") começa com uma subamostra inicial pequena[7] e vai dobrando seu tamanho enquanto enquanto houver tempo.

[7]Pequeno o suficiente para que L treine com uma subamostra deste tamanho dentro do limite de tempo

A fim de controlar o tamanho da amostra inicial, utilizamos o parâmetro $\text{frac}_{start} \in (0, 1]$, que representa a proporção de exemplos na amostra inicial. Caso o usuário não informe um valor para o parâmetro, assume-se que $\text{frac}_{start} = \frac{1}{m}$.

De forma muito abstrata, na primeira iteração T_{double} seleciona $m \cdot \text{frac}_{start}$ exemplos aleatoriamente. A cada próxima iteração $i + 1$, T_{double} seleciona aleatoriamente mais $|S_i|$ novos exemplos, onde S_i é o conjunto de exemplos treinados na iteração i.

O pseudocódigo abaixo representa de forma menos abstrata a estratégia T_{double}. Neste esquema, $\mathbf{v}_{free_{ids}}$ representa o conjunto de índices dos exemplos que ainda não foram selecionados para compor a amostra de treino.

Algorithm 3 T_{double}

function KEEP_EVALUATING: same as in $T_{SingleBatch}$

function SELECT_EVALUATION_EXAMPLES: same as in $T_{SingleBatch}$

function CHOOSE_H: same as in $T_{SingleBatch}$

global variables: $\text{size}_{next_{batch}}$, frac_{start}, $m = |X|$, $\mathbf{v}_{free_{ids}}$

function START($\mathbf{X}, \mathbf{Y}$)
  $\text{size}_{next_{batch}} \leftarrow \text{FLOOR}(\text{frac}_{start} \cdot m)$
  $\mathbf{v}_{free_{ids}} \leftarrow [1, \cdots, m]$
end

function SELECT_FIRST_EXAMPLES($\mathbf{X}, \mathbf{Y}$)
  $\mathbf{v}_{new_{ids}} \leftarrow \text{CHOOSE\_RANDOM}(\mathbf{v}_{free_{ids}}, \text{size}_{next_{batch}})$
  $\text{size}_{next_{batch}} \leftarrow \text{size}_{next_{batch}} \cdot 2$
  $\mathbf{v}_{free_{ids}} \leftarrow \mathbf{v}_{free_{ids}} - \mathbf{v}_{new_{ids}}$ ▷ This "−" represents a set difference

  $\mathbf{X}_{new} \leftarrow [\ \mathbf{X}[i]$ **for** i **in** $\mathbf{v}_{new_{ids}}\ ]$
  $\mathbf{Y}_{new} \leftarrow [\ \mathbf{Y}[i]$ **for** i **in** $\mathbf{v}_{new_{ids}}\ ]$

  return $(\mathbf{X}_{new}, \mathbf{Y}_{new})$
end

function SELECT_MORE_EXAMPLES($\mathbf{X}, \mathbf{Y}, \mathbf{X}_{eval}, \hat{\mathbf{Y}}_{eval}$)
  return SELECT_FIRST_EXAMPLES($\mathbf{X}, \mathbf{Y}$)
end

Fazemos alguns esclarecimentos para o leitor que mentalmente compila pseudocódigos[8]. Primeiro, o pseudocódigo de T_{double} não lida com casos de fronteira, como quando $size_{next\_batch} > |X|$. Segundo, a chamada de função $CHOOSE\_RANDOM(v, k)$, sobre um vetor v e um inteiro positivo $k \leq |v|$, faz um sorteio aleatório uniforme sem reposição[9] de k elementos de v.

[8]E que não consegue prosseguir a leitura enquanto o código não compilar sem erros em sua cabeça

[9]Nenhum elemento será escolhido duas vezes

Finalmente, destacamos que a escolha de novos exemplos é idêntica à escolha dos primeiros exemplos. Isto porque T_{double} não avalia a hipótese corrente para (tentar) fazer uma escolha "não puramente aleatória e uniforme" da nova batelada de exemplos. Em outras palavras, T_{double} é uma mera adaptação de $T_{SingleBatch}$ para a possibilidade de não haver tempo suficiente para treinar com o *dataset* inteiro, e esta adaptação é feita com o "doubling trick".

3.7 A estratégia Wrong Probably First (WPF)

A estratégia T_{WPF} é a nossa primeira tentativa de selecionar exemplos de forma diferente da aleatória uniforme. Se baseia no princípio[10] de que exemplos em que a hipótese corrente comete maior erro são mais úteis para treinar um novo modelo.

T_{WPF} deve ser entendida como uma extensão de T_{double}, com a única (enorme) diferença de que, a partir da segunda iteração, os exemplos a serem adicionados não são sorteados uniformemente. Seus parâmetros são os seguintes:

- $\text{frac}_{start} \in (0, 1]$
- $sample_{factor} \in [1, +\infty)$
- $w_{error} \in [0, 1]$

O parâmetro frac_{start} tem o mesmo significado explicado na seção 3.6. O parâmetro $sample_{factor}$ controla que fração dos exemplos livres (ainda não usados em treino) será avaliada para seleção dos novos exemplos (a cada iteração a partir da segunda). Já w_{error} controla a importância do erro (cometido pela hipótese corrente) de um exemplo para determinar o peso deste exemplo no sorteio de novos exemplos. O significado preciso destes parâmetros ficará mais claro, adiante, quando explicaremos toda a mecânica de T_{WPF}.

Na primeira iteração, T_{WPF} seleciona $m \cdot fracs_{start}$ exemplos de forma aleatória uniforme[11]. Ainda de forma semelhante a T_{double}, a cada próxima iteração $i + 1$, T_{WPF} seleciona mais $|S_i|$ novos exemplos, onde S_i é o conjunto de exemplos treinados na iteração i. Esta seleção, diferente do que ocorre em T_{double}, não é aleatória uniforme.

O sorteio (da batelada de exemplos a partir da segunda iteração) é feito em duas etapas. Primeiro, são sorteados uniformemente $|S_i| \cdot sample_{factor}$ exemplos, que formam a amostra A. A cada iteração que não a primeira, a amostra A muda.

[10] Isto não foi demonstrado e será verificado a partir de experimentos
[11] Exatamente como feito por T_{double}

Em seguida, para cada exemplo j da amostra A, calcula-se o erro de previsão e_j. Note que o $j-$ésimo exemplo de A não é o $j-$ésimo exemplo do *dataset*. Cada exemplo é associado a um peso w_j, com

$$w_j = w_{error} \cdot e_j^{'} + (1 - w_{error}) \cdot \frac{1}{|A|}$$

.

Aqui, $e_j^{'}$ é o erro de previsão normalizado, com $e_j^{'} = e_j / \sum_{k=1}^{|A|} e_k$. Essa transformação garante $\sum_{j=1}^{|A|} e_j^{'} = 1$, que por sua vez garante $\sum_{j=1}^{|A|} w_j = 1$

Finalmente, da amostra A são sorteados $|S_i|$ exemplos, de acordo com os pesos w_j.

Simplificadamente, cada exemplo da amostra A é associada um peso que é função de uma parcela uniforme ($1/|A|$) e de uma parcela relacionada ao erro de previsão. A influência de cada parcela é controlada pelo parâmetro w_{error}.

Algorithm 4 T_{WPF}

function START: same as in $T_{SingleBatch}$

function CHOOSE_H: same as in $T_{SingleBatch}$

function SELECT_FIRST_EXAMPLES: same as in T_{double}

global variables: $\text{size}_{next_{batch}}$, frac_{start}, $m = |X|$, $\mathbf{v}_{free_{ids}}$
more global variables: $sample_{factor}$, w_{error}, $\mathbf{v}_{eval_{ids}}$

function SELECT_MORE_EXAMPLES($\mathbf{X}, \mathbf{Y}, \mathbf{X}_{eval}, \hat{\mathbf{Y}}_{eval}$)
$k \leftarrow |\mathbf{v}_{eval_{ids}}|$ ▷ $\mathbf{v}_{eval_{ids}}$ is a global variable
$\mathbf{Y}_{true} \leftarrow$ [$\mathbf{Y}[i]$ **for** i **in** $\mathbf{v}_{eval_{ids}}$]
$\mathbf{e} \leftarrow$ [$(\hat{\mathbf{Y}}_{eval}[j] - \mathbf{Y}_{true}[j])^2$ **for** j **in** $1 \cdots k$]
e' $\leftarrow$ [$\mathbf{e}[j]/\text{SUM}(\mathbf{e})$ **for** j **in** $1 \cdots k$]
$\mathbf{w} \leftarrow$ [**e'**$[j] \cdot w_{error} + 1/k \cdot (1 - w_{error})$ **for** j **in** $1 \cdots k$]
$\mathbf{v}_{new_{ids}} \leftarrow$ CHOOSE_RANDOM($\mathbf{v}_{eval_{ids}}$, $\text{size}_{next_{batch}}$, weights $= \mathbf{w}$)

$\text{size}_{next_{batch}} \leftarrow \text{size}_{next_{batch}} \cdot 2$
$\mathbf{v}_{free_{ids}} \leftarrow \mathbf{v}_{free_{ids}} - \mathbf{v}_{new_{ids}}$ ▷ This "−" represents a set difference

$\mathbf{X}_{new} \leftarrow$ [$\mathbf{X}[i]$ **for** i **in** $\mathbf{v}_{new_{ids}}$]
$\mathbf{Y}_{new} \leftarrow$ [$\mathbf{Y}[i]$ **for** i **in** $\mathbf{v}_{new_{ids}}$]
return $(\mathbf{X}_{new}, \mathbf{Y}_{new})$
end

function KEEP_EVALUATING
if (já avaliou hipótese corrente) **then**
return $FALSE$
else
return $TRUE$
end if
end

function SELECT_EVALUATION_EXAMPLES($\mathbf{X}, \mathbf{Y}$)
$sample_{factor} \leftarrow \text{size}_{next_{batch}} \cdot sample_{factor}$
$\mathbf{v}_{eval_{ids}} \leftarrow$ CHOOSE_RANDOM($\mathbf{v}_{free_{ids}}$, $sample_{factor}$)
return [$\mathbf{X}[i]$ **for** i **in** $\mathbf{v}_{eval_{ids}}$]
end

Assim como nas demais ocasiões, esse pseudocódigo não trata de casos de fronteira (degenerados). Em particular, se SUM($\mathbf{e}$) = 0, teríamos uma divisão por ZERO na execução da função SELECT_MORE_EXAMPLES. A propósito, SUM($\mathbf{e}$) denota a soma dos elementos do vetor $\mathbf{e}$.

A chamada de função

$$\text{CHOOSE\_RANDOM}(\mathbf{v}_{eval_{ids}}, \text{size}_{next_{batch}}, \text{weights} = \mathbf{w})$$

é diferente das chamadas feitas anteriormente, em T_{double}. Aqui, são sorteados $\text{size}_{next_{batch}}$ elementos de $\mathbf{v}_{eval_{ids}}$, sem reposição, mas este sorteio não é uniforme: os pesos de cada elemento são dados pelo vetor $\mathbf{w}$. Um detalhe de implementação: na verdade, sorteamos os índices – ou ponteiros – dos exemplos em $(\mathbf{X}, \mathbf{Y})$.

Também é importante reforçar outra diferença em relação à T_{double}: esta etapa do sorteio não é feita a partir de todos os exemplos livres (ainda não usados em treino), mas sim dos exemplos para os quais a hipótese corrente foi avaliada. Isto porque este é um segundo sorteio – o primeiro deles foi feito na seleção de exemplos para teste, em SELECT_EVALUATION_EXAMPLES($\mathbf{X}, \mathbf{Y}$).

A instrução

$$(\hat{\mathbf{Y}}_{eval}[j] - \mathbf{Y}_{true}[j])^2$$

assume que a função de perda é o erro quadrático médio, de modo que a contribuição de cada exemplo é seu erro quadrático. Para adotar outra função de perda – como a diferença em módulo –, bastaria adaptar essa linha de código.

Por último, é instrutivo enxergar o comportamento de T_{WPF} quando estressamos os parâmetros w_{error} e $sample_{factor}$.

Note que se $w_{error} = 0$, a seleção de exemplos ignora os erros e T_{WPF} se torna T_{double}. O parâmetro w_{error} pode ser visto como uma forma de priorizar a escolha de exemplos (i) por uma distribuição discreta uniforme ou (ii) por uma distribuição que dá mais peso aos exemplos com maior erro de previsão.

Finalmente, se $sample_{factor} = +\infty$, não haverá primeira etapa do sorteio – a cada iteração a partir da segunda iteração, T_{WPF} irá avaliar (rotular e calcular o erro) a hipótese corrente em todo o *dataset* ainda não treinado. O parâmetro $sample_{factor}$ pode ser visto como uma medida de quanto iremos *explorar* o *dataset* (espaço de busca) antes de inferir que exemplos serão escolhidos para treino.

A seção 3.11 contém um exemplo ilustrativo do funcionamento de T_{WPF}.

3.8 A estratégia Wrong Probably First + Best Hypothesis (W+BH)

Esta estratégia é quase idêntica à T_{WPF}. A única diferença é que, a cada iteração, T_{W+BH} mantém como hipótese a ser retornada (valor guardado pela variável $h_{selected}$) a hipótese que teve o melhor resultado nas avaliações.

A cada iteração, T_{W+BH} associa ao modelo que acabou de ser treinado um erro de avaliação, calculado com base nos erros e_j da amostra A (detalhada na seção 3.7). O valor de $h_{selected}$ só é atualizado quando o erro de avaliação do modelo corrente é menor do que o erro de avaliação do modelo apontado por $h_{selected}$.

É importante notar que T_{W+BH} não faz nenhum processamento a mais do que T_{WPF}: ao solicitar que L rotule os exemplos da amostra A, também ganhamos uma estimativa do erro do modelo corrente na amostra A.

Finalmente, para tomar a decisão sobre qual hipótese manter apontada pela variável $h_{selected}$, algumas poucas adaptações devem ser feitas na inicialização da estratégia. Além disso, assumimos a existência de mais duas variáveis globals, $\text{error}_{h_{selected}}$ e $\text{error}_{h_{curr}}$, para manter intacatas as assinaturas das funções definidas em 3.1.

Algorithm 5 T_{W+BH}

function START: same as in $T_{SingleBatch}$

function SELECT_FIRST_EXAMPLES: same as in T_{double}

function SELECT_MORE_EXAMPLES: same as in T_{WPF}

function KEEP_EVALUATION: same as in T_{WPF}

function SELECT_EVALUATION_EXAMPLES: same as in T_{WPF}

global variables: $\text{size}_{next_{batch}}$, frac_{start}, $m = |X|$, $\mathbf{v}_{free_{ids}}$
more global variables: $sample_{factor}$, w_{error}, $\mathbf{v}_{eval_{ids}}$
even more global variables: $\text{error}_{h_{curr}}$, $\text{error}_{h_{selected}}$

function CHOOSE_H($h_{selected}, h_{curr}$)
  if $\text{error}_{h_{curr}} \leq \text{error}_{h_{selected}}$ **then**
    $\text{error}_{h_{selected}} \leftarrow \text{error}_{h_{curr}}$
    return h_{curr}
  else
    return $h_{selected}$
end

3.9 A estratégia Uncertain Probably First (UPF)

Uma outra maneira de não escolher exemplos de treino de forma puramente aleatória uniforme é associar a cada exemplo livre (exemplo que ainda não faz parte do conjunto de treino) uma incerteza sobre seu rótulo. Esta incerteza será usada por T_{UPF} da mesma forma que o erro de previsão é utilizado por T_{WPF}: no sorteio de novos exemplos de treino, quanto maior a incerteza sobre o rótulo de um exemplo, maior a chance de este exemplo ser sorteado.

T_{UPF} deve ser visto como uma variação de T_{WPF}, com a única diferença sendo na maneira como os pesos são atribuídos aos exemplos da subamostra de teste (identificados pelos índices de $\mathbf{v}_{eval_{ids}}$ no pseudocódigo de T_{WPF}), dos quais alguns exemplos serão sorteados para compor o novo conjunto de treino.

Ao contrário da noção de erro de previsão, que sempre existe quando tratamos de aprendizado supervisionado, a noção de incerteza de previsão não é dada (não é necessária) pelo próprio problema que queremos resolver. Segue disso que a noção de incerteza depende da espécie de L. Se L é uma Árvore de Regressão, por exemplo, podemos medir a incerteza de previsão de um exemplo e pelo desvio-padrão dos rótulos que compõe o nó-folha a que e foi direcionado (quando percorre esta árvore a partir da raiz). Já se L for um modelo de *ensemble*[12], a incerteza poderia ser associada a alguma medida de dispersão dos rótulos dados pelos modelos ao exemplo e.

Como, por premissa, L deve ser visto como uma caixa-preta, faremos uma concessão à especificação de T_{UPF}: na função de inicialização da estratégia, iremos conferir se L é um dos algoritmos para os quais sabemos calcular incerteza de previsão[13]. Caso positivo, a estratégia segue seu curso; caso negativo, a estratégia aborta sua execução.

Nos experimentos que realizamos, o único L para o qual estimamos uma incerteza foi L = Random Forest. Para esse caso, a incerteza sobre um exemplo e é o desvio-padrão dos rótulos de e de acordo com cada árvore da Random Forest. Em outras palavras, para estimar a incerteza de e (de acordo com uma Random Forest treinada, especificada), rotulamos e de acordo com cada árvore do modelo, formando uma lista de valores (previsões do modelo), e em seguida calculamos o desvio-padrão dessa lista de valores.

Não apresentaremos o pseudocódigo de T_{UPF}, dada sua semelhança com T_{WPF}.

Os resultados experimentais de T_{UPF} serão exibidos apenas na seção 4.9.

[12]Um modelo que combina modelos de aprendizado

[13]Isto é uma concessão porque a definição do problema não assume que temos essa informação sobre L

3.9.1
O paradigma de Active Learning

A estratégia T_{UPF} é inspirada no paradigma de *Active Learning*, em que exemplos não-rotulados são abundantes, mas obter os rótulos é custoso (é difícil, é caro ou exige tempo, conforme [16]). Isto porque, ao estimar a incerteza da previsão de exemplos, não usamos a informação do rótulo destes exemplos (apesar de disponível). É como se, ao querermos buscar n exemplos para incrementar o conjunto de treino, tivéssemos uma subamostra com $sample_{factor} \cdot n$ exemplos não-rotulados disponíveis. Diante dessa carência artificial de rótulos, a estratégia se baseia na incerteza de previsão dos exemplos dessa subamostra para escolher quais n exemplos iremos rotular e consequentemente disponibilizar para L treinar.

3.10
Breve discussão sobre a complexidade de algumas etapas das estratégias

As estratégias T_{WPF} e T_{W+BH} investem parte de seu tempo de treinamento em selecionar exemplos de forma aleatória não-uniforme.

Apesar de este trabalho ser essencialmente empírico, é válido observar que estas estratégias não pioram a complexidade do tempo total gasto em função do número exemplos treinados (em relação à estratégia T_{double}).

Isto porque, para treinar um conjunto de tamanho n[14], além das chamadas para treino do modelo idênticas às chamadas feitas por T_{double}, as estratégias T_{WPF} e T_{W+BH} executam as seguintes tarefas:

1. Sortear, de forma aleatória uniforme, $\mathcal{O}(n) \cdot sample_{factor}$ exemplos
2. Rotular $\mathcal{O}(n) \cdot sample_{factor}$ exemplos
3. Calcular o erro de previsão de $\mathcal{O}(n) \cdot sample_{factor}$ exemplos
4. Sortear, de acordo com pesos, $\mathcal{O}(n)$ exemplos

Supondo que as tarefas de sortear, rotular e calcular erro de previsão não são mais caras do que treinar, o tempo gasto treinando é assintoticamente pelo menos tão grande quanto o tempo gasto com todas essas tarefas.

Na prática, mesmo que as estratégias não sejam assintoticamente mais caras, esse tempo gasto sem treinar pode resultar num subconjunto final de treino menor (do que o de T_{double}).

[14]Para treinar um conjunto de tamanho n, T_{double} irá treinar conjuntos de tamanho $1, 2, 4, \cdots, n$. A soma de todos esses tamanhos (progressão geométrica de razão 2 e $log(n) + 1$ termos) é menor do que $2n$.

Supondo que, entre as tarefas sortear, rotular, calcular erro de previsão, a mais cara seja rotular, o tempo gasto sem treinar será menos impactante quando a complexidade de rotular for bem menor do que a de treinar. Este é o caso de muitos algoritmos de aprendizado[15].

3.11 Exemplo da estratégia Wrong Probably First (WPF)

Apresentamos um exemplo ilustrativo da estratégia T_{WPF} para um *dataset* com três grafias diferentes das letras A, B, C, D e E, totalizando 15 exemplos, cujo objetivo é identificar a posição da letra no alfabeto ($A = 1, B = 2, \ldots, E = 5$). O erro é uma função das posições correta e prevista de cada exemplo. Assumimos os seguintes valores para os parâmetros de T_{WPF}:

- $frac_{start} = 1/15$, de forma que o conjunto de treino inicial tem tamanho 1
- $\text{sample}_{factor} = 2.5$
- $w_{error} = 1/3$

Alguns esclarecimentos são essenciais para entender a figura:

1. Cada linha da figura 3.1 corresponde a uma iteração da estratégia.
2. Cada coluna corresponde a uma etapa de uma iteração
   (a) A primeira etapa representa o conjunto de exemplos livres no início da iteração
   (b) A segunda etapa representa o conjunto de exemplos escolhidos para serem avaliados (dos quais um subconjunto será escolhido para incrementar o conjunto de treino)
   (c) A terceira etapa representa o conjunto de treino ao fim da iteração.
3. As letras em vermelho representam os exemplos escolhidos para a próxima etapa (da esquerda para a direita)
4. Os erros de previsão apresentados na coluna do meio são valores arbitrários. Não é possível deduzi-los (calcula-los) a partir da figura.

[15]Mas certamente não é o caso de todos. Atualizar os pesos de uma rede neural artificial (de acordo com um exemplo) é tão caro quanto calcular o rótulo deste exemplo.

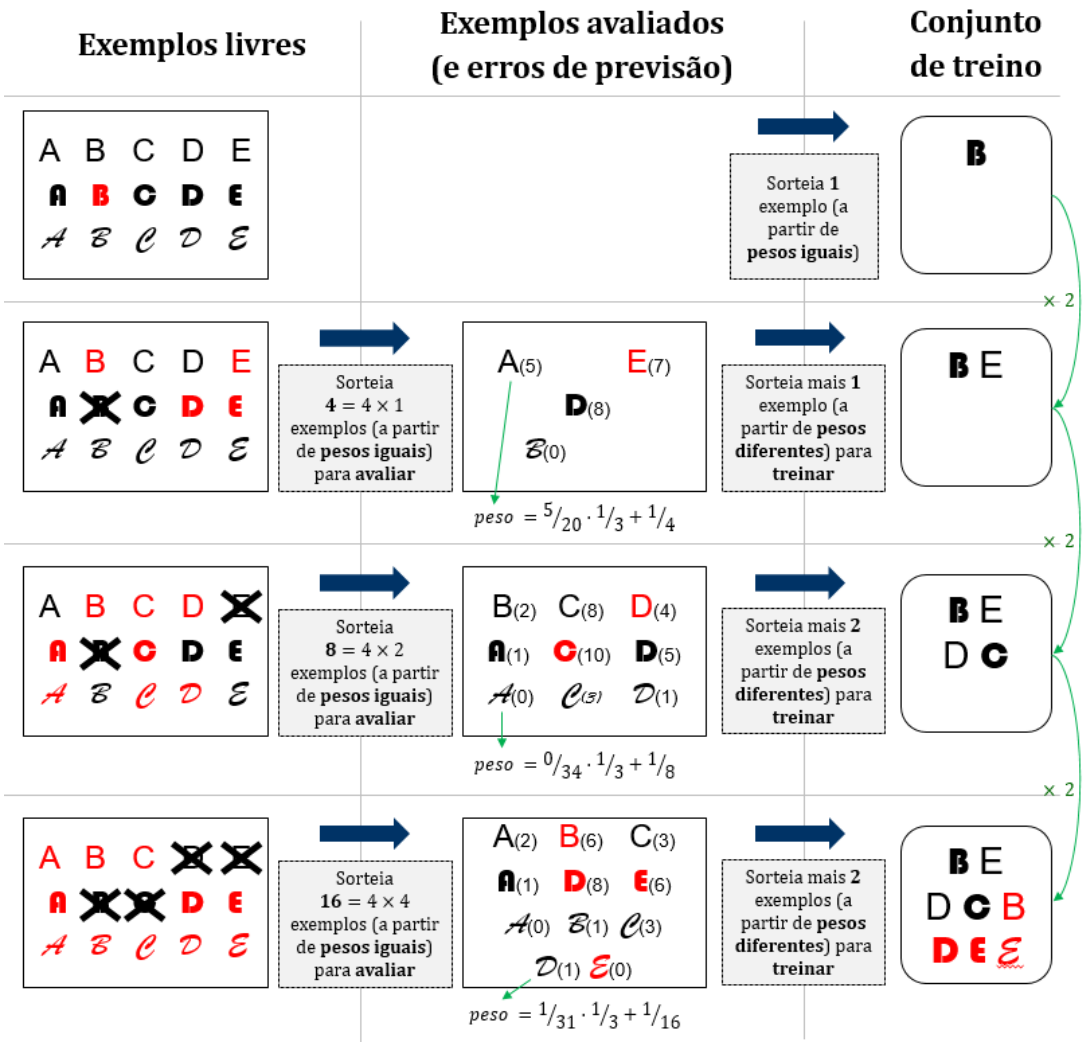

Figura 3.1: Exemplo com quatro iterações da estratégia T_{WPF}.
$sample_{factor} = 4$ $frac_{start} = 1/15$ $w_{error} = 1/3$

4
Avaliação Experimental

Este capítulo trata dos experimentos realizados para medir a qualidade das estratégias descritas na seção 3. Como premissa, assumimos que se houvesse tempo suficiente o melhor que poderíamos fazer é treinar L com o *dataset* inteiro. Considerando a possibilidade de não haver tempo hábil, uma solução natural é selecionar exemplos aleatoriamente e ir aumentando gradualmente a parcela de exemplos treinados. Esta solução é a estratégia T_{double}.

Os experimentos, em síntese, consistem em verificar, ao longo do intervalo de tempo $(0, t_{lim}]$[1], a *qualidade* dos modelos produzidos pelas estratégias de seleção de exemplos (ou de crescimento gradual do *dataset* treinado).

O objetivo principal dos experimentos foi comparar as estratégias T_{W+BH} e T_{double}.

4.1
Datasets

Foram realizados experimentos com 12 *datasets*, descritos na tabela 4.1. Os *datasets* foram retirados de dois repositórios: UCI e Kaggle. Cada dataset passou pelas seguintes transformações:

1. dados categóricos sem noção de ordem foram transformados em colunas com *one hot encoding*

2. dados categóricos com noção de ordem foram transformados em números inteiros a partir de 0

3. cada atributo foi normalizado[2]

[1] O valor de t_{lim} depende do par (L, *dataset*). Isso vai ser melhor explicado ao longo deste capítulo.

[2] A normalização foi feita com base nos dados do conjunto de treino e aplicada sobre o conjunto de treino e sobre o conjunto de teste

Tabela 4.1: Datasets

Nome	# exemplos	# atributos
superconductivty	21.263	81
hydraulic system valve condiciton	2.205	154
pmsm temperature	998.070	8
zap imoveis sp	50.898	14
dielectron	99.915	16
Bike-Sharing-Dataset hour	17.379	31
sgemm product	24.1600	14
default of credit card clients	30.000	32
SeoulBikeData	8.760	15
transcoding mesurment	68.784	17
spotify	174.389	13
tabular playground	300.000	14

Cada *dataset* foi embaralhado e em seguida separado em dois conjuntos: um conjunto de treino com 70% dos exemplos e um conjunto de teste com 30% dos exemplos.

4.2 Algoritmos de aprendizado

Foram realizados experimentos com 3 algoritmos de aprendizado:

- Support Vector Regression (SVR)
- Decision Tree Regressor
- Random Forest Regressor

Foram escolhidos algoritmos de aprendizado comumente usados em problemas de regressão. Não fizemos experimentos com Regressão Linear devido ao tempo de execução muito curto desse algoritmo (para treinar e rotular), o que torna difícil simular o cenário em que o tempo é um recurso escasso.

O algoritmo de Árvore de Regressão é o disponibilizado no pacote *scikitlearn*, de Python, e é baseado nos modelos descritos em [11] e [3]. Cada folha da árvore é associada a média de seus exemplos de treino. A configuração padrão do algoritmo neste pacote é tal que os nós da árvore são expandidos até o erro em cada nó folha ser zero (até que todos os exemplos do nó folha tenham o mesmo rótulo).

O algoritmo de Random Forest (Regressor) é um *ensemble* de Árvores de Regressão: o algoritmo treina 100 Árvores de Regressão em 100 subconjuntos

de treino sorteados aleatoriamente. O rótulo de um exemplo corresponde à média dos rótulos de cada uma das 100 árvores.

O algoritmo de *Support Vector Regression* (SVR) também é o disponibilizado no pacote *scikitlearn*, de Python. Trata-se de uma implementação de *Epsilon-Support Vector Regression*, descrita em [18] e [4]. O algoritmo segue uma ideia bastante semelhante a de algoritmos de SVM de classificação. Dado uma tolerância $\epsilon > 0$ e um *dataset* $\{(x_1, y_1), \ldots (x_m, y_m)\}$, procuramos a melhor curva da forma $f(x) = w \cdot x + b$ que satisfaz $|f(x_i) - y_i| \leq \epsilon, \quad i = 1 \ldots m$. A melhor curva é aquela que minimiza $||w||^2$. Quando não existe função que aproxima todos os pontos x_i a y_i dentro da tolerância ϵ — que é o equivalente, para classificação, de não existir hiperplano que separa perfeitamente os pontos —, o problema de minimização é reescrito com variáveis de folga, para permitir (mas penalizar) que alguns pontos estejam a uma distância maior do que ϵ da curva. Em outras palavras, é um problema de minimização, em que queremos achar uma curva (hiperplano) que minimiza distâncias (entre a curva e os rótulos y_i), mas distâncias menores do que ϵ são toleradas (somam zero a função-objetivo a ser minimizada).

4.3 Erro esperado e erro empírico

Na teoria de aprendizado estatístico ([17]), o *erro esperado* de uma hipótese h é dado por[3]

$$L_{\mathcal{D}}(h) \stackrel{def}{=} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}[\ell(h(x), y)]$$

, em que:

- $\mathcal{D}$ é a distribuição de probabilidade dos exemplos[4]
- ℓ é a função de perda $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

Como por premissa não conhecemos $\mathcal{D}$, calcularemos o erro empírico de h (sobre um *dataset* ou uma amostra $(X, Y) = S$), dado por

$$L_S(h) \stackrel{def}{=} \frac{1}{|S|} \sum_{(x,y)\in S} \ell(h(x), y)$$

Finalmente, como esta dissertação trata de problemas de regressão, utilizamos como função de perda ℓ o erro quadrático.

[3] A definição pode ser ainda mais geral, e esta é uma simplificação para o caso de aprendizado supervisionado em problemas de regressão

[4] No caso geral de aprendizado supervisionado, a distribuição de probabilidade é sobre o conjunto $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$

$$L_S(h) \overset{def}{=} \frac{1}{|S|} \sum_{(x,y)\in S} (h(x_i) - y_i)^2 \overset{def}{=} \frac{1}{m} \sum_{i=1}^{m} (h(x_i) - y_i)^2$$

O erro empírico é calculado sobre um conjunto de exemplos que não fez parte do treino, chamado conjunto de teste. Este conjunto não é enxergado pela estratégia.

4.4 Medida de qualidade de uma estratégia

A medida de qualidade de uma estratégia num instante de tempo t — sobre um *input* (X, Y, L, t_{lim}) e um *dataset* de teste (X_{test}, Y_{test}) — é o *erro empírico* da hipótese apontada por $h_{selected}$ no instante t, erro esse medido sobre (X_{test}, Y_{test}). Lembramos aqui que $h_{selected}$ é, por definição, a hipótese que a estratégia selecionaria se o experimento terminasse naquele instante - e que isto muda ao longo do tempo, conforme a estratégia obtém novas hipóteses de L.

Como este trabalho é sobretudo empírico, apresentamos duas maneiras de medir a qualidade de uma estratégia: o erro do modelo retornado ao final do experimento e o erro do modelo ao longo do experimento. A partir daqui, usaremos como abuso de notação o termo "erro da estratégia" como sinônimo de "erro da hipótese apontada pela estratégia".

Dado um input (X, Y, L, t_{lim}) e um dataset de teste (X_{test}, Y_{test}), defina para a estratégia T:

- $modelo_T(t)$ = modelo (treinado) apontado por T no instante t (ou simplesmente o valor de $h_{selected}$ no instante t)
- $erro_T(t)$ = erro empírico de $modelo_T(t)$ aplicado sobre (X_{test}, Y_{test})
- $erro_{full}$ = erro empírico do modelo que treina com (X, Y) sem restrição de tempo sobre (X_{test}, Y_{test}) (ou simplesmente o erro empírico da hipótese retornada por $T_{SingleBatch}$)
- $erro_{rel_T}(t) = erro_T(t)/erro_{full}$

A justificativa para a definição e o uso $erro_{rel_T}(t)$ é a seguinte: como estamos tratando de problemas de regressão, para facilitar a comparação entre erros de uma estratégia T em dois *datasets* distintos, usamos como medida de performance o erro de T em relação ao erro do modelo que treina com todo o dataset de treino (X, Y).

Cada trio (*dataset*, *learner*, *teacher*) foi rodado com 5 *seeds*, que influenciam a inicialização do *learner*. O valor de $erro_T(t)$, portanto, é uma média do valor do erro para cada uma das *seeds*.

Finalmente, esclarecemos que colher estas estatísticas demanda tempo, mas este tempo não afetou o tempo-limite de execução de cada experimento. A cada instante t em que $erro_{rel_T}(t)$ é calculado, é como se parássemos o cronômetro que regula o tempo de execução da estratégia.

Em resumo, dado o *input* (X, Y, L, t_{lim}) um dataset de teste (X_{test}, Y_{test}), analisaremos a qualidade de uma estratégia T de duas maneiras:

1. Pelo valor de $erro_{rel_T}(t_{lim})$, erro empírico ao final do experimento
2. Pelo gráfico de $erro_{rel_T}(.)$ em função do tempo, que mostra o erro empírico ao longo do experimento

4.5 Ambiente computacional dos experimentos

Os experimentos foram realizados num processador Core i9-7900X 3.3GHz, com 128GB RAM DDR4, no Windows 10, na linguagem de programação *Python*. Os *learners* foram retirados do pacote *scikitlearn* ([15]) e utilizados com seus parâmetros pré-definidos[5].

Para as estratégias T_{WPF} e T_{W+BH}, a não ser que seja mencionado algo diferente, utilizamos os valores $sample_{factor} = 4$ e $w_{error} = 1/3$. Na seção 4.8, mostramos que a performance dessas estratégias se mantém superior à de T_{double} para diversos outros valores de $sample_{factor}$ e w_{error}.

Para todas as estratégias (exceto $T_{SingleBatch}$, que não tem parâmetros), foi utilizado $\text{frac}_{start} = 1\%$.

4.6 Resultados

Os resultados comparam a performance das estratégias T_{W+BH} e T_{double} conforme o exposto na seção 4.4: pelo erro das estratégias ao fim do tempo-limite e pelo erro da estratégias ao longo do tempo.

O tempo-limite – para um par (L, *dataset*) – foi estabelecido como o tempo para L treinar com todo o conjunto de treino (X, Y), sujeito ao limite

[5]Isso explica por que não estimamos a incerteza da previsão das Árvores de Regressão para a estratégia T_{WPF}: por padrão, as árvores têm altura ilimitada e os nós são expandidos até que cada folha seja pura (todos os exemplos de uma folha têm o mesmo rótulo)

superior de 5 minutos. Este teto de 5 minutos foi arbitrado para permitir análises de sensibilidade dos parâmetros de T_{W+BH}. No apêndice A, exibimos evidências de que tempos diferentes não afetariam a direção dos resultados.

A tabela 4.6 exibe os erros dos modelos treinados por T_{double} e T_{W+BH} para cada par (L, *dataset*) listado nas seções 4.2 e 4.1.

	Árv. de Regressão		Random Forest		SVR	
Dataset	T_{double}	T_{W+BH}	T_{double}	T_{W+BH}	T_{double}	T_{W+BH}
superconductivty	1.57	1.51	1.35	1.18	1.04	0.99
hydraulic system	1.46	1.63	1.21	1.28	1.04	0.97
pmsm temperature	1.96	1.64	2.66	1.84	1.17	1.08
zap imoveis sp	1.24	1.28	1.69	1.08	1.00	0.99
dielectron	1.38	1.18	1.65	1.26	1.22	1.01
Bike-Sharing-hour	1.29	1.25	1.26	1.09	1.03	1.00
sgemm product	4.74	2.40	5.38	2.00	2.12	1.19
default of credit card	0.99	0.97	1.01	1.04	1.00	0.99
SeoulBikeData	1.12	1.13	1.16	1.06	1.04	1.00
transcoding mesurment	0.98	1.02	1.00	1.05	1.00	0.98
spotify	1.11	1.23	1.05	1.01	1.03	1.00
tabular playground	1.01	1.02	1.01	1.01	1.03	1.02
Média das médias	1.57	1.36	1.70	1.24	1.14	1.02
# erro$_{T_{W+BH}}$ < erro$_{T_{Double}}$	6 / 12		8 / 12		12 / 12	

Tabela 4.2: Erros dos modelos finais, das estratégias T_{double} e T_{W+BH}. Os valores com fundo cinza mostram que estratégia foi melhor em cada caso (cada par *Learner*, *dataset*)

O valor 1.16 (na linha SeoulBikeData, coluna Random Forest / T_{double}) se lê da seguinte forma: ao fim do experimento com o algoritmo de aprendizado L = "Random Forest" e o *dataset* SeoulBikeData, a estratégia T_{double} teve um erro empírico 16% superior ao erro empírico da hipótese que L retorna ao treinar com todo o *dataset*.

Para cada um dos *learners*, T_{W+BH} teve em média um erro final inferior ao erro final de T_{double}. No caso das Árvores de Regressão, no entanto, T_{double} se saiu melhor em metade dos *datasets*. Para os outros dois *learners*, T_{W+BH} foi

melhor na maioria dos *datasets*. Em particular, para o SVR, T_{W+BH} produziu um modelo final com erro de teste inferior ao erro do modelo produzido por T_{double} em todos os *datasets*.

As figuras 4.1, 4.2 e 4.3 exibem o erro das estratégias ao longo do tempo, para cada um dos *learners*. Os resultados sugerem que as estratégias são competitivas para Árvores de Regressão e que T_{W+BH} é superior T_{double} nos demais casos - RandomForest e SVR. Neste contexto, a superioridade é evidenciada pelo gráfico de T_{W+BH} estar abaixo (com erro menor) do gráfico de T_{double}.

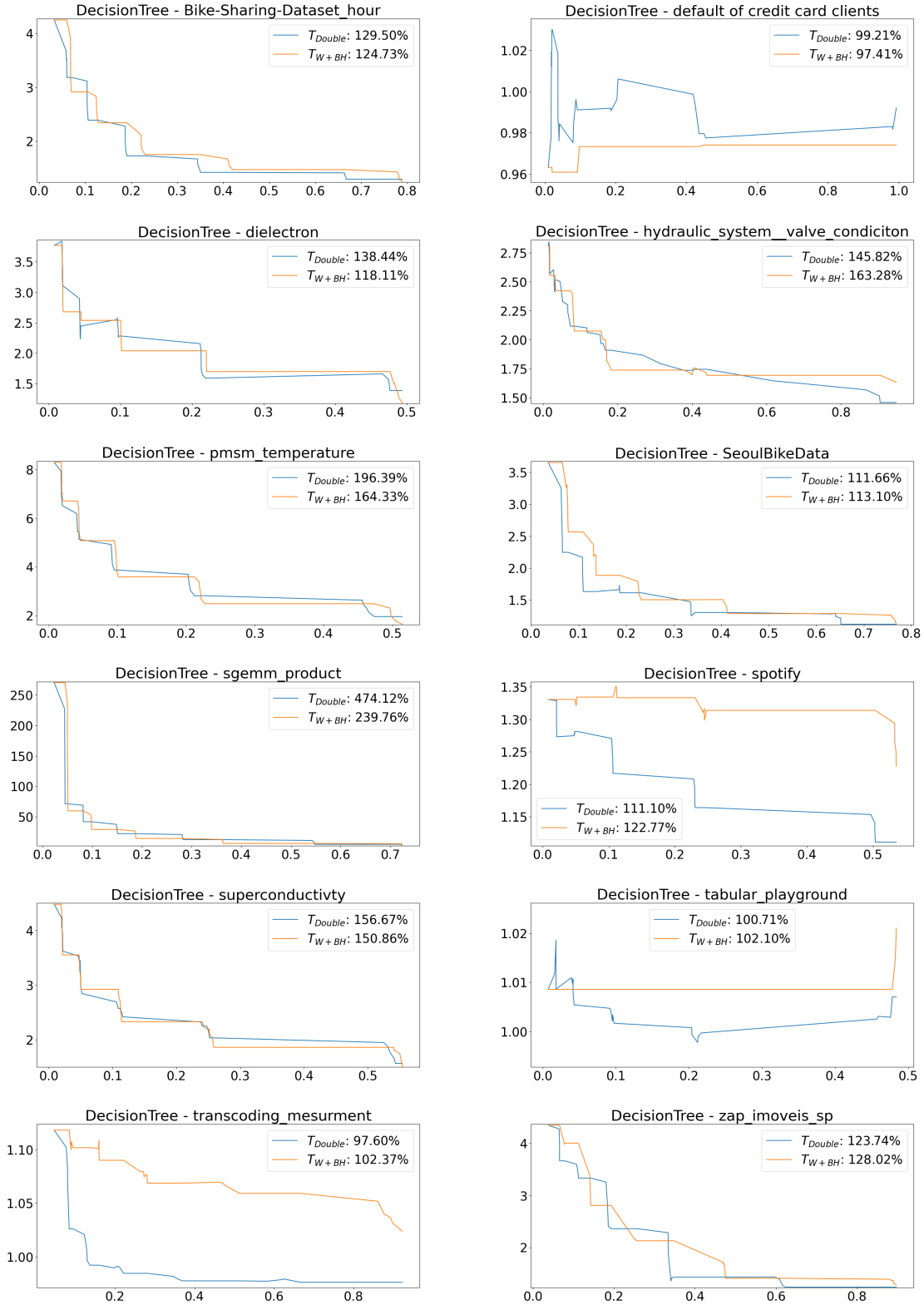

Figura 4.1: Erro médio (eixo vertical) ao longo do tempo (eixo horizontal) no conjunto de teste para as estratégias T_{double} e T_{W+BH}, com L = Árvores de Regressão. A legenda exibe o erro médio ao fim do experimento.

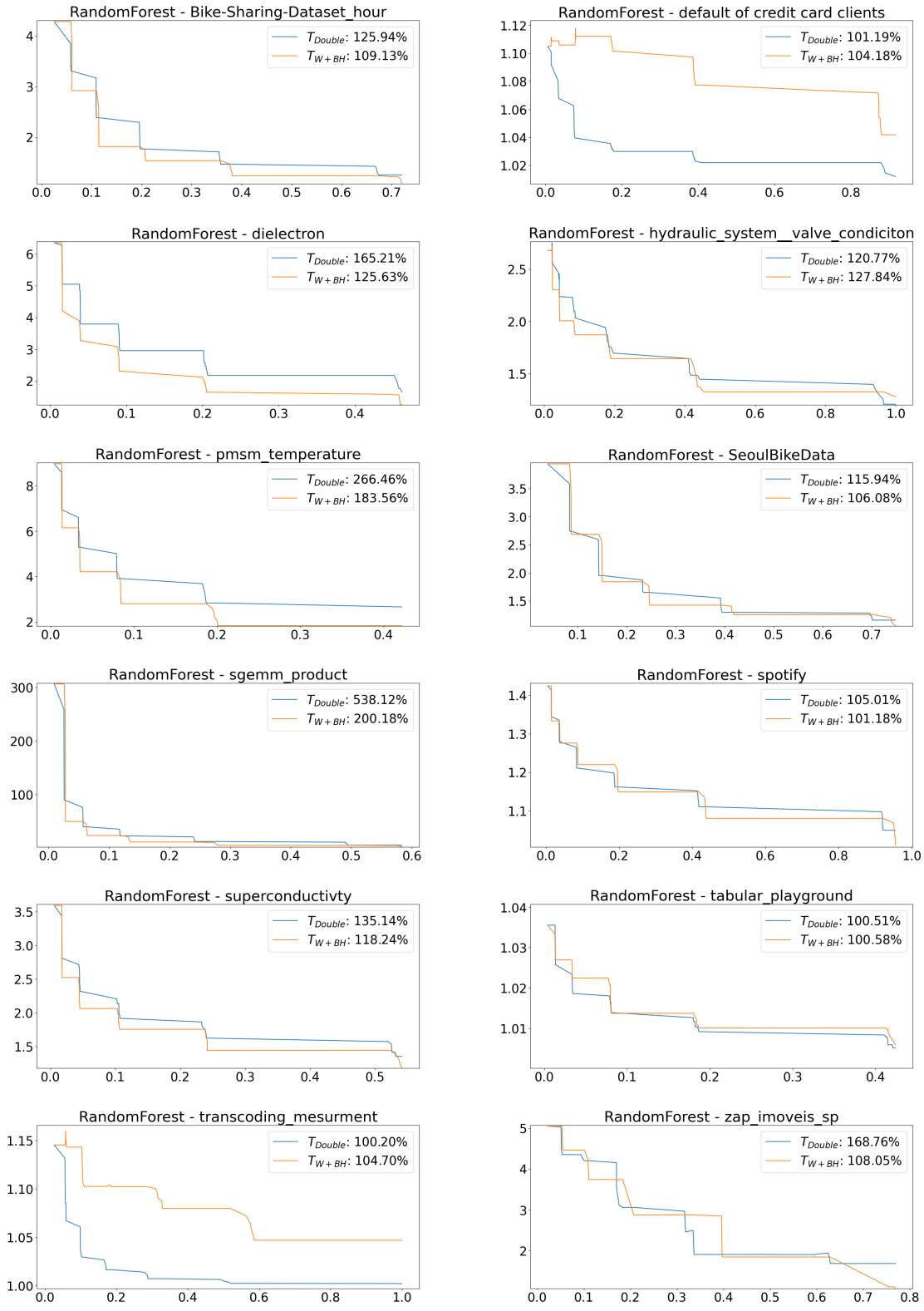

Figura 4.2: Erro médio (eixo vertical) ao longo do tempo (eixo horizontal) no conjunto de teste para as estratégias T_{double} e T_{W+BH}, com L = Random Forest. A legenda exibe o erro médio ao fim do experimento.

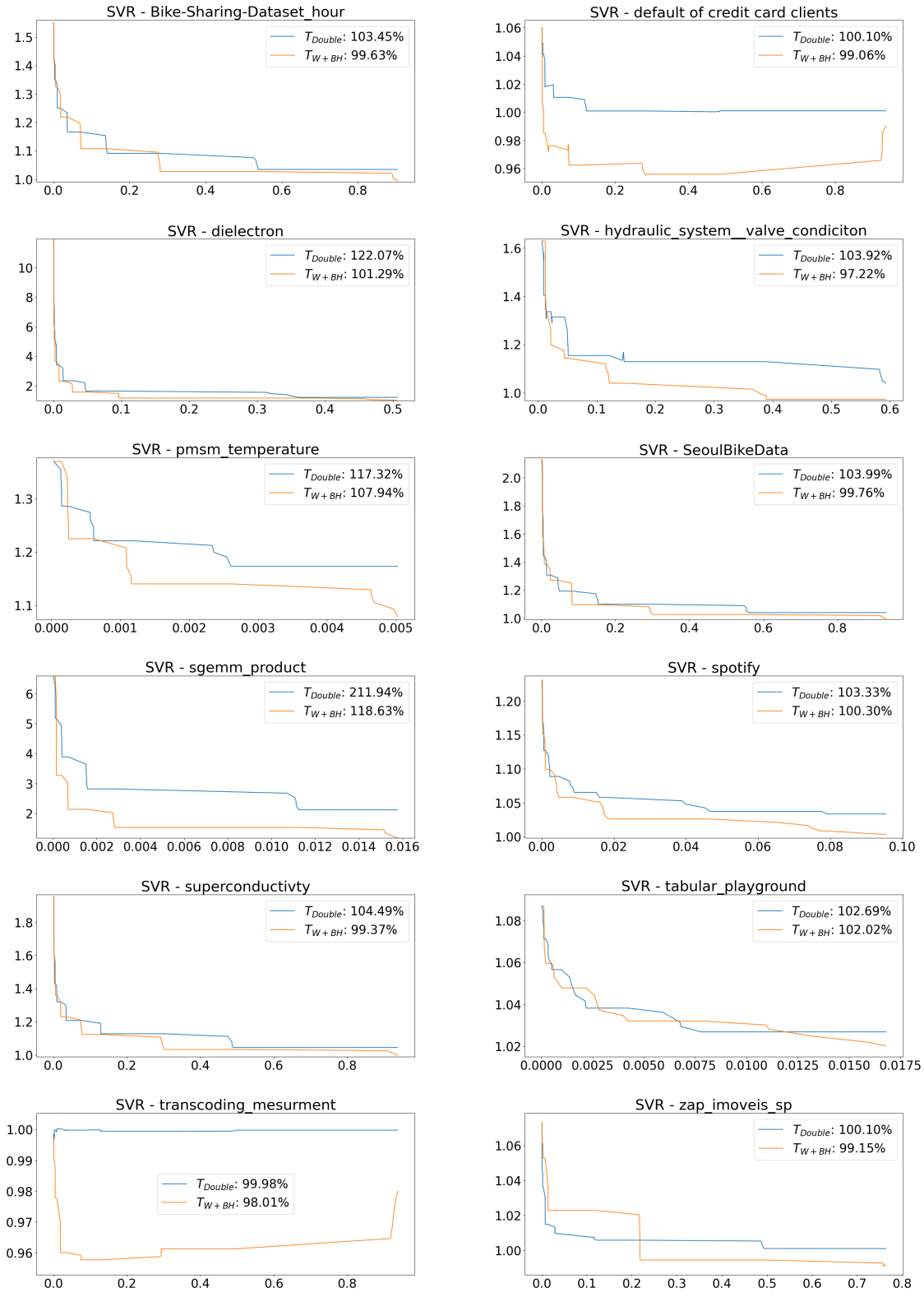

Figura 4.3: Erro médio (eixo vertical) ao longo do tempo (eixo horizontal) no conjunto de teste para as estratégias T_{double} e T_{W+BH}, com $L =$ SVR. A legenda exibe o erro médio ao fim do experimento.

4.6.1 Alguns resultados para Wrong Probably First

As estratégias T_{WPF} e T_{W+BH} são muito semelhantes. A única distinção — conforme exibido pelos pseudocódigos das seções 3.7 e 3.8 — é a política de troca de hipótese: enquanto T_{WPF} sempre seleciona a última hipótese treinada por L, T_{W+BH} só troca de hipótese quando a última hipótese treinada (h_{curr}) teve um erro empírico (no conjunto de avaliação) melhor do que o da hipótese atualmente apontada ($h_{selected}$)[6].

Esta semelhança sugere que as estratégias terão desempenho semelhantes, o que é evidenciado pelos gráficos da figura 4.4, em que as linhas representando o erro de cada uma das estratégias praticamente coincidem.

Nestes experimentos, em vez de um gráfico por par (L, *dataset*), mostramos um gráfico para cada L. Cada gráfico corresponde a uma média dos gráficos fixando L e variando o *dataset*.

Concluímos que T_{W+BH} raramente mantém uma hipótese que não a última treinada, mas, quando toma essa decisão, geralmente o resultado é positivo.

[6]Ou quando nenhuma hipótese existe ainda, caso em que $h_{selected} = NULL$!

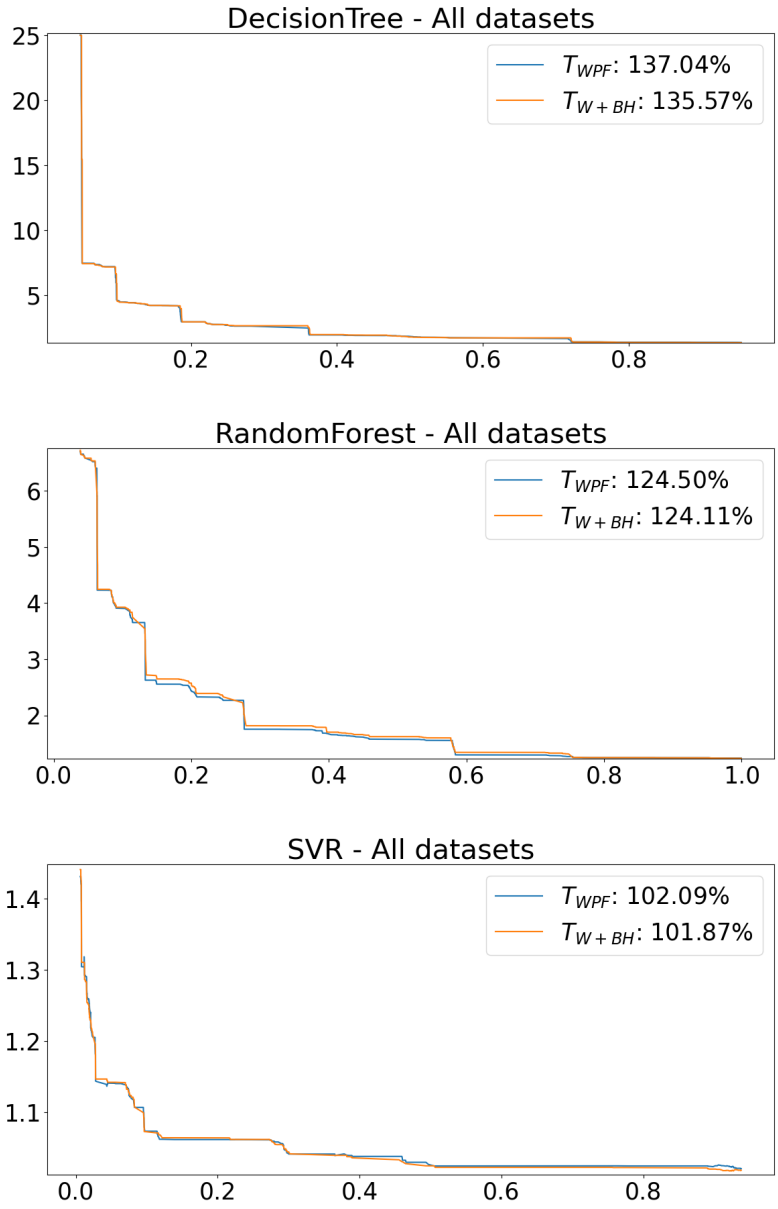

Figura 4.4: Erro médio (eixo vertical) ao longo do tempo (eixo horizontal) no conjunto de teste para as estratégias T_{W+BH} e T_{WPF}. A legenda exibe o erro médio ao fim do experimento.

4.7 O tamanho do conjunto de treino de cada estratégia

A premissa de que exemplos em que a hipótese corrente comete maior erro são mais úteis para treinar um novo modelo é a motivação para as estratégias T_{WPF} e T_{W+BH}. Mesmo que isso seja verdade, este tempo gasto avaliando exemplos reduz o tempo de treinamento, o que se traduz em conjuntos de treino finais menores (ou em hipóteses finais treinadas com menos exemplos do que

as de T_{double}). A tabela 4.7 mostra a relação entre o tamanho do conjunto de treino final de T_{W+BH} e de T_{double}— nos dois casos em comparação ao tamanho total do *dataset* —, destacando em negrito os casos em que houve diferença (a favor de T_{double}, sempre).

Nem sempre os números da tabela serão aproximadamente uma potência de 2, porque cada entrada da tabela é uma média de 5 experimentos (5 valores de *seed*).

	Árv. de Regressão		Random Forest		SVR	
Dataset	T_{double}	T_{W+BH}	T_{double}	T_{W+BH}	T_{double}	T_{W+BH}
superconductivty	0.32	0.32	0.32	0.32	0.64	0.64
hydraulic system	0.50	0.37	0.62	0.37	0.62	0.28
pmsm temperature	0.32	0.32	0.19	0.16	0.08	0.08
zap imoveis sp	0.32	0.32	0.32	0.32	0.64	0.64
dielectron	0.32	0.32	0.32	0.32	0.64	0.64
Bike-Sharing-hour	0.32	0.32	0.32	0.32	0.64	0.64
sgemm product	0.32	0.32	0.32	0.32	0.16	0.16
default of credit card	0.51	0.07	0.64	0.64	0.64	0.64
SeoulBikeData	0.32	0.32	0.32	0.32	0.64	0.64
transcoding mesurment	0.32	0.32	0.45	0.32	0.64	0.64
spotify	0.32	0.32	0.64	0.64	0.32	0.32
tabular playground	0.32	0.13	0.32	0.32	0.16	0.16

Tabela 4.3: Tamanho relativo dos modelos finais. Os valores com fundo cinza mostram que estratégia selecionou mais exemplos em cada caso (cada par *Learner*, *dataset*)

.

De fato, em alguns casos o tempo gasto por T_{W+BH} escolhendo exemplos para treino (em vez de simplesmente sortear alguns exemplos) ocasiona que a estratégia rode uma iteração[7] a menos.

Em 7 pares (L, *dataset*) tivemos T_{double} com conjunto de treino final maior do que o de T_{W+BH}. Desses 7, em 4 deles T_{double} teve desempenho superior.

[7]Vários cientistas da computação usam as palavras "iteragir" e "interagir" sem distinção, o que é um erro. Curiosamente, para falar de estratégias de seleção de exemplos, qualquer uma das duas palavras serve

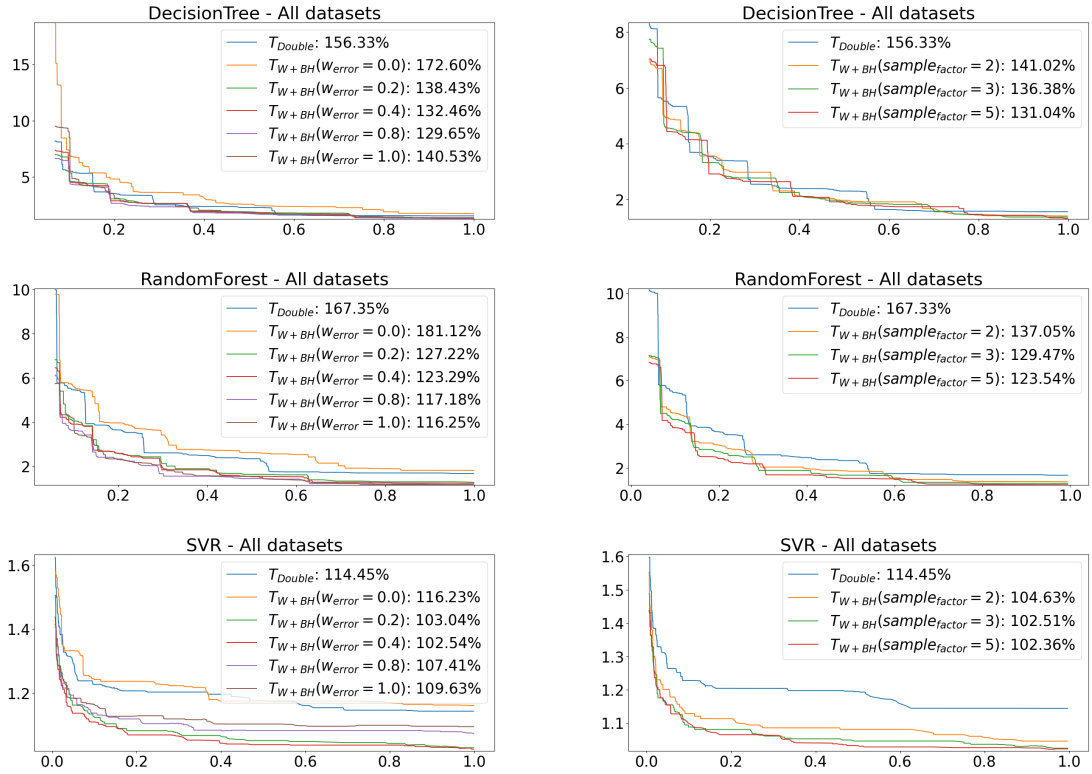

Figura 4.5: Erro médio (eixo vertical) ao longo do tempo (eixo horizontal) no conjunto de teste para a estratégia T_{W+BH}, variando os parâmetro w_{error} e $sample_{factor}$. A legenda exibe o erro médio ao fim do experimento.

Considerando que no geral T_{double} ganhou de T_{W+BH} em apenas 10 dos 36 pares, os resultados sugerem o que é esperado: que conjuntos de treino maiores — todo o resto constante — geram modelos melhores (com menor erro empírico).

4.8 Análise de sensibilidade

A estratégia T_{W+BH} tem dois parâmetros a mais do que a estratégia aleatória T_{double}: $sample_{fator}$ e w_{error}. Conduzimos experimentos para demonstrar que, apesar de a escolha destes parâmetros influenciar a qualidade do modelos de aprendizado retornados pela estratégia, em geral a estratégia T_{W+BH} vai ser superior a T_{double} para um intervalo grande de parâmetros utilizados. A figura 4.5 exibe o erro empírico ao longo do tempo no conjunto de teste para estas duas estratégias.

Quando fazemos w_{error} variar entre 0.2 e 0.8, mantemos inalterado $sample_{factor} = 4$, conforme os valores apresentados na seção 4.5. Analogamente, quando fazemos $sample_{fator}$ variar entre 2 e 5, mantemos inalterado $w_{error} = 1/3$, conforme os valores apresentados na seção 4.5.

Nestes experimentos, em cada uma das duas análises de sensibilidade, em

vez de um gráfico por par (L, *dataset*), mostramos um gráfico para cada L. Cada gráfico corresponde a uma média dos gráficos fixando L e variando o *dataset*. A estratégia $T_{W+BH}(w_{error} = 0.0)$ é um caso degenerado da estratégia T_{W+BH} — corresponde ao T_{Double}, com a desvantagem do tempo gasto rotulando exemplos de forma desnecessária, uma vez que estes rótulos (ou seus erros) não serão usados no sorteio de exemplos.

4.9 Experimentos com a estratégia baseada em incerteza

A estratégia T_{UPF} sorteia novos exemplos de treino com base na incerteza de previsão destes exemplos. Seus detalhes foram descritos na seção 3.9.

Só fizemos experimentos com L = Random Forest. Para Árvores de Regressão, as configurações pré-estabelecidas pelo pacote *scikitlearn* geram uma árvore com erro zero sobre o conjunto de treino (cada nó é expandido até que não haver exemplos com rótulos diferentes).

Os gráficos da figura 4.6 e a tabela 4.9 comparam a qualidade das estratégias T_{double} e T_{UPF}. A estratégia T_{UPF} teve desempenho inferior à T_{double} na maioria dos casos (*datasets*).

No entanto, se nos restringirmos aos casos em que T_{UPF} conseguiu treinar com um conjunto tão grande quando o de T_{double}, a estratégia se mostra competitiva. Isto sugere que a explicação para o desempenho ruim de T_{UPF} foi o tempo gasto inferindo que exemplos usar para treino (em vez de treinando).

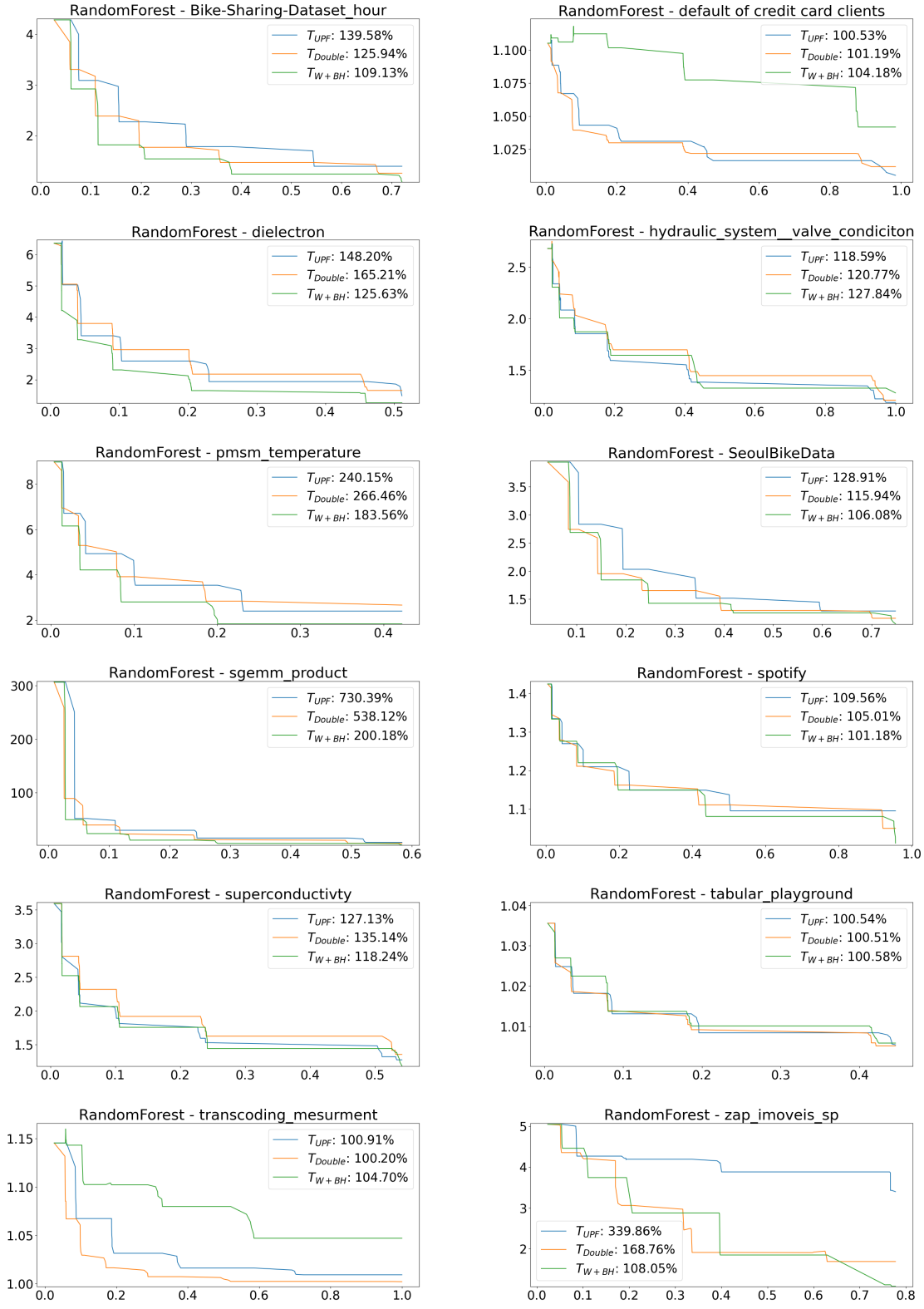

Figura 4.6: Erro médio (eixo vertical) ao longo do tempo (eixo horizontal) no conjunto de teste para as estratégias T_{UPF}, T_{double} e T_{W+BH}, com L = Random Forest. A legenda exibe o erro médio ao fim do experimento.

Dataset	T_{double}	T_{UPF}	$\frac{size(T_{UPF})}{size(T_{Double})}$
superconductivty	1.35	1.27	1
hydraulic system	1.21	1.19	1
pmsm temperature	2.66	2.40	0.8
zap imoveis sp	1.69	3.40	0.5
dielectron	1.65	1.48	1
Bike-Sharing-hour	1.26	1.40	0.5
sgemm product	5.38	7.30	0.5
default of credit card	1.01	1.01	1
SeoulBikeData	1.16	1.29	0.5
transcoding mesurment	1.00	1.01	0.4
spotify	1.05	1.10	0.5
tabular playground	1.01	1.01	1
Média das médias	1.70	1.90	-
# $\text{erro}_{T_{UPF}} < \text{erro}_{T_{Double}}$	5 / 12		-

Tabela 4.4: Erros dos modelos finais, das estratégias T_{double} e T_{UPF}, para $L =$ Random Forest. Os valores com fundo cinza mostram que estratégia foi melhor em cada caso (cada *dataset*). A última coluna exibe o tamanho do conj. de treino final de T_{UPF} em relação ao de T_{double}.

5
Conclusões e Trabalhos Futuros

5.1
Algumas considerações

Nossa abordagem geral de estratégias (seção 3.1), assume que L não é um algoritmo online. Primeiro, é importante esclarecer que esta premissa não faz parte da definição geral dada ao problema. Segundo, o pseudocódigo pode ser adaptado a este cenário trocando poucas linhas de código (na verdade, apenas a instrução $L.FIT(\mathbf{X}_{train}, \mathbf{Y}_{train})$). Poderíamos inclusive inserir como *input* da estratégia uma variável booleana para indicar se L é ou não online (incremental). Em resumo, essa premissa foi assumida apenas para tornar o pseudocódigo mais simples.

A estratégia que adapta T_{double} fazendo um *ensemble* das hipóteses produzidas ao longo das interações com L não é uma estratégia válida. Isto porque a hipótese h' formada por esse *ensemble* pode não ser uma das hipóteses que L é capaz de produzir, o que faz parte da exigência do *output* da estratégia.

Essa exigência foi feita pensando na estratégia como um *wrapper* de L. Do ponto de vista conceitual, pode ser que o usuário, ao escolher $L =$ Árvore de Regressão, queira uma árvore como hipótese (por exemplo, porque o usuário quer obter hipóteses fáceis de compreender). Do ponto de vista prático, pensando na implementação da estratégia numa determinada linguagem de programação, o usuário pode esperar que o objeto retornado pela estratégia tenha uma estrutura de dados específica. Por exemplo, na linguagem *Python*, as árvores de regressão retornadas pela chamada do método fit de um objeto *DecisionTreeRegressor* do pacote *sklearn* são uma estrutura de dados com atributos e métodos específicos[1], que o usuário pode assumir numa etapa posterior do seu próprio código.

[1]Esse objeto retornado por fit tem os atributos $n\_samples$ e $max\_features$, cujo significado não é importante nesse contexto

5.2 Conclusões

Nesse trabalho, discutimos estratégias de seleção de exemplos (de treino) em problemas de regressão, cenário em que exemplos rotulados são abundantes e o tempo total de processamento dos dados é escasso. Para formalizar o problema, buscamos uma adaptação do paradigma de *Iterative Machine Teaching*, proposto em [13].

Nossa estratégia T_{W+BH} seleciona exemplos com base em seu erro de predição e foi avaliada de forma empírica. No conjunto robusto de experimentos que realizamos, T_{W+BH} teve desempenho superior a T_{double}, uma solução natural para o problema em questão, baseada no *doubling trick*.

Como trabalhos futuros, enxergamos três caminhos: primeiro, tentar obter garantias teóricas (de convergência a uma hipótese-alvo) para as estratégias propostas; segundo, explorar o erro de previsão de exemplo de outras maneiras (como limitar a contribuição do erro ao peso no sorteio, para diminuir a influência de *outliers*); terceiro, elaborar estratégias que façam seleção de exemplos com base em outras estatísticas que não o erro de previsão, como a incerteza de previsão.

Referências Bibliográficas

[1] BOTTOU, L.. **On-line learning and stochastic approximations**. In: IN ON-LINE LEARNING IN NEURAL NETWORKS, p. 9–42. Cambridge University Press, 1998.

[2] BOTTOU, L.; BOUSQUET, O.. **The tradeoffs of large scale learning**. In: Platt, J.; Koller, D.; Singer, Y. ; Roweis, S., editors, ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, volumen 20. Curran Associates, Inc., 2008.

[3] BREIMAN, L.; FRIEDMAN, J. H.; OLSHEN, R. A. ; STONE, C. J.. **Classification and regression trees**. Routledge, 2017.

[4] CHANG, C.-C.; LIN, C.-J.. **LIBSVM: A library for support vector machines**. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[5] CHEN, Y.; SINGLA, A.; AODHA, O. M.; PERONA, P. ; YUE, Y.. **Understanding the role of adaptivity in machine teaching: The case of version space learners**, 2018.

[6] CICALESE, F.; FILHO, S.; LABER, E. ; MOLINARO, M.. **Teaching with limited information on the learner's behaviour**. In: III, H. D.; Singh, A., editors, PROCEEDINGS OF THE 37TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, volumen 119 de **Proceedings of Machine Learning Research**, p. 2016–2026. PMLR, 13–18 Jul 2020.

[7] COHN, D. A.; GHAHRAMANI, Z. ; JORDAN, M. I.. **Active learning with statistical models**. CoRR, cs.AI/9603104, 1996.

[8] DASGUPTA, S.; HSU, D.; POULIS, S. ; ZHU, X.. **Teaching a black-box learner**. In: Chaudhuri, K.; Salakhutdinov, R., editors, PROCEEDINGS OF THE 36TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, volumen 97 de **Proceedings of Machine Learning Research**, p. 1547–1555. PMLR, 09–15 Jun 2019.

[9] DU, J.; LING, C.. **Active teaching for inductive learners**. In: SDM, 2011.

[10] GOLDMAN, S.; KEARNS, M.. **On the complexity of teaching**. Journal of Computer and System Sciences, 50(1):20–31, 1995.

[11] HASTIE, T.; TIBSHIRANI, R. ; FRIEDMAN, J.. **The Elements of Statistical Learning**. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[12] LEWIS, D. D.; CATLETT, J.. **Heterogeneous uncertainty sampling for supervised learning.** In: Cohen, W. W.; Hirsh, H., editors, ICML, p. 148–156. Morgan Kaufmann, 1994.

[13] LIU, W.; DAI, B.; HUMAYUN, A.; TAY, C.; YU, C.; SMITH, L. B.; REHG, J. M. ; SONG, L.. **Iterative machine teaching**. In: Precup, D.; Teh, Y. W., editors, PROCEEDINGS OF THE 34TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, ICML 2017, SYDNEY, NSW, AUSTRALIA, 6-11 AUGUST 2017, volumen 70 de **Proceedings of Machine Learning Research**, p. 2149–2158. PMLR, 2017.

[14] LIU, W.; DAI, B.; LI, X.; LIU, Z.; REHG, J. M. ; SONG, L.. **Towards black-box iterative machine teaching**, 2018.

[15] PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M. ; DUCHESNAY, E.. **Scikit-learn: Machine learning in Python**. Journal of Machine Learning Research, 12:2825–2830, 2011.

[16] SETTLES, B.. **Active learning literature survey**. 2009.

[17] SHALEV-SHWARTZ, S.; BEN-DAVID, S.. **Understanding Machine Learning: From Theory to Algorithms**. Cambridge University Press, USA, 2014.

[18] VAPNIK, V. N.. **Statistical Learning Theory**. Wiley-Interscience, 1998.

[19] ZILLES, S.; LANGE, S.; HOLTE, R. ; ZINKEVICH, M.. **Models of cooperative teaching and learning**. The Journal of Machine Learning Research, 12:349–384, 02 2011.

A
Experimentos com outros limites de tempo

Os experimentos da seção 4.6 foram realizados com limite de tempo distinto para cada par (L, *dataset*). O valor foi o tempo que L leva para treinar com todo o *dataset*, respeitando o limite superior de 5 minutos.

Este corte (limite superior) foi arbitrado para permitir a análise de sensibilidade dos parâmetros da estratégia T_{W+BH}, na seção 4.8.

Nos gráficos da figura A.1, mostramos que com um limite de tempo distinto, de 1 minuto, as principais conclusões continuam válidas: T_{W+BH} se mostrou superior a T_{double} em geral, com performance similar à de T_{double} no caso de Árvores de Regressão e melhor do que a de T_{double} nos demais casos.

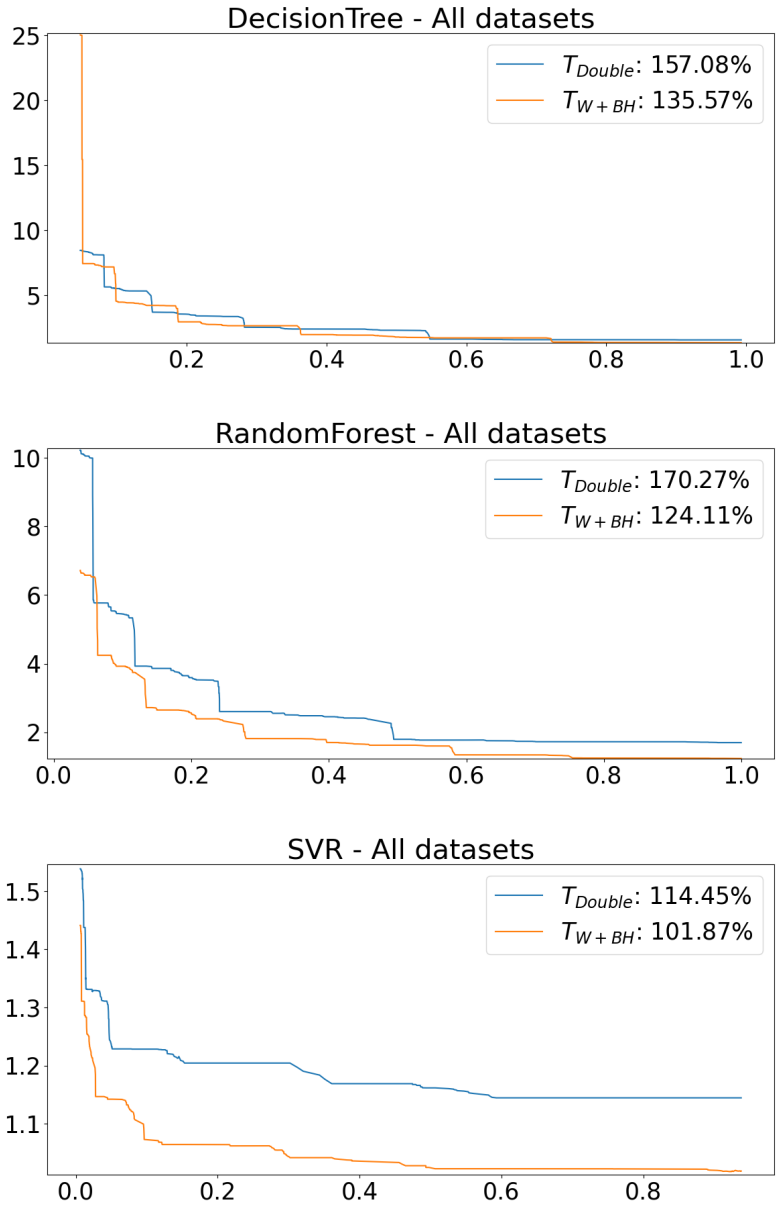

Figura A.1: Erro médio (eixo vertical) ao longo do tempo (eixo horizontal) no conjunto de teste para as estratégias T_{double} e T_{W+BH} para $L \in$ {Árvore de Regressão, Random Forest, SVR}, com o tempo-limite alterado para 1 minuto. A legenda exibe o erro médio ao fim do experimento.

Assim como na seção 4.8, os gráficos (para cada L), correspondem a médias dos gráficos por *dataset*.