Mateus Levi Simões Fernandes

Métodos de classificação estatística: uma comparação

RELATÓRIO DE PROJETO FINAL

DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DEPARTAMENTO DE INFORMÁTICA

Programa de Graduação em Engenharia de Computação

Rio de Janeiro
Junho de 2022

Pontifícia Universidade Católica
do Rio de Janeiro

Mateus Levi Simões Fernandes

Métodos de classificação estatística: uma comparação

Relatório de Projeto Final

Relatório de Projeto Final, apresentado ao programa de Engenharia de Computação da PUC-Rio como requisito parcial para a obtenção do titulo de Engenheiro de Computação.

Orientador: Prof. Cristiano Augusto Coelho Fernandes

Rio de Janeiro
Junho de 2022

Agradecimentos

A Deus, pela minha vida e por tudo nela.

Aos meus pais, Marcelo e Andréa, por todo o amor e cuidado.

Ao professor Cristiano Augusto, pela orientação e apoio necessários para a execução deste projeto.

A todos os professores embaixo dos quais estudar, por terem fomentado meu desejo de aprender e me ajudado a chegar até esse ponto.

À Deborah Sanches e toda a equipe do SOU-CTC, por terem me incentivado e me dado uma oportunidade inesquecível, na forma da bolsa TEPP, ainda no início da graduação.

A todos os amigos que fiz ao longo da graduação, por terem feito os desafios dos últimos anos experiências muito melhores de se enfrentar do que normalmente seriam.

Resumo

Fernandes, Mateus; Fernandes, Cristiano. **Métodos de classificação estatística: uma comparação**. Rio de Janeiro, 2022. 62p. Projeto de Graduação – Departamento de Engenharia Elétrica e Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Esse projeto teve como objetivo a análise de diferentes métodos de classificação estatística, visando entender os fundamentos por trás de cada um e suas características. Os métodos investigados foram: regressão logística, *linear discriminant analysis*, *k-nearest neighbours*, árvores de classificação, *random forests* e *support vector machines*. Após um estudo conceitual de cada método, estes foram aplicados a três diferentes bases de dados: *Taiwan Company Bankruptcy*, *Student Alcohol Consumption* e PNAD COVID19. Os modelos foram implementados utilizando o pacote **caret**, disponível na linguagem R. A partir de um conjunto de métricas como acurácia, ROC-AUC e *log loss*, foram escolhidos como mais acurados os modelos de *gradient boosting* e *random forests*. Assim, concluiu-se que métodos de classificação baseados em *ensemble* possuem melhor capacidade preditiva de forma geral, apesar de outros métodos também serem capazes de gerarem bons resultados e possuirem outras características vantajosas como interpretabilidade e tempo de execução.

Palavras-chave

Classificação, Estatística, Dados

Sumário

1
Introdução

1.1
Motivação

No contexto da sociedade moderna, a importância dos dados é indiscutível. A popularização de conceitos como *big data* e o apoio à decisão baseado em dados teve como consequência o intenso crescimento da área de ciência de dados nos últimos anos, tanto na academia quanto na indústria.

Nesse novo cenário, uma ferramenta que se destacou por sua utilidade universal foi a da classificação estatística. Para entendê-la, podemos olhar para os seguintes exemplos de problemas de classificação:

- Um banco que deve decidir se cede ou não crédito para um cliente a partir de suas variáveis de cadastro;
- Um hospital que deseja determinar um diagnóstico de um paciente recém-chegado, a partir de seus dados médicos e de outros pacientes;
- Uma empresa que deseja identificar o perfil de compras de um novo cliente, usando os registros das suas compras passadas e variáveis de cadastro.

Examinando o caso do hospital em particular, temos como propósito usar características pessoais do paciente (como sexo, idade, pressão sanguínea, entre outros) para associá-lo a algum diagnóstico. Embora um profissional médico poderá fazer isso caso venha a atender o paciente individualmente, seria interessante que a condição do paciente pudesse ser identificada o mais rápido possível de forma automatizada, especialmente para casos mais habituais.

Agora, suponha a existência de um histórico de pacientes do hospital. Nele, estão listados pacientes passados e atuais, suas características pessoais e o diagnóstico que tiveram quando chegaram ao hospital. Se as relações entre as características listadas e os diagnósticos pudessem ser quantificadas a partir desse histórico, haveria uma maneira de diagnosticar novos pacientes apenas a partir dessas mesmas características.

É exatamente essa quantificação das relações entre propriedades presentes em bases de dados e um conjunto de possíveis saídas que define a ferramenta da classificação estatística. Métodos de classificação geram uma função

que mapeia uma entrada qualquer em uma saída, a partir de padrões extraídos de um conjunto de dados, o qual por sua vez tem pares entrada-saída definidos a priori.

Observe que esse tipo de classificação estatística é um caso de aprendizagem supervisionada, já que gera um mapeamento a partir de padrões pré-definidos (no conjunto de dados usado para treinamento do classificador). Diferencia-se, então, da classificação não-supervisionada, que tem como propósito a geração e descobrimento de grupos homogêneos de objetos a partir de um conjunto de dados inicial, como na análise de agrupamento (*cluster analysis*).

É possível citar diversos casos reais de aplicação da classificação supervisionada explicada acima: zoneamento climático da China visando auxiliar medidas de planejamento público (Yang et al. 2020), identificação de tumores cerebrais (Ismael et al. 2018) e avaliação de crédito por entidades financeiras (Louzada et al. 2016), entre outros.

Dessa forma, é difícil negar a relevância do estudo de métodos de classificação no momento atual da sociedade. Por servirem como uma maneira de generalizar e aplicar padrões (às vezes imperceptíveis ao olho nu) presentes em grandes conjuntos de dados, esses métodos representam não apenas uma invenção necessária para o contexto atual da indústria como uma ferramenta de imenso potencial para o desenvolvimento humano no futuro.

É necessário, porém, destacar: não existe almoço grátis no assunto de métodos de aprendizado (Wolpert 1996). Apesar da promessa e utilidade de técnicas de classificação, é imprescindível o estudo e análise dos diferentes métodos existentes para a aplicação apropriada destes em diferentes problemas; pois não existe um único método adequado para todos os casos.

1.2 Objetivos do trabalho

Esse projeto busca realizar um estudo de diferentes métodos na área de classificação estatística, visando explorar os fundamentos por trás dessas técnicas e suas diversas aplicações no contexto moderno. Visando abordar tanto métodos tradicionais quanto representantes do estado da arte da área de classificação, são apresentados neste projeto os seguintes métodos:

- Regressão logística
- Análise linear discriminante (LDA)
- *K-nearest neighbours* (KNN)
- Árvores de decisão (*classification tree*)
- *Random forests*

- *Gradient boosting*
- *Support-vector machines* (SVM)

A estrutura do projeto se divide em duas partes principais: uma análise dos conceitos por trás de cada método e uma aplicação prática via geração de modelos sobre três bases de dados. Na primeira parte, deseja-se entender a maneira como cada método funciona: qual é a ideia por trás de seu funcionamento, como é estimado e como de fato faz a classificação de novos dados. Na segunda parte, deseja-se realizar um experimento sobre dados que sirva como uma complementação prática da discussão de caráter teórica feita anteriormente.

Visando abordar tanto o caso de classificação binária quanto o multinomial, foram selecionadas três bases de dados: uma referente à identificação de falência de empresas baseando-se em indicativos financeiros, uma relacionada à classificação da nota final de alunos baseando-se em fatores acadêmicos e sociais e uma terceira relacionada ao diagnóstico de COVID baseando-se em sintomas relatados. Com a criação de modelos sobre cada uma dessas bases, foi possível avaliar a performance e resultados gerados pelos métodos estudados no projeto.

Finalmente, foram apresentados ao longo das seções deste documento diversos conceitos úteis para o estudo da área de classificação de forma geral - como por exemplo o uso de diferentes métricas para validação de modelos e pré-processamento de dados para uso com métodos de aprendizado.

2
Definições iniciais

2.1
Classificadores

Como descrito no capítulo 1, um classificador pode ser definido de maneira simples como uma ferramenta que atrela uma classe/categoria à uma observação de dados. Agora, buscaremos formalizar essa definição.

Suponha um conjunto de observações O na forma de pares entrada-saída, descrito por $O = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, onde $X_i = (x_{i1}, \ldots, x_{in})$. Nessa definição, X_i representa a entrada, na forma de um conjunto de valores para as variáveis independentes. Já Y_i a variável dependente, representa a saída - que no caso, é uma classe/categoria entre k possibilidades mutuamente exclusivas e exaustivas.

Estimamos uma função objetivo $\hat{f}$ a partir de um subconjunto das observações (chamado "conjunto de treino") em O. Essa função tem como propósito retornar uma classe do conjunto de saída a partir de uma dada observação das variáveis independentes, sendo descrita pela seguinte equação:

$$\hat{Y'} = \hat{f}(X')$$

Onde X' é uma observação qualquer das variáveis independentes, e Y' a classe estimada para essa observação. A função $\hat{f}(X')$ descreve o nosso classificador. Para estimarmos a acurácia da classificação, podemos realizar o seguinte cálculo:

$$ERR = \frac{1}{n} \sum_{i=1}^{n} \mathrm{I}(Y_i \neq \hat{Y}_i)$$

Onde n é o número de observações do conjunto de treino, $\hat{Y}_i$ é a saída da função $\hat{f}$ para a observação x_i - isso é, a classe retornada pela função objetivo estimada do classificador. Finalmente, a função I retorna 0 quando a classificação foi correta ($Y_i = \hat{Y}_i$) e 1 caso contrário. Dessa forma, ERR será igual à fração de classificações incorretas em relação à quantidade total de observações no conjunto de treino.

Para que um classificador apresente boa acurácia, ele deve minimizar o número de classificações incorretas para as observações do conjunto de teste - não consideradas durante a estimação de $\hat{f}$ - não apenas para as observações do conjunto de treino.

2.2 Um classificador ótimo

Tendo estabelecido que minimizar o erro de um classificador quanto a um conjunto de teste é imprescindível para o desenvolvimento de um bom método de classificação, seria ideal tentar estabelecer um classificador que alcance o menor valor possível de erro desse teste.

Suponha que fosse criado um classificador cujo método consistisse em categorizar uma observação qualquer na classe mais provável dela estar, a partir dos valores das suas variáveis independentes. Ou seja, escolher a classe j que maximize $\Pr(Y = j \mid X = x_0)$ para um conjunto de valores x_0. Essa é a descrição do classificador de Bayes - e é possível provar que ele minimiza o valor da medida de erro para um conjunto de teste. Ele pode ser escrito como

$$C^{\text{Bayes}}(x) = \underset{r \in \{1,2,\ldots,K\}}{\operatorname{argmax}} \Pr(Y = r \mid X = x).$$

Apesar do classificador de Bayes ser, por definição, o classificador ótimo (no sentido de que ele minimiza classificações incorretas na amostra teste), ele é impossível de ser computado em casos reais. A razão disso se dá pelo fato de sua definição depender de algo que é praticamente impossível de se obter: a distribuição condicional de Y (uma classe) dado X (uma observação).

Mesmo assim, o classificador de Bayes tem importância por definir uma base para alguns métodos de classificação, que sanam a falta da função de distribuição assumindo hipóteses sobre a sua forma. Além disso, o classificador de Bayes serve como um benchmark teórico para qualquer classificador. Isso porque a taxa de erro de Bayes (*Bayes error rate*) é a menor taxa de erro possível dentro um conjunto de teste para um classificador, dada por:

$$1 - E\left(\max_j \Pr(Y = j \mid X)\right)$$

Algo que é importante notar é que o classificador de Bayes minimiza o erro total de todos os classificadores: ele retorna o menor número possível de observações classificadas erroneamente, independentemente de classe. Isso nem sempre é o retorno ideal de um classificador - dependendo da situação, pode ser útil ter uma taxa de erro total maior para se garantir maior acurácia na

identificação de determinada classe em novas observações, o que será analisado posteriormente.

Assim, o classificador de Bayes representa um ideal que não é possível de ser obtido na prática. Somando-se isso ao fato de que o objetivo de um processo de classificação nem sempre é minimizar o erro total, reitera-se a importância do estudo e comparação de diferentes métodos de classificação - visando o estabelecimento de um conjunto de classificadores que possam ser usados de maneira intercalada e de acordo com a necessidade dos problemas enfrentados.

3
Métodos

Neste capítulo, iremos descrever conceitualmente cada um dos métodos de classificação selecionados para estudo, explicitando seus funcionamentos e introduzindo características importantes de cada um.

3.1
Regressão logística

Um dos métodos mais básicos em toda a área do aprendizado estatístico é o da regressão linear estimada por mínimos quadrados ordinários (MQO), onde a variável resposta Y só pode assumir 2 categorias. Nesse modelo, a probabilidade de ocorrência de Y é uma função linear de um conjunto de variáveis preditoras X.

No âmbito da classificação, porém, o uso do modelo de probabilidade linear em sua forma mais comum não é apropriado (James et al. 2013) - já que pode gerar probabilidades maiores que 1 ou menores que 0, além de violar as hipóteses de normalidade e homocedasticidade do erro usadas para a estimação de parâmetros via MQO.

Para aplicarmos os conceitos de regressão em problemas de classificação utiliza-se então a regressão logística, que modela a *probabilidade* de uma observação pertencer à uma determinada categoria através de uma função que garante a estimativa da probabilidade entre 0 e 1.

3.1.1
Caso binomial

A equação de um modelo de regressão logística com p regressores e duas classes (C1, C2) possíveis é dada por:

$$\Pr(Y = C1|X) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{1 + e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}} = \frac{1}{1 + e^{-(\beta_0+\beta_1 X_1+\cdots+\beta_p X_p)}}$$

Onde $\Pr(Y = C1|X)$ é a probabilidade da entrada $X = (X_1, ..., X_p)$ ter como saída a categoria C1 e $\beta = (\beta_0, \beta_1, \ldots, \beta_p)$ é o vetor de parâmetros do modelo; sendo β_0 o intercepto e β_i o parâmetro associado ao preditor X_i.

Sendo a variável de saída dicotômica, podemos estabelecer um valor de *cut off*, dado por c, que transformará a probabilidade dada pela equação acima em uma saída qualitativa. Caso $\Pr(X) \geq c$, categorizamos X com saída $C1$; caso $\Pr(X) < c$, categorizamos X com saída $C2$. Finalmente, esse modelo é estimado pela maximização da função de log-verossimilhança, através de algoritmos de otimização para funções não-lineares.

3.1.2
Regressão logística multinomial

Caso queiramos modelar uma situação na qual há mais de dois valores de saídas, precisamos expandir o cálculo da probabilidade para considerar isso.

Supondo a existência de K classes, estabelecemos a K-ésima classe como o nível basilar. O modelo da regressão logística multinomial descreve as seguintes equações:

$$\Pr(Y = k|X) = \frac{e^{\beta_{k_0} + \beta_{k_1} X_1 + \cdots + \beta_{k_p} X_p}}{1 + \sum_{l=1}^{K-1} e^{\beta_{l_0} + \beta_{l_1} X_1 + \cdots + \beta_{l_p} X_p}}$$

para as classes $k = 1, ..., K-1$ e

$$\Pr(Y = k|X) = \frac{1}{1 + \sum_{l=1}^{K-1} e^{\beta_{l_0} + \beta_{l_1} X_1 + \cdots + \beta_{l_p} X_p}}$$

para a classe K. É importante notar que a escolha de uma classe como nível basilar não altera os valores de previsão de modelos resultantes, mas sim a grandeza e o sinal de cada um de seus parâmetros estimados.

A estimação e previsão a partir do modelo é feita da mesma maneira que na regressão logística binomial: obtêm-se os valores dos parâmetros pelo método de máxima verossimilhança e calcula-se a probabilidade de uma dada entrada X pertencer à cada uma das classes. Define-se uma categoria prevista para cada entrada a partir da identificação da classe k que gera o maior valor de $\Pr(Y = k|X)$.

3.2
Análise linear discriminante (LDA)

De forma semelhante à regressão logística, o método de análise linear discriminante (conhecido como LDA, proveniente do inglês *linear discriminant analysis*) busca encontrar $\Pr(Y|X)$, isso é, a probabilidade de Y pertencer a uma certa classe dada as características X.

Enquanto esse valor é calculado de forma direta no método de regressão logística, LDA usa o Teorema de Bayes para obtê-lo a partir de hipóteses sobre as distribuições das classes de saída (Hastie et al. 2001).

Para explicitar o funcionamento do método LDA, primeiro usamos o teorema de Bayes para escrever a seguinte equação para $\Pr(Y|X)$:

$$\Pr(Y = k|X = x) = \frac{\Pr(Y = k)\Pr(X = x|Y = k)}{\Pr(Y = i)\sum_{i=1}^{K}\Pr(X = x|Y = i)}$$

Note que a equação acima estabelece que podemos calcular a probabilidade de uma determinada entrada X de dados pertencer à classe k a partir dos termos $\Pr(Y = k)$ e $\Pr(X = x|Y = k)$.

O primeiro destes termos, $\Pr(Y = k)$, é a probabilidade de que uma observação qualquer pertença à classe k (ou seja, tenha como saída a categoria k); chamaremos este de π_k.

O segundo termo, $\Pr(X = x|Y = k)$, pode ser compreendido como a função de densidade de X para uma observação da classe k; retornaria valor alto caso haja alta probabilidade de uma observação na classe k ter $X \approx x$ e um valor baixo se isso for improvável. Chamaremos este de $f_k(x)$.

Note que ambos esses termos dependem de informações difíceis de se obterem na realidade: uma é associada à distribuição de dados entre as diferentes classes de saída possíveis (conjunto Y) e outra depende de conhecermos a distribuição de observações pertences à cada classe do conjunto Y.

Para resolver isso, o método LDA assume hipóteses que permitem o cálculo de $\Pr(Y = k|X = x)$ pelo teorema de Bayes. A primeira é que π_k é igual à fração de observações no conjunto de dados de treinamento que pertencem à classe k. A segunda é que cada $f_k(x)$ é equivalente à uma distribuição multivariada Gaussiana, com uma média μ_k específica de cada classe e uma matriz de covariância Σ comum para todas - ou seja, $X \sim (\mu_k, \Sigma)$.

Assim, cada $f_k(x)$ é dada por:

$$f_k(x) = \frac{1}{\sqrt{(2\pi)^p|\Sigma|}}\exp(-\frac{1}{2}(x - \mu_k)^T\Sigma^{-1}(x - \mu_k)$$

Onde p é o número de componentes do vetor de entrada X (ou seja, o número de preditores do sistema). Quando inserimos essa equação no teorema de Bayes mostrado anteriormente, chegamos na seguinte equação:

$$\Pr(Y = k|X = x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log\pi_k$$

Note que após a estimação dos parâmetros desconhecidos $\mu_1, \ldots, \mu_k$, $\pi_1, \ldots, \pi_k$ e Σ, a equação acima descreve o que desejamos: a probabilidade de uma entrada de dados X pertencer à uma classe Y. Assim, tendo esses

parâmetros estimados, o método LDA então usa essa equação para classificar novas entradas, selecionando sempre a classe de saída cujo valor da equação acima corresponde ao maior dentre todas as classes de Y.

Note que a equação definitiva do modelo é linear em função de x, fato que caracteriza o método LDA. Uma versão mais geral deste método chamado análise quadrática discriminante (QDA) pode ser usada para modelar relações não-lineares de maneira análoga à como o LDA modela relações lineares, fazendo hipóteses diferentes sobre os dados sendo modelados (James et al. 2013).

3.3 k-Nearest Neighbours (kNN)

Nas seções anteriores, descrevemos métodos de classificação paramétricos. Estes estabelecem hipóteses sobre a forma da função f que descreve um modelo - por exemplo, de que é uma função linear ou quadrática. Essa rotina, por razões claras, tende a gerar bons resultados quando a relação verdadeira entre a variável dependente e os regressores possui a forma assumida pelo modelo; o que nem sempre é verdade.

Nessas situações, é interessante considerar métodos não-paramétricos. Esses, assim como os modelos paramétricos, estimam a função f de maneira que fique o mais próxima possível de todos os pontos de treino disponíveis, porém com a distinção de não assumirem uma forma para f a priori. Dessa forma, métodos não paramétricos costumam ser mais flexíveis, capazes de modelarem dados com forma menos estruturada - porém necessitando de mais observações para a geração de uma modelagem acurada, além de, em alguns casos, oferecerem pior interpretabilidade dos resultados (James et al. 2013).

Um exemplo comum de método não paramétrico de classificação é o *k-Nearest Neighbours* (kNN), isso é, k-vizinhos mais próximos.

Dado um inteiro k definido a priori e uma observação de teste x, o classificador kNN identifica os k pontos de treino que estejam mais próximos de x, gerando um conjunto C_k de pontos. A probabilidade condicional de x pertencer à uma determinada classe j é calculada com base na proporção de pontos em C_k cuja classe é j, como exibido na seguinte equação:

$$\Pr(Y = j|X = x) = \frac{1}{K} \sum_{C_k} I(y_i = j)$$

O método então classifica a entrada x como pertencendo à categoria cuja probabilidade condicional foi a maior dentro do conjunto C_k previamente estabelecido. Na Figura 3.1, podemos ver exemplos de predição usando kNN com valores de k diferentes.

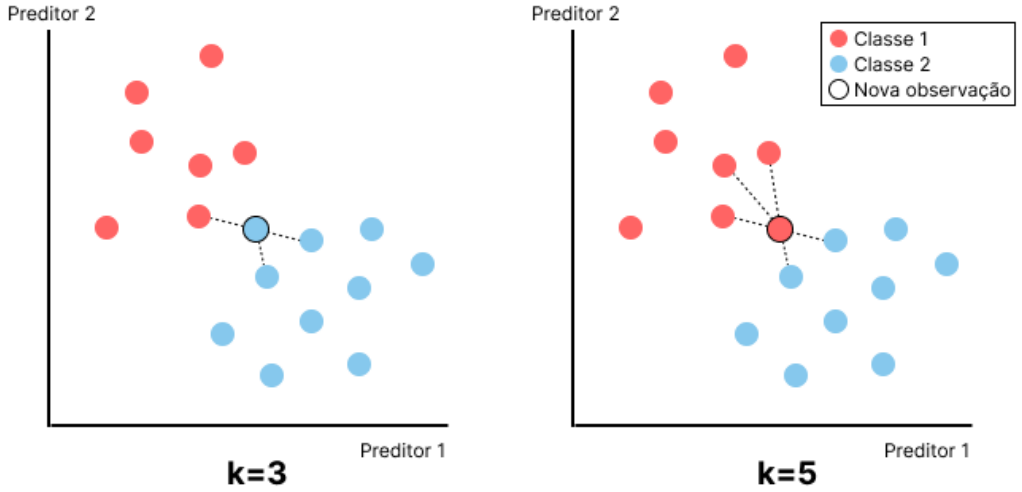

Figura 3.1: Dois casos de predição de uma nova observação usando modelo kNN. Fonte: elaboração própria.

Apesar (ou talvez por causa) de sua simplicidade, o kNN possui alguns problemas. Em primeiro lugar, o sucesso da classificação feita pelo método é extremamente dependente no valor escolhido para k. Quanto maior o valor de k, maior o viés do modelo resultante e menor a variância; quanto menor o valor de k, menor o viés e maior a variância.

Além disso, por ser um exemplo de aprendizado preguiçoso que precisa executar seu algoritmo para cada nova observação a ser classificada, aplicar o kNN em bases de dados largas pode ser custoso. Essas desvantagens podem, porém, ser minimizadas a partir de técnicas como a seleção automática do k e a indexação de observações de treino.

Ainda assim, o kNN em sua forma padrão não é considerado um bom método para uso em tarefas de classificação ou bases de dados mais complexas (Cunningham et al. 2007). Apesar disso, continua sendo importante como um método de simples implementação e interpretabilidade; e por gerar um algoritmo intuitivo e generalizável, pode ser usado como uma etapa intermediária para a criação de modelos mais complexos (Guo et al. 2003).

3.4 Árvores de classificação

Seja $X = \{X_1, X_2, \ldots, X_p\}$ um vetor dos preditores de um problema de classificação. Temos que o conjunto de todos os possíveis valores de X forma um espaço de dimensão p chamado de espaço de decisão ou espaço preditivo do problema.

Agora, suponha que esse espaço possa ser dividido em um conjunto

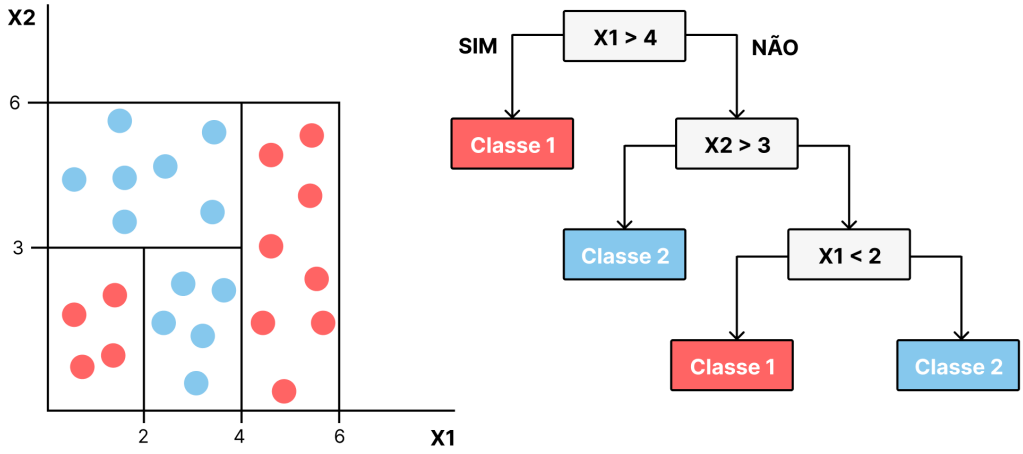

Figura 3.2: Exemplo de árvore de decisão para uma base com dois preditores e classificação binária. Note a separação do espaço preditivo em regiões de acordo com as partições feitas na árvore. Fonte: elaboração própria.

R de regiões mutualmente exclusivas e que não se sobreponham, com cada região sendo assinalada uma categoria de saída j do conjunto Y baseando-se nas observações de treinamento distribuídas pelo espaço de decisão. Novas observações podem ser classificadas de acordo com sua posição no espaço e consequente região R_i.

Essa é a ideia por trás dos modelos estatísticos de árvores de classificação. Essa divisão do espaço em regiões é comumente representada por uma estrutura composta por nós - cuja forma resultante se assemelha à uma árvore, nomeando então esse tipo de modelo.

A divisão do espaço preditivo do problema é feita por um algoritmo de particionamento recursivo (*recursive partitioning*), pelo qual o espaço é inicialmente dividido em duas regiões iniciais. O algoritmo é aplicado novamente sobre cada uma das regiões geradas, e assim sucessivamente até que algum critério de parada determinado *a priori* seja alcançado, finalizando então a partição.

Para efetuar a divisão de cada região, é necessário estabelecer um critério de partição, a fim de determinar a divisão ótima de cada espaço. Embora usar a taxa de erro de classificação das amostras de treino dentro de cada região que seria criada por uma dada partição seja uma opção natural, na prática ela não é uma medida suficientemente sensível para garantir o crescimento de uma árvore adequada para classificar novas observações (James et al. 2013).

Embora haja algumas opções válidas de critério de partição, um comumente usado é o índice de Gini, que indica a "pureza" de uma região - isso é, se ela possui predominantemente observações de uma única classe - e é dado

pela seguinte equação:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

Onde $\hat{p}_{mk}$ denota a proporção de observações de teste em uma região m que pertencem à classe k. Note que se um conjunto de possíveis regiões formadas por uma partição do espaço separar bem amostras de classes distintas, teremos valores de $\hat{p}_{mk}$ próximas de 0 e 1 - o que por sua vez gerará um valor baixo para o índice de Gini.

Consequentemente, podemos usar esse índice para verificar em cada iteração do algoritmo de partição recursiva se uma divisão do espaço é boa ou não verificando se ela gera regiões/nós mais puros (com um menor valor do índice de Gini) do que outras partições candidatas.

Com isso, é simples definir o algoritmo de particionamento recursivo: a cada iteração, busca-se a variável independente e o valor de corte (no domínio dessa variável) que gera as regiões mais puras possíveis (a partir, por exemplo, do índice de Gini). Esse processo é repetido, gerando mais regiões/nós e aumentando o tamanho da árvore, até que seja atingido algum critério; por exemplo, até que todas as regiões tenham menos de uma quantidade n de observações.

O algoritmo de particionamento recursivo por si só já gera uma árvore de classificação que pode ser usada para previsão de novas classificações. Apesar disso, o algoritmo de particionamento recursivo possui variância alta; alterações pequenas no conjunto de dados usado para treinamento tendem a gerar resultados muito diferentes.

Isso, por sua vez, faz com que as árvores geradas sofram de *overfitting*, com bons resultados no conjunto de treino mas baixas taxas de acerto para amostras *out-of-sample* (Buntine 1991).

Assim, é necessário aplicar o processo de poda ("*pruning*"), pelo qual determinadas partições feitas na árvore original são desconsideradas - gerando regiões maiores do que as anteriores e possivelmente diminuindo os acertos no conjunto de treino, mas melhorando significativamente a taxa de classificação para novas observações de dados. Um algoritmo de poda comumente usado é o *cost-complexity pruning* (Breiman et al. 1983), no qual o objetivo é minimizar uma combinação linear do erro de classificação e da complexidade da árvore (definida pelo seu número de nós, ou seja, seu tamanho).

Após o processo de crescimento e poda de uma árvore, ela pode ser usada para classificação de novas observações. Como comentado anteriormente, a categoria assinalada à uma determinada observação será equivalente à classe majoritária da região dentro da qual ela está contida.

Assim como o kNN, porém, a capacidade preditiva de um modelo de árvore em sua forma básica tende a ser pior que a de outros métodos usados para classificação (James et al. 2013). Por essa razão, as principais vantagens associadas ao uso de árvores de classificação são a simplicidade de seu funcionamento e de interpretação de seus resultados, principalmente devido à capacidade de visualização gráfica do modelo.

3.5 Random forests

Como observado na seção anterior, uma única árvore de classificação tende a ter sua performance preditiva limitada, sendo um modelo de variância alta e, consequentemente, com tendência de *overfitting*. Uma maneira de minimizar o problema da variância alta de um experimento é realizando-o múltiplas vezes e fazendo uma média dos resultados. Quando essa ideia é aplicada no contexto de árvores de classificação, temos o processo conhecido como *bagging* (Breiman 1996).

A ideia do *bagging* se baseia primeiramente em gerar B amostras do conjunto de dados de treino original via *boostrapping*, processo que pode ser compreendido como amostragem aleatória com reposição. Gera-se então B árvores de classificação - uma para cada amostra gerada do conjunto de dados original. Finalmente, faz-se a média de todos esses modelos para gerar um único modelo de baixa variância e performance significativamente melhorada (James et al. 2013).

No contexto de classificação, por exemplo, um modelo de "bagged trees" gerado dessa forma assinalará uma categoria para uma nova observação dos dados primeiro verificando qual a classe prevista por cada árvore individual para aquela observação; a classe mais frequente entre todas as previsões do conjunto de *bagged trees* será a assinalada para a observação no final do processo.

O *bagging*, porém, não é sem suas limitações. É possível demonstrar que a variância de um modelo de *bagged trees* é dada pela seguinte equação:

$$\sigma_B^2 = \frac{1-\rho}{B}\sigma^2 + \rho\sigma^2$$

Onde ρ é uma variável que mede o quão correlacionadas as B árvores do modelo são e σ^2 é a variância média das árvores.

Note que como prevemos, na medida que B cresce - isso é, usamos mais árvores no processo de *bagging*, menor é o primeiro termo de σ_B^2. Olhando o segundo termo, porém, vemos que $\sigma_B^2 \approx \sigma^2$ caso as árvores sejam

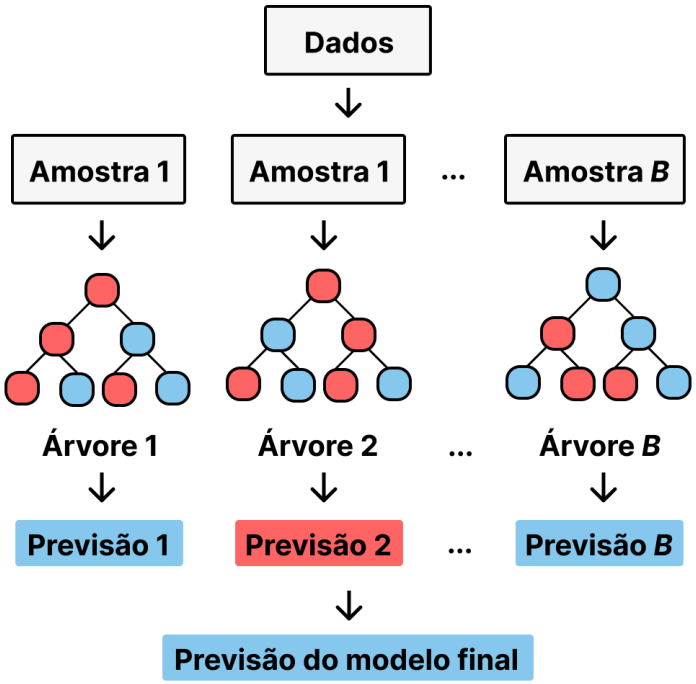

Figura 3.3: Ilustração do processo de *bagging* (genérico, com árvores de classificação). Fonte: elaboração própria.

correlacionadas ($\rho \approx 1$).

O caso em que a correlação entre as árvores no *bagging* é alto não é muito raro: caso exista um preditor (ou conjunto de preditores) que faça uma boa separação das classes, é altamente provável que muitas - senão todas - as B árvores geradas usem esse(s) mesmo(s) preditor(es) para gerarem as partições do espaço preditivo. Caso isso aconteça, o *bagging* não representará um grande avanço sobre o modelo único da árvore, já que a média de muitas árvores de alta variância (por serem similares entre si) ainda será uma árvore de alta variância.

Se queremos fazer com que outros preditores (fora desse conjunto de preditores fortes) sejam usados pelas árvores - e, portanto, que elas sejam descorrelatas entre si - podemos adicionar uma nova regra ao processo do *bagging*: cada iteração do algoritmo de particionamento recursivo só poderá considerar um subconjunto de m preditores como candidatos para partição.

Essa alteração, apesar de simples, possui um efeito considerável por causar um aumento da descorrelação entre as árvores de classificação individuais que são agregadas para a formação do modelo final - o qual é chamado de modelo de *random forest*. O valor tipicamente escolhido para m é $\sqrt{p}$, onde p é o número total de preditores na base de dados.

Em conclusão, modelos *random forest* minimizam o problema da alta variância de uma única árvore de classificação pela agregação de várias árvores

- as quais são treinadas em amostras de dados diferentes e forçadas a escolher entre subconjuntos aleatórios dos preditores para gerar suas partições. O resultado é um modelo que possui capacidade preditiva superior ao de uma árvore individual e maior robustez à alteração nos dados.

Vale destacar que um modelo *random forest* sacrifica uma das vantagens do modelo de árvore original para obter melhores resultados: a interpretabilidade. Embora existam métricas que permitam o usuário avaliar quais variáveis foram mais importantes para o modelo (normalmente verificando quais causam uma maior variação do índice de Gini para as árvores individuais), perde-se a visualização gráfica estrutural simples que existe no modelo de árvore única.

3.6 Gradient boosting

Na seção 3.4, observamos que árvores de classificação são modelos de alta variância, com tendência ao *overfitting* na medida que crescem de tamanho. Na seção 3.5, apresentamos o modelo *random forest*, que serve como uma solução para esse problema: ao fazermos a média de várias árvores treinadas para aprender de forma extensiva as características de amostras diferentes, criamos um novo modelo que possui capacidade preditiva boa para a amostra como um todo, minimizando o problema da variância de suas partes individuais.

Existe, porém, outra maneira de combinar várias árvores para a criação de um modelo único. Ao invés de criarmos várias grandes árvores (de alta variância) e posteriormente gerarmos uma união delas com o intuito de diminuir a variância do todo, podemos criar pequenas árvores (de alto viés) e adicionarmos seus resultados sequencialmente, tendo no fim do processo um modelo de excelente capacidade preditiva - processo o qual chamamos de *gradient boosting* (Friedman 2001).

Intuitivamente, a ideia de modelos de *boosting* é aprender de forma "lenta". Quando criamos uma árvore de classificação de profundidade limitada, temos que ela será um modelo de alto viés e baixa variância, pois dificilmente capturará bem as características da amostra sobre a qual ela é treinada. Apesar disso, se conseguirmos criar outra árvore que capture pelo menos parte das características dos dados cuja primeira árvore *não* capturou e a unirmos com a primeira, e repetirmos esse processo sequencialmente, teremos no final um modelo feito de árvores "pequenas" que, quando unidas, terão capturado a amostra como um todo e serão capazes de classificar novas amostras.

Formalizando de maneira genérica essa intuição, podemos dizer que a

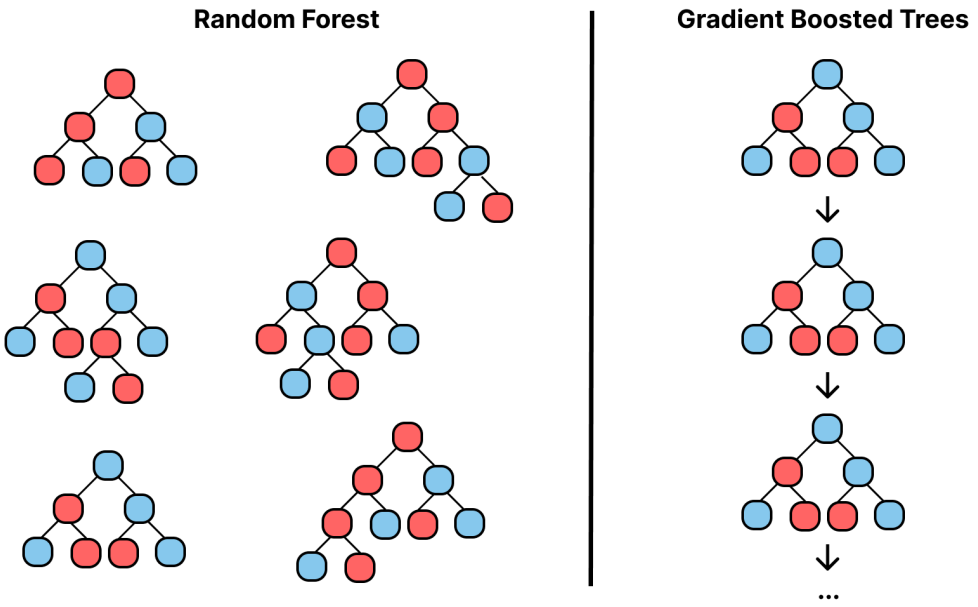

Figura 3.4: Ilustrações dos métodos baseados em árvore. Embora ambos se baseiem na ideia da geração de várias árvores para a criação de um modelo agregado, o *gradient boosting* assume uma sequencialidade nessa geração. Fonte: elaboração própria.

função do modelo $F_m(x)$ na m-ésima iteração será:

$$F_m(x) = F_{m-1}(x) + \lambda h_m(x)$$

Onde $\lambda \in [0, 1]$ é um parâmetro de escala, que será usado para determinar o peso dado para cada nova iteração do algoritmo, e $h_m(x)$ é a função que representa a árvore gerada na m-ésima iteração. O parâmetro m, representando o número máximo de iterações, comumente tem um valor definido *a priori*; com o algoritmo podendo acabar em menos iterações caso alguma $F_i(x)$ não represente uma melhoria sobre a $F_{i-1}(x)$ para $0 < i \leq m$.

Para explicarmos o resto do método, é necessário definir o que é uma função de perda (*loss function*). O propósito de uma função de perda $L(y, \hat{y})$ é medir a distância entre um valor observado y e uma previsão $\hat{y}$; na medida que o valor de $L(y, \hat{y})$ decresce, podemos afirmar que a previsão está se aproximando do valor observado y.

Por essa razão, o *gradient boosting* baseia seu funcionamento na minimização de uma função de perda - mais especificamente, na aplicação do método do gradiente sobre L (Hastie et al. 2001). A cada iteração, calculamos os

pseudo-resíduos r_{im} do modelo F_{m-1}, para cada $(x_i, y_i) \in (X, Y), i = n$:

$$r_{im} = \left[\frac{\partial L(Y, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)} \right]$$

Note que chamamos $r_m = \{r_{1m}, \ldots, r_{nm}\}$ (onde n é igual ao número de amostras de treino na base de dados) de pseudo-resíduos porque eles representam o vetor que aponta para a minimização da função de perda, mas o que naturalmente entenderíamos como os resíduos de um modelo (a diferença direta entre valores observados e previstos).

Para que a proposta de adição de métodos sequencial funcione, a subárvore de cada iteração será então treinada não sobre os pares entrada-saída (x_i, y_i), mas sim sobre (x_i, r_{im}) - sendo justamente essa configuração que fará com que cada subárvore construída aprenda os dados de maneira a complementar o aprendizado das árvores anteriores.

Ao treinar árvores sobre o gradiente da função de perda (atualizado continuamente), o modelo de *gradient boosting* se aproxima iterativamente de um mínimo local de L - e portanto, de um modelo que se aproxima dos dados observados.

Em que *aspecto* o modelo se aproxima (por exemplo, em termos de acurácia preditiva ou de treino) dependerá da função de perda escolhida. No âmbito da classificação, a função escolhida comumente é o desvio multinomial (Hastie et al. 2001), dada pela seguinte equação:

$$L(y, p(x)) = -\sum^{K} I(y = k) \log p_k(x)$$

Onde K é o conjunto de todas as classes de saída e $p_k(x)$ é a probabilidade de uma observação x ser da classe k.

3.7 Support-vector machines (SVM)

O último método de classificação a ser abordado neste projeto é obtido a partir da generalização de um método mais simples: o classificador de margem máxima (*maximal margin classifier*).

Suponha uma base de dados composta por um conjunto X de n observações, cada uma com p preditores. Uma amostra qualquer $x_i = \{x_{i1}, \ldots, x_{ip}\}$ pertence à uma categoria $y_i \in \{-1, +1\}$. Agora, suponha que as classes são *linearmente separáveis*: isso é, podemos traçar pelo menos um hiperplano (uma linha, caso $p = 2$) no espaço p-dimensional que contém as observações de

tal forma que todas as observações fiquem em apenas uma das duas regiões resultantes da divisão pelo hiperplano.

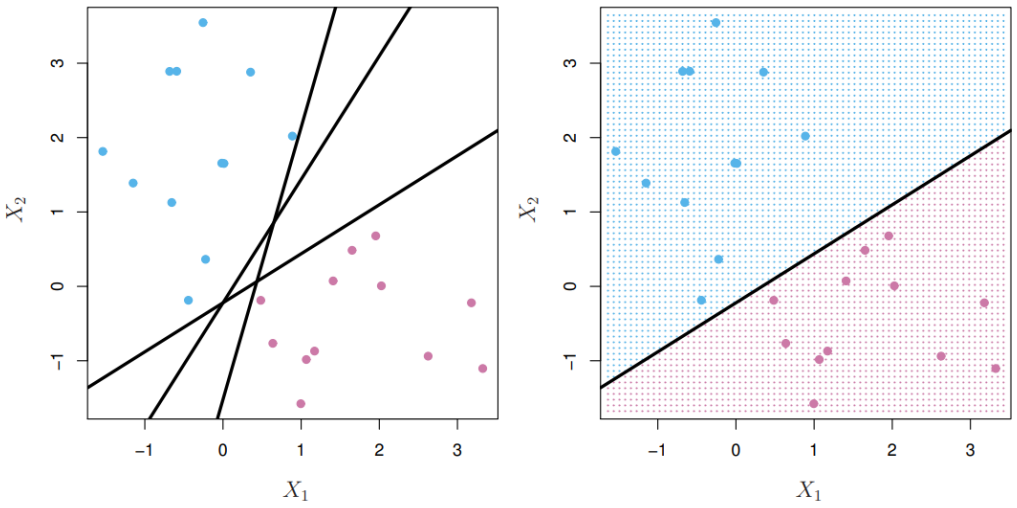

Figura 3.5: Visualização de dados linearmente separáveis. Na imagem esquerda, as várias linhas que podem separar as observações de classes diferentes. Na direita, as regiões geradas pela separação de uma das linhas. Fonte: *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013.

Se esse for o caso para uma base de dados qualquer, temos que qualquer um dos hiperplanos que separa as classes serve como a base para um classificador - podemos assinalar à uma nova observação a classe da região na qual ela se encontra no plano, da mesma forma que fizemos com árvores de classificação. Isso está representado visualmente na Figura 3.5 para o caso $p = 2$, onde cada hiperplano é uma linha que divide o plano bidimensional.

Como é possível ver na figura, porém, existem infinitos hiperplanos que separam as duas classes quando essas são linearmente separáveis. Dessa forma, é necessário definir algum critério para a escolha do hiperplano ótimo. Para o *maximal margin classifier*, a escolha é relativamente natural: devemos usar o hiperplano cuja distância até as observações de cada classe seja a maior possível - ou seja, o hiperplano de margem máxima.

O hiperplano de margem máxima M pode ser encontrado pela resolução de um problema de otimização relativamente simples (James et al. 2013), representado abaixo:

$$\max_{\beta_0,\beta_1,\ldots,\beta_p,M} M \tag{3-1}$$

$$\text{s.a.} \sum_{j=1}^{p} \beta_j^2 = 1 \tag{3-2}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq M \quad \forall i = 1, \ldots, n \tag{3-3}$$

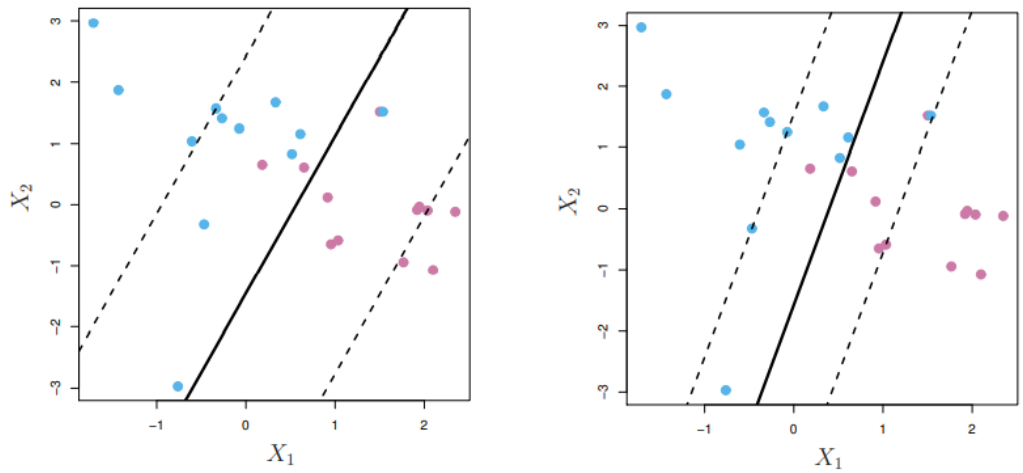

Figura 3.6: Dois exemplos de classificadores do tipo *support vector classifier* criados sobre o mesmo conjunto de dados porém com valores do parâmetro C diferentes - sendo o valor do classificador da esquerda maior que o da direita. Note que em ambos os casos algumas classificações estão no lado incorreto do hiperplano. Fonte: *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013.

Onde os parâmetros $\{\beta_1, \ldots, \beta_p\}$ são os coeficientes do hiperplano.

Infelizmente, é raro encontrar bases de dados reais nos quais as classes de interesse são linearmente separáveis, o que limita o poder de classificadores desse tipo. A solução natural é afrouxar as restrições do problema, permitindo que pelo menos algumas observações de outra classe fiquem nas regiões "incorretas" do hiperplano. Dessa forma, agora queremos encontrar o hiperplano que melhor separe *a maioria* das observações, podendo classificar incorretamente algumas. O novo problema de otimização é dado por:

$$\max_{\beta_0,\beta_1,\ldots,\beta_p,M} M \tag{3-4}$$

$$\text{s.a.} \sum_{j=1}^{p} \beta_j^2 = 1 \tag{3-5}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geq M(1 - \epsilon_i) \quad \forall i = 1, \ldots, n \tag{3-6}$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_1 \leq C \tag{3-7}$$

Onde C é um parâmetro não-negativo e $\epsilon_1, \ldots, \epsilon_n$ são variáveis de folga que permitirão que a modelagem considere classificações "incorretas". Mais especificamente, uma variável ϵ_i representará a posição da observação x_i em relação ao hiperplano: caso $\epsilon_i = 0$, a observação está no lado correto do hiperplano; caso $epsilon_i > 0$, a observação está entre a margem e o hiperplano; caso $epsilon_i > 1$, a observação está no lado errado do hiperplano [NOTA: Cris,

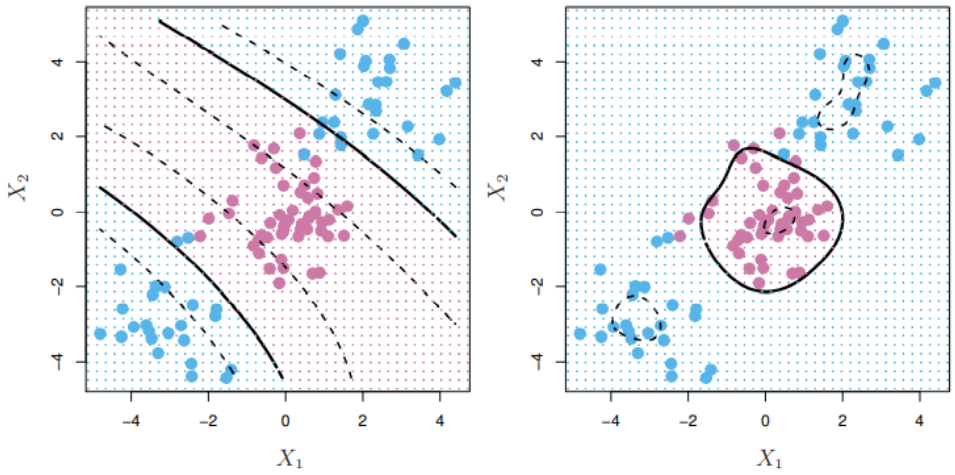

Figura 3.7: Exemplos de *support vector machines* com diferentes *kernels*: à esquerda, um polinomial, e à direita, um radial. Fonte: *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013.

ainda vou fazer uma imagem para essa frase especificamente!].

Com essa nova formulação, geramos o que chamamos de *support vector classifiers*, ou *soft margin classifiers*. No segundo nome, *soft* se refere ao fato de que a margem estabelecida para o hiperplano separador não é estrita, permitindo que algumas observações estejam incorretamente classificadas. O termo *support vector* se refere às observações cujas distâncias ao hiperplano definem a margem deste: caso elas fossem movidas, o hiperplano também teria sua forma alterada.

Como é possível perceber, porém, todos os métodos que vimos nessa seção trabalham com separações lineares, independentemente da dimensão p do espaço dos dados. Para sermos capazes de modelar dados nos quais a separação entre as classes tem forma não-linear, devemos usar modelos do tipo *support vector machines*.

A ideia por trás de modelos do tipo *support vector machine* é que se mapearmos o espaço p-dimensional original para um novo espaço m-dimensional, com $m > p$, podemos continuar utilizando o problema de otimização original para encontrar um hiperplano de separação linear - isso é, o *support vector classifier*.

A função que faz esse mapeamento é chamada de função *kernel*. O propósito dessa função é quantificar a similaridade de duas observações, sendo comumente representada na forma $K(x_i, x_{i'})$. Na geração do *support vector classifier* original, uma função *kernel* da seguinte forma é usada:

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

De forma que podemos definir o *support vector classifier* como:

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x_i, x_{i'}) = \beta_0 + \sum_{i \in S} \alpha_i x_{ij} x_{i'j}$$

Onde S é o conjunto de observações que são *support vectors* do hiperplano. A primeira parte da equação acima é uma representação geral de *support vector machines*; já que a função *kernel* pode assumir diferentes formas e, portanto, gerar hiperplanos capazes de criar separações não-lineares. Um *kernel* comumente usado é o polinomial de grau d:

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d.$$

4
Aplicações

Neste capítulo, iremos aplicar os métodos estudados sobre três diferentes bases de dados, visando gerar resultados que permitam a comparação e análise prática dos modelos de classificação.

4.1
Metodologia

Nesta seção, falaremos dos processos gerais aplicados sobre todas as bases de dados usadas; etapas mais particulares de cada uma serão expostas nas seções seguintes.

Toda a criação e exploração dos modelos de aprendizado que serão apresentados ao longo das próximas seções se baseou principalmente no uso do pacote **caret** (Kuhn 2008), disponível para a linguagem R. Essa escolha se deu devido à possibilidade de padronizar todo o processo de configuração, treinamento e teste dos diferentes modelos a serem aplicados no projeto. Além disso, o pacote disponibiliza diversos modelos de classificação para uso, entre os quais foram escolhidos alguns para uso neste projeto (descritos na Tabela 4.1).

Inicialmente, foi feito um pré-processamento básico em todas as bases de dados. Esse pré-processamento incluiu a mudança de nomes de valores de variáveis categóricas para evitar erros de interpretação do R e a remoção de

Tabela 4.1: Modelos do **caret** escolhidos para a etapa de aplicação do projeto. A sigla de cada modelo se refere ao valor passado para o parâmetro `method` da função `train` do pacote.

Método teórico	Sigla	Título
Regressão Logística	`glm`	Generalized Linear Model
Regressão Log. Multinomial	`multinom`	Penalized Multinomial Regression
LDA	`lda`	Linear Discriminant Analysis
KNN	`knn`	k-Nearest Neighbors
Classification Tree	`rpart`	CART
Random Forest	`rf`	Random Forest
Gradient Boosting	`gbm`	Stochastic Gradient Boosting
SVM	`svmPoly`	Support Vector Machines with Polynomial Kernel

variáveis com correlação maior que 70% usando a função `findCorrelation` do pacote.

Com as bases de dados pronta, foram criados conjuntos de treino e teste - respectivamente compostos de 80% e 20% dos dados. O conjunto de treino foi usado para a estimação de modelos, visando a geração de um modelo para cada um dos sete métodos de classificação expostos no capítulo 3. Já o conjunto de teste será usado para a geração de matrizes de confusão para alguns dos modelos após o fim do processo de treinamento.

O pacote **caret** facilita a estimação e *tuning* de parâmetros de modelo por meio de uma única função `train`. Por meio desta, é possível automatizar o teste de vários valores para os diferentes parâmetros de cada método (por exemplo, o número de árvores m sendo usadas para um modelo *random forest*), escolher o melhor modelo de acordo com alguma métrica definida *a priori* e gerar métricas de *performance* do modelo selecionado nos dados usados para treino.

Para a seleção da configuração de um modelo entre várias possibilidades de seu conjunto de parâmetros, um método de validação comumente usado é o de *k-fold cross validation* (James et al. 2013). Neste, o conjunto de dados usado para treinamento é inicialmente dividido de forma aleatória em k subconjuntos (*folds*); estes são então usados como base para um algoritmo iterativo no qual um modelo é treinado sobre $k-1$ subconjuntos e depois testado sobre o k-ésimo subconjunto que foi deixado de fora. A acurácia final do modelo é calculada como a agregação da acurácia dos modelos sobre o conjunto deixado de lado (*holdout*) durante cada iteração do *cross-validation*.

Apesar da efetividade desse método como uma maneira de reduzir o viés de testes de um modelo estatístico, ele pode sofrer de um grau elevado de variância, já que o resultado final pode depender significativamente da divisão feita inicialmente do conjunto nos k *folds*. Uma maneira de reduzir a variância, portanto, é criar n conjuntos de diferentes k *folds* e aplicar o método de *cross validation* sobre cada um desses conjuntos: processo chamado então de n *times repeated cross validation*.

Para cada um dos 7 métodos escolhidos para estudo, foi usado *5 times repeated 10-fold cross validation* para a escolha de apenas um modelo para cada método. A métrica de acurácia usada para a escolha final variou de base para base, e será exposta nas seções seguintes. Os *folds* usados na estimação de todos os modelos foram os mesmos - criados *a priori* via a função `createMultiFolds` - visando manter as mesmas condições de estimação mesmo entre os diferentes métodos.

Para a comparação dos 7 modelos finais em cada base, foram agregadas

várias métricas de *performance*, que serão explicadas na próxima subseção do documento. Além disso, foi usado o pacote **MLeval** (John 2020), que gera gráficos e métricas de forma automática usando as métricas resultantes do *resampling* feito durante a estimação de modelos via a função `train` mencionada anteriormente.

4.1.1 Métricas de ajuste

Para facilitar a interpretação dos resultados, iremos apresentar de forma simplificada as métricas e termos usados para a validação dos modelos ao longo das próximas seções.

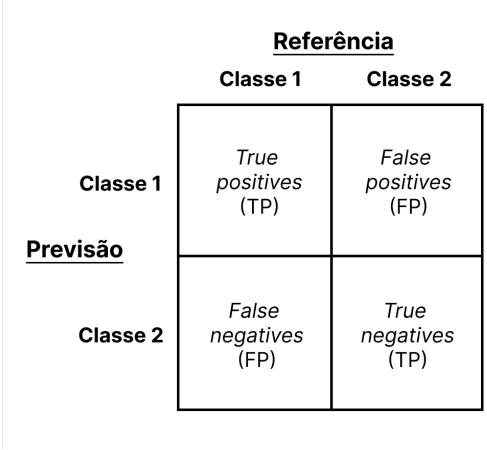

Figura 4.1: Exemplo de matriz de confusão para um problema de classificação binário. Fonte: elaboração própria.

Uma das maneiras mais básicas de entender um modelo de classificação é a partir de uma matriz de confusão - uma matriz m por m (onde m é o número de classes da variável dependente) na qual as previsões de um modelo para uma base de dado são relacionadas com as classes reais de cada observação prevista.

No caso de um problema binário, a matriz resultante de um modelo é da forma exibida na Figura 4.1. Valores na diagonal principal da matriz representam acertos, isso é, observações para qual a classe prevista foi igual à classe verdadeira (de referência); valores fora dessa diagonal indicam observações cuja classe não foi prevista corretamente pelo modelo.

Observações que foram classificadas corretamente, no caso do problema binário, se dividem em dois casos: *true positives*/TP e *true negatives*/TN. De maneira análoga, observações que foram classificadas incorretamente podem fazer parte do grupo *false positive*/FP ou *false negative*/FN. A quantidade total de observações previstas deve ser igual à $\text{TP} + \text{TN} + \text{FP} + \text{FN}$.

A partir desses grupos definidos por uma matriz de confusão, podemos definir as métricas que serão usadas nas próximas seções:

- **Acurácia**: a porcentagem de observações corretamente classificadas pelo modelo; equivalente a $\frac{TP+TN}{TP+TN+FP+FN}$.
- **Sensitividade (*sensitivity*)/*recall***: a taxa de observações da classe "positiva" que foram corretamente classificadas pelo modelo. Equivalente a $\frac{TP}{TP+FN}$.
- **Especificidade (*specificity*)**: a taxa de observações da classe "negativa" que foram corretamente classificadas pelo modelo. Equivalente a $\frac{TN}{TN+FP}$.
- **Precisão (*precision*)**: a taxa de classificações corretas pertencetes à classe "positiva" entre as previsões corretas. Equivalente a $\frac{TP}{TP+TN}$.

Tendo essas métricas definidas, restam apenas três outras que mencionaremos no restante do documento; essas, porém, precisam de explicações adicionais.

Modelos de classificação podem ter como saída um de dois tipos de previsões: determinísticas, quando o próprio modelo determina a classe prevista, e probabilísticas, quando o modelo retorna probabilidades da observação estar em cada uma das classes de saída. Note que é fácil gerar saídas do primeiro tipo simplesmente determinando algum valor de *cutoff* a partir do qual uma observação é classificada como uma classe ou outra.

A partir de saídas probabilísticas, é possível gerar outras métricas interessantes. Uma é a *log loss* (perda logística), equivalente ao desvio multinomial que definimos na seção 3.6 (*Gradient boosting*).

Outro resultado que pode ser gerado a partir de saídas probabilísticas é a curva ROC, de *receiver operating characteristic*. A ideia da curva ROC (visualizável na Figura 4.3) é plotar os valores de sensitividade e taxa de *false positive* (equivalente à $1-$specificity) para diferentes valores de *cutoff*, permitindo uma rápida avaliação e comparação visual de modelos (James et al. 2013). Já que uma curva ROC ideal preenche todo o plano e, por isso, tem valor de área igual a 1, uma forma de comparar numericamente diferentes curvas ROC é quantificando a área debaixo da curva (AUC ou *area under curve*), gerando a métrica ROC-AUC. Dessa forma, por construção a ROC-AUC $\in [0, 1]$ e quanto mais perto de 1, melhor a capacidade classificatória do modelo.

Finalmente, outra curva que pode ser gerada de maneira análoga à curva ROC porém usando outras métricas como base é a curva PR, de *precision-recall* (Figura 4.4). A mesma lógica usada para a métrica ROC-AUC pode então ser aplicada sobre essa curva, gerando a PR-AUC.

4.2 Base 1: Company Bankruptcy

A primeira base usada no projeto consiste de dados sobre empresas taiwanesas, coletados a partir do *Taiwan Economic Journal* entre os anos de 1999 e 2009 (Liang et al. 2016). A base consiste de 95 preditores (todos numéricos, sendo indicadores financeiros como ativos totais e fluxo de caixa) e uma variável dependente binária indicando se a empresa sofreu falência de acordo com parâmetros estabelecidos pela bolsa de valores de Taiwan.

Tabela 4.2: Algumas das variáveis da base de dados 1; sendo a primeira a usada como dependente para os modelos estimados.

Nome	Tipo	Descrição
Bankrupt	Categórica	Valor binário indicando se a empresa faliu ou não.
OGM	Numérica	Margem bruta operacional (*operating gross margin.*
OPR	Numérica	Margem operacional (*operating profit rate*).
NVPSA	Numérica	Valor líquido por ação (*net value per share*).
CFPS	Numérica	Fluxo de caixa por ação (*cash flow per share*).
NIF	Categórica	Valor binário indicando se o resultado líquido da empresa nos dois últimos anos foi positivo ou não.

Um primeiro fato a se destacar sobre essa base é que apesar da abundância de preditores a serem considerados inicialmente, a função `findCorrelation` do **caret** identificou 32 destes como tendo uma correlação maior do que 70%. Posteriormente, com o corte destes preditores altamente correlacionados, os dados passaram a possuir apenas 63 variáveis independentes para uso na estimação.

O segundo fato a ser notado sobre a base é que ela é extremamente desbalanceada: das 6819 observações, apenas 220 são de fato empresas que sofreram falência; ou seja, apenas 3% das observações pertencem à classe que seria mais interessante de identificar.

Esse tipo de distribuição desbalanceada de amostras entre as classes da variável dependente de uma base pode causar diversos problemas. Um exemplo é a dificuldade que modelos terão de separar a classe minoritária da(s) classe(s) majoritária(s) (Sun et al. 2011), principalmente em casos onde a quantidade total de observações não é grande o bastante para minimizar a ausência (relativa) de amostras da classe minoritária.

Outro problema notável causado pela presença de um desbalanceamento de classes é a menor confiabilidade de métricas comuns de validação de modelos

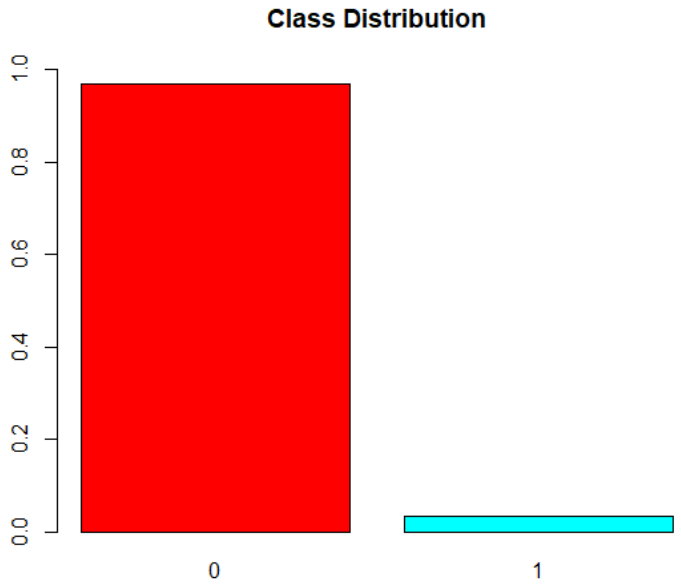

Figura 4.2: Distribuição das instâncias da base de dados em relação à classe *Bankrupt*; onde o valor 1 representa as empresas que sofreram bancarrota. Fonte: elaboração própria.

(Branco et al. 2015). Um exemplo simples é a métrica de acurácia (*accuracy*), que mede a porcentagem de amostras corretamente classificadas pelo modelo. No caso dessa base, se gerássemos um modelo que classificasse todas as observações de dados como não tendo falido, teríamos uma acurácia de 97% - mesmo usando um modelo que não possui valor real como ferramenta de compreensão dos dados ou previsão do que de fato seria importante em uma situação real (quais empresas possuem chance de falência).

Por essas razões e considerando que é mais interessante gerarmos modelos que consigam identificar casos de falência mesmo que sacrificando a identificação de empresas que não faliram (a classe majoritária), foi definida como métrica para seleção do melhor modelo para cada método de classificação a PR-AUC - sigla para *Precision Recall-Area Under Curve*. Apesar das métricas de precisão e *recall* não serem imunes aos problemas advindos de uma distribuição de classes desbalanceada, elas se adequam melhor a esse tipo de problema (John 2020).

Para mitigar as consequências negativas desse tipo de distribuição irregular entre as classes da variável dependente, o pacote **caret** propõe métodos de *sampling* adicionais, que se somam a outros métodos previamente estabelecidos como o *cross-validation* (Kuhn 2019). Dessa forma, foi escolhido como método de *resampling* adicional o *downsampling*, no qual amostras da classe majoritária no conjunto de treino são aleatoriamente selecionadas até que se

Tabela 4.3: Métricas dos modelos estimados na base 1, obtidas durante a estimação via *cross validation*. Os melhores valores para cada coluna (exceto tempo) estão marcados em negrito.

Método	Acc	ROC-AUC	Sens	Spec	Prec	*Log loss*	Tempo
LogReg	0.786	0.787	0.753	0.788	0.106	3.759	4.840
LDA	0.817	0.860	0.783	0.818	0.126	0.721	3.434
KNN	0.620	0.631	0.559	0.622	0.047	0.739	9.000
CTree	0.799	0.854	0.840	0.798	0.122	0.728	5.250
RForest	0.834	**0.931**	**0.874**	0.833	0.149	0.359	60.598
GBoost	**0.849**	**0.931**	0.866	**0.849**	**0.160**	**0.344**	26.904
SVM	0.811	0.884	0.837	0.810	0.128	0.560	175.29

forme um subconjunto do mesmo tamanho das amostras da classe minoritária, de forma que suas proporções se igualem.

Embora a técnica de *downsampling* em teoria acarrete em perda de informação e, portanto, adicione variância aos modelos gerados, testamos outros tipos de *resampling* (como *upsampling*, que tem funcionamento análogo) e percebemos resultados semelhantes entre as técnicas; de forma que, por fim, o *downsampling* foi escolhido como opção final.

4.2.1 Resultados

As métricas de performance dos modelos finais (um para cada método) agregadas ao longo da *repeated cross-validation* estão exibidas na Tabela 4.3, com ênfase no modelo que obteve melhor performance para cada uma.

De forma geral, o modelo de *gradient boosting* obteve o melhor resultado, com valores melhores em quase todas as métricas. As Figuras 4.3 e 4.4, que exibem respectivamente as curvas de ROC e *precision-recall*, permitem a chegada na mesma conclusão de maneira visual.

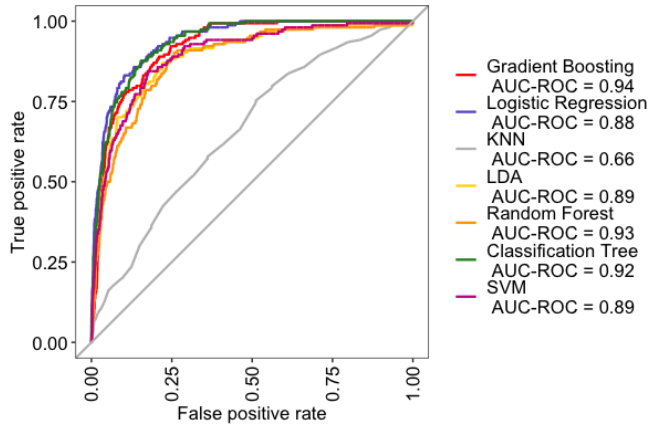

Figura 4.3: Curva ROC (*receiver operating characteristic*) dos modelos finais treinados na base 1.

Os valores relativamente semelhantes de sensitividade e especificidade dos modelos nos permitem concluir que a configuração que aplicamos para modelagem (em particular, os métodos de *resampling*) foram efetivos em fazer com que a classe minoritária fosse adequadamente considerada nas estimações dos parâmetros dos modelos, apesar da sua menor proporção entre as amostras da base.

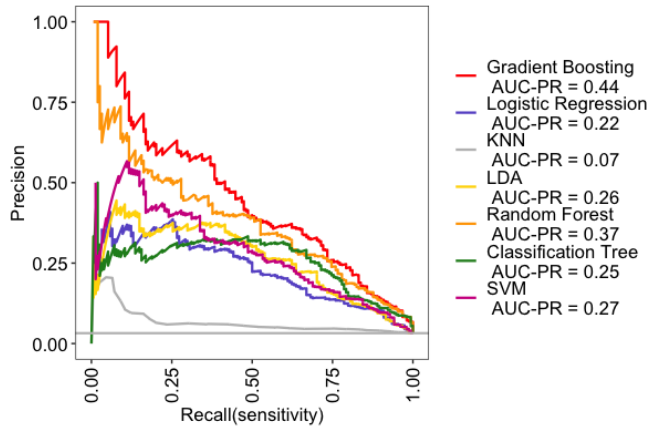

Figura 4.4: Curva PR (*precision-recall*) dos modelos finais treinados na base 1.

Embora os modelos de *boosting* e *random forest* tenham tido as melhores performances sobre a base, nota-se que o KNN destoou de todos os outros modelos, resultado em métricas muito precárias. Isso pode ser justificado pela presença de uma grande quantidade de preditores na base: já que o cálculo da distância entre pontos considera todos os preditores, é possível que muitos preditores tenham valores semelhantes mesmo entre classes diferentes,

diminuindo o efeito de preditores que consigam separar bem as classes no cálculo da distância entre os vizinhos de um determinado ponto.

Gradient Boosting

	Reference	
Prediction	Bankrupt	NotBankrupt
Bankrupt	40	195
NotBankrupt	4	1124

Logistic Regression

	Reference	
Prediction	Bankrupt	NotBankrupt
Bankrupt	39	236
NotBankrupt	5	1083

KNN

	Reference	
Prediction	Bankrupt	NotBankrupt
Bankrupt	23	486
NotBankrupt	21	833

Figura 4.5: Matrizes de confusão para três dos modelos criados sobre a base 1. Note que apesar do modelo de regressão logística conseguir resultados similares ao de *gradient boosting* para a classe "positiva", o último gera menos observações falsas negativas.

4.3 Base 2: Student Alcohol Consumption

A segunda base usada no projeto é composta por dados sobre alunos de duas escolas secundárias portuguesas ao longo de um mesmo ano escolar (Corsil et al. 2008). Os dados contêm 32 preditores, que podem ser divididos em: informações gerais (como sexo e idade), sociais (qualidade de relações familiares, tempo gasto com amigos) e acadêmicas (quantidade de reprovações e notas de duas provas no ano escolar). Todos os dados foram obtidos por meio de questionários completados pelos alunos, enquanto apenas as informações acadêmicas foram obtidas por outra fonte (relatórios acadêmicos disponibilizados pelas escolas).

Tabela 4.4: Algumas das variáveis da base de dados 2, pré-tratamento da variável dependente.

Nome	Tipo	Descrição
G3	Numérica	Valor entre 1 e 20 referente à nota do aluno na última prova do ano escolar.
G2	Numérica	Valor entre 1 e 20 referente à nota do aluno na segunda prova do ano escolar.
G1	Numérica	Valor entre 1 e 20 referente à nota do aluno na primeira prova do ano escolar.
activity	Categórica	Valor binário indicando se o aluno pratica atividades extracurriculares.
higher	Categórica	Valor binário indicando se o aluno deseja fazer ensino superior.
famrel	Categórica	Qualidade das relações familiares do aluno (valores entre "muito ruim" e "muito bom").
health	Categórica	Qualidade da saúde do aluno (valores entre "muito ruim" e "muito bom").
Walc	Categórica	Consumo de álcool pelo aluno durante o fim de semana (valores entre "muito baixo" e "muito alto").

A variável dependente dessa base - a nota do aluno na última prova do ano escolar (G3 na Tabela 4.4) - é, em sua forma original, numérica. Para que possamos tratar o problema de prever esse valor como um de classificação e não de regressão, foi aplicada uma conversão dessa variável de forma a torná-la categórica, de acordo com o padrão ERASMUS exibido na Tabela 4.5.

Tabela 4.5: Padrão de conversão de nota numérica para categoria ERASMUS.

Tipo	I	II	III	IV	V
Numérico	16-20	14-15	12-13	10-11	0-9
Categoria	A	B	C	D	F

Diferentemente do caso da base 1, as observações dessa base (649 no total) são bem distribuídas entre as diversas classes da variável dependente (Figura 4.6). Embora a maioria dos alunos tenha tido como resultado na última prova a categoria IV, as outras classes são relativamente bem populadas de amostras; dessa forma, essa base não possui as mesmas necessidades de *resampling* adicional como a anterior.

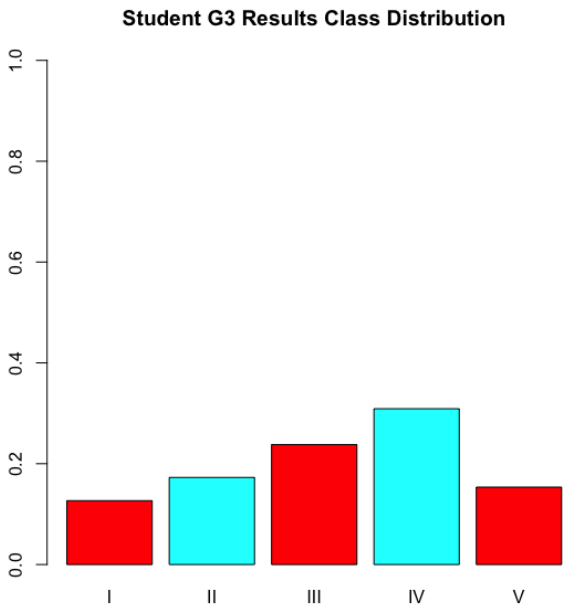

Figura 4.6: Distribuição das amostras entre as classes da variável dependente da base 2 - a nota na última prova de cada aluno.

Por termos definido o problema de classificação nessa base como um do tipo multinomial - no qual a variável dependente possui mais que 2 categorias - é necessário reconsiderar o processo de avaliação de modelos de classificação.

Métricas como sensitividade, especificidade, precisão e outras mais são definidas com base no problema de classificação binário, isso é, em resultados que podem ser classificados como a ocorrência ou não ocorrência de um evento. Apesar de ser possível caracterizar o problema multinomial como um problema binário assumindo (por exemplo) uma postura de medir a performance em relação à uma classe considerando todas as outras como a classe de não ocorrência, podemos também usar outras métricas mais adequadas para o caso multinomial.

Assim, a métrica que usamos para seleção do melhor modelo para cada método foi a perda logística (*log loss*), que busca quantificar a distância de um modelo em relação ao modelo ideal por meio da comparação de suas previsões e as categorias corretas de cada observação. Quanto menor o valor de *log loss* para um dado classificador, melhor ele está classificando as observações.

4.3.1 Resultados

Para essa base, obtivemos resultados menos discrepantes entre os modelos finais de cada método quando comparados com os resultados da base 1. Novamente, o modelo de *gradient boosting* se destacou como possuindo as melhores métricas, com o modelo *random forest* tendo resultados semelhantes.

Tabela 4.6: Métricas dos modelo estimados na base 2, obtidas durante a estimação via *cross validation*.

Método	Acc	*Log loss*	ROC-AUC	Tempo
LogReg	0.709	1.186	0.862	24.810
LDA	0.751	0.962	0.893	3.613
KNN	0.807	1.081	0.925	4.563
CTree	0.745	0.900	0.868	5.136
RForest	0.823	0.642	**0.943**	130.545
GBoost	**0.834**	**0.629**	0.942	95.781
SVM	0.733	0.911	0.883	473.939

O que é possível perceber ao analisarmos os resultados com mais detalhes é uma influência muito grande dos preditores G1 e G2 - as notas das outras provas feitas pelo aluno naquele ano escolar - sobre a previsão da variável dependente G3. Isso é particularmente visível na árvore de classificação gerada pelo método `rpart` (Figura 4.7), que usa exclusivamente o preditor G2 para fazer a divisão do espaço de decisão. Assim, preditores que poderiam ser entendidos como impactantes sobre a nota final do aluno - por exemplo, a quantidade de álcool que ele ingere nos fins de semana - não foram identificados como tendo um efeito separador significativo sobre as classes da variável dependente.

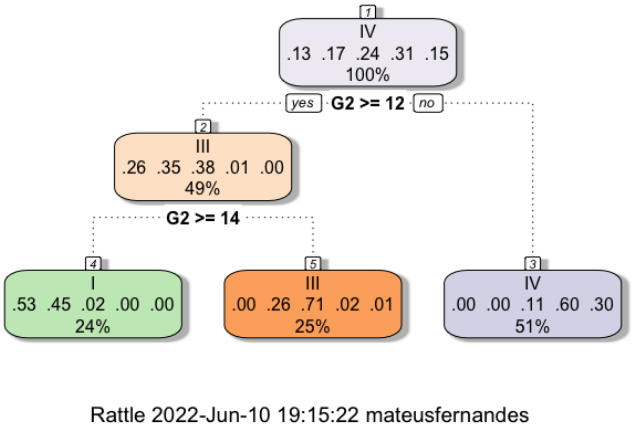

Figura 4.7: Árvore de classificação estimada pelo método `rpart` para a segunda base de dados.

Esses resultados estão de acordo com o trabalho realizado pelos criadores da base de dados (Corsil et al. 2008), que identificaram uma melhor performance pelos modelos de classificação baseados em árvore. A justificativa por trás desse comportamento pode estar relacionada com o número relativamente

alto de preditores de pouca importância: modelos baseados em árvore tendem a usar apenas alguns preditores (aqueles que geram uma maior separação das classes) para fazer o particionamento do espaço preditivo, diferentemente de métodos como o KNN e SVM, que usam todos os preditores de uma base em seus cálculos (respectivamente, para quantificar a distância entre vizinhos e de margem/*kernel*).

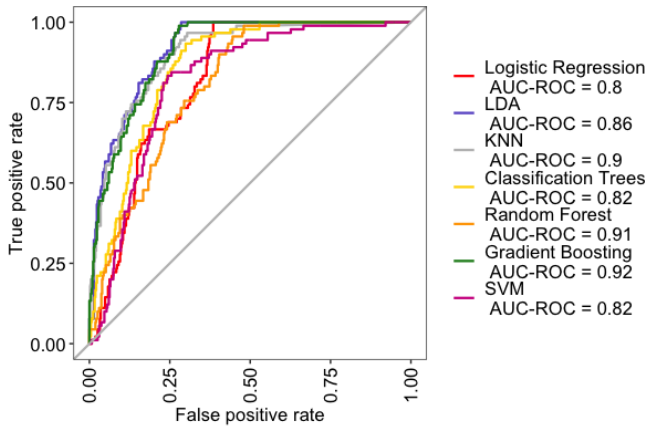

Figura 4.8: Curva ROC (*receiver operating characteristic*) dos modelos finais treinados na base 2.

Pelas matrizes de confusão geradas para os modelos (Figura 4.9), é possível perceber que a maior fonte de erro para o modelo de regressão logística (o que obteve o pior valor na métrica *log loss*) parece estar relacionada à uma dificuldade de classificar corretamente a classe IV - a qual é, comparativamente, bem classificada pelo modelo de *gradient boosting*.

Gradient Boosting

	Reference				
Prediction	I	II	III	IV	V
I	12	2	0	0	0
II	3	12	0	0	0
III	0	7	25	2	0
IV	1	1	5	38	7
V	0	0	0	0	13

Logistic Regression

	Reference				
Prediction	I	II	III	IV	V
I	11	7	0	0	0
II	4	5	6	1	0
III	0	7	16	10	0
IV	1	3	8	24	9
V	0	0	0	5	11

Figura 4.9: Matrizes de confusão para os modelos de melhor e pior performance (em termos de *log loss* dos gerados a partir da base 2.

4.4 Base 3: PNAD COVID19

A terceira base de dados usada foi criada a partir da Pesquisa Nacional por Amostra de Domicílios (PNAD COVID19) feita pelo IBGE (Base dos Dados 2022). A pesquisa tinha como objetivo estimar o número de pessoas com sintomas associados à síndrome gripal e monitorar os impactos da pandemia da COVID-19 no mercado de trabalho brasileiro, podendo ser dividida portanto em duas partes: uma direcionada a questões de saúde e outra a questões de trabalho.

Para uso no nosso projeto, decidimos focar na primeira parte, isso é, nas perguntas relacionadas à área de saúde feitas pela pesquisa. A ideia era transformar essa base em uma base que relacionasse sintomas relatados com um diagnóstico de COVID19 por parte do entrevistado, já que não foi encontrada nenhuma base *online* adequada que tivesse essas características. Com essa transformação feita, seria possível criar um modelo que classificasse uma observação como tendo COVID ou não, baseando-se em sintomas identificados e outros dados básicos (como idade, sexo e raça).

Um problema encontrado, porém, é que muitas das perguntas da pesquisa podiam ter como resposta valores "N/A" e semelhantes, o que dificultaria o uso das amostras com esses valores no processo de aprendizado. Além disso, muitas perguntas (e, portanto, colunas da base) estavam conectadas entre si, fazendo com que informações interessantes oferecidas pela base ficassem inicialmente fragmentadas. Um exemplo disso eram as perguntas das pesquisa relacionadas ao entrevistado ter feito teste de COVID ou não: uma coluna da base era dedicada a perguntar se o entrevistado havia feito um tipo de teste ou não, e apenas a coluna seguinte indicaria o resultado do teste de fato (se foi positivo, negativo ou inconclusivo).

Dessa forma, foi necessário fazer um pré-tratamento dos dados de forma a criar preditores e até uma variável dependente mais facilmente usável para a criação de modelos. Observações com qualquer uma das colunas relacionadas à sintomas possuindo valores faltantes foram removidas da base, já que ao todo representavam menos de 0.5% das amostras. Para a criação da variável dependente (indicando se o entrevistado teve COVID ou não), foram unidas em uma nova variável binária (chamada `covid`) as três colunas relacionadas aos diferentes tipos de teste abordados na entrevista (coleta nasal, exame de sangue via veia e via furo no dedo). Caso o entrevistado houvesse indicado um resultado positivo para pelo menos um dos testes, a variável `covid` indicaria `true`; caso contrário (ou seja, recebeu resultados inconclusivos ou negativos), a variável indicaria `false`.

Tabela 4.7: Algumas das variáveis da base de dados 3 após a realização do tratamento sobre os dados originais. Todas as variáveis relacionadas a sintomas tinham seu valor definido com base no entrevistado relatar o sintoma pelo menos uma semana antes da execução da pesquisa pelo IBGE.

Nome	Tipo	Descrição
covid	Categórica	Valor binário indicando se o entrevistado relatou pelo menos um exame positivo de COVID19.
senseloss	Categórica	Valor binário indicando se o entrevistado relatou ter perda de sentidos (olfato, paladar, etc).
cough	Categórica	Valor binário indicando se o entrevistado relatou ter tosse.
sorethroat	Categórica	Valor binário indicando se o entrevistado relatou ter dor de garganta.
tiredness	Categórica	Valor binário indicando se o entrevistado relatou fadiga.
age	Numérica	Idade do entrevistado.
edu	Categórica	Nível de escolaridade do entrevistado.

Ao fim desse pré-tratamento, a base ajustada conta com um total de 10.832 observações e 14 preditores; sendo 10 desses variáveis relacionadas à se o entrevistado possuía ou não um sintoma comum entre pacientes de COVID - por exemplo, perda de sentidos e fadiga. Apesar da maioria das amostras - aproximadamente 80% - ter um diagnóstico negativo de COVID, vemos que nesta base não existe a mesma configuração de desbalanço extremo entre as classes da variável dependente como na base 1. Dessa forma, a escolha de métrica para seleção do modelo final para cada método foi a ROC-AUC (*receiver operating characteristic - area under curve*), que serve como uma boa métrica para determinação da capacidade de um modelo de classificação de separar observações negativas e positivas (Rosset 2004).

4.4.1 Resultados

Para a terceira base, não obtivemos bons resultados em nenhum dos modelos. Isso é particularmente visível pela curva ROC dos modelos (Figura 4.10) - todos os modelos geraram curvas muito próximas da linha base da curva ROC, que é equivalente à curva esperada de um modelo que classifica amostras aleatoriamente entre as duas classes (isso é, prevê 50/50).

Tabela 4.8: Métricas dos modelos estimados na base 3, obtidas durante a estimação via *cross validation*.

Método	Acc	ROC-AUC	Sens	Spec	Prec	*Log loss*	Tempo
LogReg	**0.762**	0.583	0.069	0.993	0.780	0.546	5.541
LDA	**0.762**	0.583	0.070	0.993	0.782	0.546	5.537
KNN	0.744	0.537	0.065	0.971	0.436	0.942	47.216
CTree	0.758	0.567	**0.073**	0.987	0.670	0.557	7.582
RForest	0.759	0.541	0.057	**0.994**	0.783	5.981	1624.1
GBoost	**0.762**	**0.585**	0.069	**0.994**	0.796	**0.545**	127.78
SVM	**0.762**	0.527	0.069	**0.994**	**0.801**	0.548	1338.2

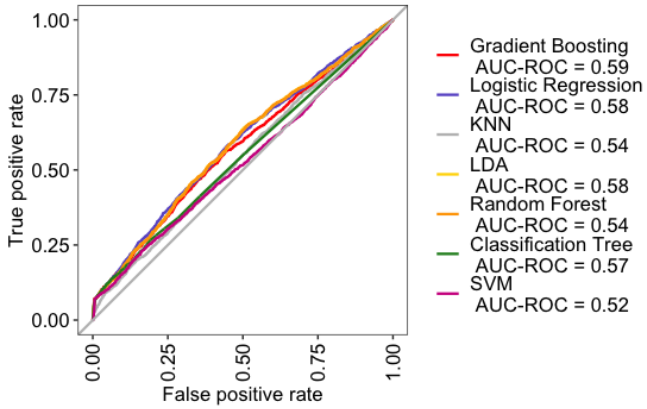

Figura 4.10: Curva ROC (*receiver operating characteristic*) dos modelos finais treinados na base 3.

Utilizando outras métricas (Tabela 4.8), podemos ver que o problema dos modelos provavelmente vai além de uma simples ênfase na previsão correta das classe majoritária em detrimento à previsão da classe minoritária. Em estimações iniciais feitas para a base 1 (na qual a classe minoritária tinha muitas poucas observações), obtivemos resultados similares aos dessa base, com altos valores de especificidade e baixos valores de sensitividade - porém com altos valores de ROC-AUC, indicando que os modelos conseguiam separar bem as classes. Aqui, os baixos valores da ROC-AUC indicam que os modelos não foram capazes de separar apropriadamente as classes da variável dependente. Apesar dos valores de acurácia parecerem aceitáveis (em torno de 70% para todos os modelos), isso é exatamente um exemplo do caso explicitado na seção 4.1 de bases desbalanceadas, na qual um modelo de pouca utilidade pode adquirir resultados aparentemente bons.

Uma justificativa para esses resultados seria o fato de que eles foram obtidos a partir de entrevistas e não de resultados médicos de fato: dessa forma, é possível que entrevistados não tenham relatado corretamente os sintomas que sentiram, seja de propósito ou por esquecimento/engano.

Outra possível justificativa seria a partir do fato de que apesar de muitos dos sintomas na base em teoria serem associados com COVID (como febre, coriza, entre outros), estes também podem indicar outras condições semelhantes (como viroses ou outras síndromes gripais). Dessa forma, seria difícil criar um modelo de classificação para diagnóstico sem considerar fatores mais específicos de saúde além de sintomas tão gerais.

G. Boosting (p=0.5)

Prediction	Reference X1	X0
X1	28	9
X0	398	1262

G. Boosting (p=0.2)

Prediction	Reference X1	X0
X1	369	1018
X0	57	253

KNN

Prediction	Reference X1	X0
X1	30	31
X0	396	1240

Figura 4.11: Matrizes de confusão para dois dos modelos gerados a partir da base 2. Note que as matrizes da esquerda e do centro foram obtidas a partir do mesmo modelo de *gradient boosting*; para a obtenção da matriz central, porém, utilizou-se um valor menor de *cutoff* para a determinação de uma observação ser classificada como positiva ("X1"). Embora isso melhore a taxa de acertos para as observações da classe positiva, o prejuízo na previsão correta de observações da outra classe é notável.

Seja qual for o caso, esses resultados nos levam à conclusão de que - considerando os dados gerados pelo tratamento dos resultados da pesquisa feita pelo IBGE - não é possível traçar uma relação significativa entre os sintomas mais relacionados com COVID e um diagnóstico certo da doença. Essa conclusão, porém, fica limitada ao escopo da base e dos modelos criados: outros tipos de modelagem e outras bases de dados poderiam gerar fins diferentes.

5
Conclusão

Neste projeto, foram apresentados sete métodos de classificação e os conceitos nos quais se baseiam. De forma geral, pode-se concluir que - pelo menos de um ponto de vista teórico - não existe um método completamente superior aos demais: mesmo métodos que servem como melhorias ou extensões de outros (como o *random forest* em relação às árvores de classificação) costumam envolver algum *tradeoff*.

De forma geral, pode-se afirmar que os métodos de implementação mais complexas - como o *gradient boosting* e *support vector machines* - tendem a sacrificar interpretabilidade para obterem melhores resultados na classificação de novos dados. Modelos de árvore de classificação e regressão linear, apesar de não serem tão poderosos em termos de capacidade preditiva, possuem grande valor por oferecerem uma compreensão melhor dos dados na forma de uma visualização intuitiva e valores de parâmetros interpretáveis, respectivamente. Mesmo um modelo como o KNN, que não apresenta nem performance nem interpretabilidade notável, serve como um *benchmark* para outros modelos e um bom ponto de entrada para o entendimento de problemas de classificação devido ao seu funcionamento de fácil compreensão.

Na parte de aplicação prática do projeto, identificamos as melhores performances por parte dos modelos mais complexos, com o *gradient boosting* apresentando melhores resultados em todos os casos - o que não foi inesperado: uma vez que métodos de *ensemble* são conhecidos por possuírem capacidade preditiva elevada (James et al. 2013). É importante destacar, porém, que esses resultados não deveriam servir como base para um entendimento de que o *gradient boosting* é o melhor método para classificação em todos os casos. Como foi colocado na primeira seção deste documento, nenhum método (seja de classificação ou de outro tipo de aplicação) é adequado para todos os problemas no âmbito do aprendizado de máquina. Também deve-se levar em conta que os resultados relativos à acurácia classificatória dos métodos foram baseados na aplicação destes a apenas três bases de dados; sendo necessário um estudo mais aprofundado e usando mais bases para a determinação de conclusões sobre a capacidade classificatória desses métodos de forma mais geral.

A
Apêndice: Códigos

Código 1: Código de aplicação na base 1.

```
set.seed(191)
rm(list = ls())

library(randomForest)
library(caTools)
library(caret)
library(ggplot2)
library(MLeval)
library(rattle)
library(themis)

# IMPORTING AND PREPARING DATA

data <- read.csv("Databases/bankrupt.csv", header = TRUE)

# we remove the Net Income Flag variable since all rows have the
    same value
data <- subset(data, select=-Net.Income.Flag)
data$ID <- seq.int(nrow(data))
data$Bankrupt. <- factor(data$Bankrupt.)
data$Liability.Assets.Flag <- factor(data$Liability.Assets.Flag)

# removing collinear variables
data_num <- subset(data, select=-c(Bankrupt.,Liability.Assets.Flag))
corr.matrix <- cor(data_num)
hc <- findCorrelation(corr.matrix, cutoff = 0.70)
data_num <- data_num[,-hc]
data_factor <- subset(data, select=c(ID, Bankrupt., Liability.Assets
    .Flag))
data_factor$Liability.Assets.Flag <- make.names(data_factor$
    Liability.Assets.Flag)
data <- subset(merge(data_factor, data_num, by="ID"), select=-ID)
rm(list= c("data_num", "data_factor"))
levels(data$Bankrupt.) <- c("NotBankrupt", "Bankrupt")
data$Bankrupt. <- factor(data$Bankrupt., levels=rev(levels(data$
    Bankrupt.)))

inTraining <- createDataPartition(data$Bankrupt., p = .8, list =
    FALSE)
training <- data[ inTraining,]
```

```
36 holdout  <- data[-inTraining,]
37 rm(inTraining)
38 full_data <- data
39 data <- training
40
41 folds <- createMultiFolds(data$Bankrupt., k=10, times=5)
42
43 # data is very imbalanced. 6599 cases of non-bankruptcy and only 220
       cases of bankruptcy
44 barplot(prop.table(table(data$Bankrupt.)),
45         col = rainbow(2),
46         ylim = c(0, 1.0),
47         main = "Company Bankruptcy Class Distribution")
48
49 fitControl <- trainControl(method = "repeatedcv", repeats=5, index=
       folds, savePredictions = "final", classProbs = TRUE,
50                            sampling="down", verboseIter = FALSE,
                                  summaryFunction = prSummary,
51                            returnResamp = "final")
52
53 train_model <- function(method_name, family_name=NULL, training_data
       =data, testing_data=holdout) {
54     t1 <- proc.time()
55     if (is.null(family_name)) {
56         temp_model <- train(Bankrupt.~., data=training_data, method=
              method_name,
57                             trControl = fitControl, metric="AUC")
58     } else {
59         temp_model <- train(Bankrupt.~., data=training_data, method=
              method_name,
60                             family=family_name, trControl =
                                  fitControl, metric="AUC")
61     }
62     t2 <- proc.time()
63     temp_model$elapsed_time <- (t2 - t1)[[3]]
64     temp_model$cm <- confusionMatrix(temp_model)
65     temp_model$default_summary <- defaultSummary(temp_model$pred)
66     temp_model$pr_summary <- prSummary(temp_model$pred, lev=levels(
          temp_model$pred$obs))
67     temp_model$log_loss <- mnLogLoss(temp_model$pred, lev=levels(
          temp_model$pred$obs))
68     temp_model$two_summary <- twoClassSummary(temp_model$pred, lev=
          levels(temp_model$pred$obs))
69
70     temp_model$test_pred <- predict(temp_model, newdata=testing_data
          , type="prob")
71     temp_model$test_pred$obs <- testing_data$Bankrupt.
72     temp_model$test_pred$Group <- method_name
73     nonprob_pred <- predict(temp_model, newdata=testing_data)
74     temp_model$test_cm <- confusionMatrix(nonprob_pred, testing_data
          $Bankrupt., mode="everything", positive="Bankrupt")
```

```
75      return (temp_model)
76  }
77
78  # LDA
79  # Net Growth Rate causes problems in some folds because a lot of
        values are repeated
80  set.seed(191)
81  lda <- train_model("lda", training_data=subset(data, select=-c(Net.
        Value.Growth.Rate)))
82
83  # SVM
84  set.seed(191)
85  svm <- train_model("svmPoly")
86
87  # Random Forest
88  set.seed(191)
89  forest <- train_model("rf")
90
91  # Logistic Regression
92  set.seed(191)
93  logreg <- train_model("glm")
94
95  # KNN
96  set.seed(191)
97  knn <- train_model("knn")
98
99  # Classification Tree
100 set.seed(191)
101 tree <- train_model("rpart")
102 fancyRpartPlot(tree$finalModel)
103
104 # Gradient Boosting
105 set.seed(191)
106 boost <- train_model("gbm")
107
108 # Prepping final data visualization
109 methods <- c("Gradient Boosting", "Logistic Regression", "KNN", "LDA
        ", "Random Forest", "Classification Tree", "SVM")
110 roc <- c(boost$two_summary[1],
111          logreg$two_summary[1],
112          knn$two_summary[1],
113          lda$two_summary[1],
114          forest$two_summary[1],
115          tree$two_summary[1],
116          svm$two_summary[1])
117
118 kappa <- c(boost$default_summary[2],
119               logreg$default_summary[2],
120               knn$default_summary[2],
121               lda$default_summary[2],
122               forest$default_summary[2],
```

```
123                tree$default_summary[2],
124                svm$default_summary[2])
125
126 accuracy <- c(boost$default_summary[1],
127                logreg$default_summary[1],
128                knn$default_summary[1],
129                lda$default_summary[1],
130                forest$default_summary[1],
131                tree$default_summary[1],
132                svm$default_summary[1])
133
134 sensitivity <- c(boost$two_summary[2],
135                  logreg$two_summary[2],
136                  knn$two_summary[2],
137                  lda$two_summary[2],
138                  forest$two_summary[2],
139                  tree$two_summary[2],
140                  svm$two_summary[2])
141
142 specificity <- c(boost$two_summary[3],
143                  logreg$two_summary[3],
144                  knn$two_summary[3],
145                  lda$two_summary[3],
146                  forest$two_summary[3],
147                  tree$two_summary[3],
148                  svm$two_summary[3])
149
150 precision <- c(boost$pr_summary[2],
151                logreg$pr_summary[2],
152                knn$pr_summary[2],
153                lda$pr_summary[2],
154                forest$pr_summary[2],
155                tree$pr_summary[2],
156                svm$pr_summary[2])
157
158 recall <- c(boost$pr_summary[3],
159             logreg$pr_summary[3],
160             knn$pr_summary[3],
161             lda$pr_summary[3],
162             forest$pr_summary[3],
163             tree$pr_summary[3],
164             svm$pr_summary[3])
165
166 logloss <- c(boost$log_loss[1],
167              logreg$log_loss[1],
168              knn$log_loss[1],
169              lda$log_loss[1],
170              forest$log_loss[1],
171              tree$log_loss[1],
172              svm$log_loss[1])
173
```

```
time <- c(boost$elapsed_time,
          logreg$elapsed_time,
          knn$elapsed_time,
          lda$elapsed_time,
          forest$elapsed_time,
          tree$elapsed_time,
          svm$elapsed_time)

stats <- data.frame(accuracy, roc, sensitivity, specificity,
    precision, recall, logloss, time)
rm(accuracy, kappa, roc, sensitivity, specificity, precision, recall
    , logloss, time)
rownames(stats) <- methods
print(stats)

model_list <- list(logreg, lda, knn, tree, forest, boost, svm)
preds_list <- list()
for (model in model_list) {
    preds_list <- rbind(preds_list, model$test_pred)
}

eval_objects <- evalm(list(boost, logreg, knn, lda, forest, tree,
    svm), gnames=methods, positive="Bankrupt", silent=T, rlinethick
    = 0.8)
eval_test <- evalm(preds_list, positive="Bankrupt", silent=T,
    rlinethick = 0.8)

logregpred <- as.factor(ifelse(predict(logreg, newdata=holdout, type
    ="prob")$Bankrupt>0.5, "Bankrupt", "NotBankrupt" ))
logregcm <- confusionMatrix(logregpred, holdout$Bankrupt., mode="
    everything", positive="Bankrupt")

boostpred <- as.factor(ifelse(predict(boost, newdata=holdout, type="
    prob")$Bankrupt>0.5, "Bankrupt", "NotBankrupt" ))
boostcm <- confusionMatrix(boostpred, holdout$Bankrupt., mode="
    everything", positive="Bankrupt")

knnpred <- as.factor(ifelse(predict(knn, newdata=holdout, type="prob
    ")$Bankrupt>0.5, "Bankrupt", "NotBankrupt" ))
knncm <- confusionMatrix(knnpred, holdout$Bankrupt., mode="
    everything", positive="Bankrupt")
```

Código 2: Código de aplicação na base 2.

```
set.seed(191)
rm(list = ls())

library(randomForest)
library(caTools)
library(caret)
library(ggplot2)
```

```
8  library(MLeval)
9  library(rattle)
10 library(dplyr)
11 library(MLmetrics)
12
13 # IMPORTING AND PREPARING DATA
14
15 data <- read.csv("Databases/students.csv", header = TRUE)
16 variables <- names(data)
17 non_factors <- c("G1", "G2", "G3", "absences", "age")
18 factors <- variables[!(variables %in% non_factors)]
19 data[,factors] <- lapply(data[,factors], make.names)
20 data[,factors] <- lapply(data[,factors], factor)
21
22 # Transformando G3 em categórica usando o padrão ERASMUS
23 data <- mutate(data, newG3 = case_when(
24     G3 > 15 ~ "I",
25     G3 > 13 ~ "II",
26     G3 > 11 ~ "III",
27     G3 > 9 ~ "IV",
28     TRUE ~ "V"
29 ))
30
31 data$G3 <- data$newG3
32 data <- subset(data, select=-c(newG3))
33 data$G3 <- factor(data$G3)
34
35 inTraining <- createDataPartition(data$G3, p = .8, list = FALSE)
36 training <- data[ inTraining,]
37 holdout  <- data[-inTraining,]
38 rm(inTraining)
39 full_data <- data
40 data <- training
41
42 folds <- createMultiFolds(data$G3, k=10, times=5)
43
44 barplot(prop.table(table(data$G3)),
45         col = rainbow(2),
46         ylim = c(0, 1.0),
47         main = "Student G3 Results Class Distribution")
48
49
50 fitControl <- trainControl(method = "repeatedcv", repeats=5, index=
       folds, savePredictions = "final", classProbs = TRUE,
51                            verboseIter = FALSE, summaryFunction =
                                  multiClassSummary,
52                            returnResamp = "final")
53
54 train_model <- function(method_name, family_name=NULL, training_data
       =data, testing_data=holdout) {
55     t1 <- proc.time()
```

```
56      if (is.null(family_name)) {
57          temp_model <- train(G3~., data=training_data, method=method_
                name,
58                              trControl = fitControl, metric="logLoss"
                                    )
59      } else {
60          temp_model <- train(G3~., data=training_data, method=method_
                name,
61                              family=family_name, trControl =
                                    fitControl, metric="logLoss")
62      }
63      t2 <- proc.time()
64      temp_model$elapsed_time <- (t2 - t1)[[3]]
65      temp_model$cm <- confusionMatrix(temp_model)
66      temp_model$default_summary <- defaultSummary(temp_model$pred)
67      temp_model$log_loss <- mnLogLoss(temp_model$pred, lev=levels(
            temp_model$pred$obs))
68      temp_model$multi_summary <- multiClassSummary(temp_model$pred,
            lev=levels(temp_model$pred$obs))
69
70      temp_model$test_pred <- predict(temp_model, newdata=testing_data
            , type="prob")
71      temp_model$test_pred$obs <- testing_data$G3
72      temp_model$test_pred$Group <- method_name
73      return (temp_model)
74  }
75
76  # LDA
77  set.seed(191)
78  lda <- train_model("lda")
79
80  # SVM
81  svm <- train_model("svmPoly")
82
83  # Random Forest
84  set.seed(191)
85  forest <- train_model("rf")
86
87  # Logistic Regression
88  # nao eh binomial aqui!!
89  set.seed(191)
90  logreg <- train_model("multinom")
91
92  # KNN
93  set.seed(191)
94  knn <- train_model("knn")
95
96  # Classification Tree
97  set.seed(191)
98  tree <- train_model("rpart")
99  fancyRpartPlot(tree$finalModel)
```

```
100
101 # Gradient Boosting
102 set.seed(191)
103 boost <- train_model("gbm")
104
105 # Prepping final data visualization
106 methods <- c("Logistic Regression", "LDA", "KNN", "Classification
      Trees", "Random Forest", "Gradient Boosting", "SVM")
107 mean_balanced_accuracy <- c(logreg$results[order(logreg$results$
      logLoss, decreasing=FALSE),][1,]$Mean_Balanced_Accuracy,
108           lda$results[order(lda$results$logLoss, decreasing=FALSE)
                ,][1,]$Mean_Balanced_Accuracy,
109           knn$results[order(knn$results$logLoss, decreasing=FALSE)
                ,][1,]$Mean_Balanced_Accuracy,
110           tree$results[order(tree$results$logLoss, decreasing=FALSE)
                ,][1,]$Mean_Balanced_Accuracy,
111           forest$results[order(forest$results$logLoss, decreasing=
                FALSE),][1,]$Mean_Balanced_Accuracy,
112           boost$results[order(boost$results$logLoss, decreasing=FALSE
                ),][1,]$Mean_Balanced_Accuracy,
113           svm$results[order(svm$results$logLoss, decreasing=FALSE)
                ,][1,]$Mean_Balanced_Accuracy)
114 log_loss <- c(logreg$results[order(logreg$results$logLoss,
      decreasing=FALSE),][1,]$logLoss,
115                  lda$results[order(lda$results$logLoss, decreasing=
                       FALSE),][1,]$logLoss,
116                  knn$results[order(knn$results$logLoss, decreasing=
                       FALSE),][1,]$logLoss,
117                  tree$results[order(tree$results$logLoss, decreasing
                       =FALSE),][1,]$logLoss,
118                  forest$results[order(forest$results$logLoss,
                       decreasing=FALSE),][1,]$logLoss,
119                  boost$results[order(boost$results$logLoss,
                       decreasing=FALSE),][1,]$logLoss,
120                  svm$results[order(svm$results$logLoss, decreasing=
                       FALSE),][1,]$logLoss)
121 AUC <- c(logreg$results[order(logreg$results$logLoss, decreasing=
      FALSE),][1,]$AUC,
122           lda$results[order(lda$results$logLoss, decreasing=FALSE)
                ,][1,]$AUC,
123           knn$results[order(knn$results$logLoss, decreasing=FALSE)
                ,][1,]$AUC,
124           tree$results[order(tree$results$logLoss, decreasing=FALSE)
                ,][1,]$AUC,
125           forest$results[order(forest$results$logLoss, decreasing=
                FALSE),][1,]$AUC,
126           boost$results[order(boost$results$logLoss, decreasing=FALSE
                ),][1,]$AUC,
127           svm$results[order(svm$results$logLoss, decreasing=FALSE)
                ,][1,]$AUC)
128
```

```
time <- c(logreg$elapsed_time,
          lda$elapsed_time,
          knn$elapsed_time,
          tree$elapsed_time,
          forest$elapsed_time,
          boost$elapsed_time,
          svm$elapsed_time)

stats <- data.frame(mean_balanced_accuracy, log_loss, AUC, time)
rm(mean_balanced_accuracy, log_loss, AUC, time)
rownames(stats) <- methods
stats <- stats[order(stats$log_loss, decreasing = FALSE),]
print(stats)

# https://cran.r-project.org/web/packages/MLeval/vignettes/
    introduction.pdf
eval <- evalm(list(logreg, lda, knn, tree, forest, boost, svm),
    gnames=methods, silent=F, rlinethick = 0.8)
boostpred <- predict(boost, newdata=holdout)
boostcm <- confusionMatrix(boostpred, holdout$G3, mode="everything")

logregpred <- predict(logreg, newdata=holdout)
logregcm <- confusionMatrix(logregpred, holdout$G3, mode="everything
    ")
```

Código 3: Código de aplicação na base 3.

```
set.seed(191)
rm(list = ls())

library(randomForest)
library(caTools)
library(caret)
library(ggplot2)
library(MLeval)
library(rattle)
library(dplyr)

# IMPORTING AND PREPARING DATA

data <- read.csv("Databases/covid_dados_edited.csv", header = TRUE)
data <- subset(data, select=-c(X))
variables <- names(data)
non_factors <- c("age")
factors <- variables[!(variables %in% non_factors)]
data[,factors] <- lapply(data[,factors], make.names)
data[,factors] <- lapply(data[,factors], factor)
# X1 -> teve covid
# X0 -> não teve covid
data$covid <- factor(data$covid, levels=rev(levels(data$covid)))

```

```
25 inTraining <- createDataPartition(data$covid, p = .06, list = FALSE)
26 training <- data[ inTraining,]
27 holdout  <- data[-inTraining,]
28 rm(inTraining)
29 full_data <- data
30 data <- training
31
32 folds <- createMultiFolds(data$covid, k=10, times=5)
33
34 barplot(prop.table(table(training$covid)),
35         col = rainbow(2),
36         ylim = c(0, 1.0),
37         main = "Covid Diagnostic Distribution")
38
39 fitControl <- trainControl(method = "repeatedcv", repeats=5, index=
       folds, savePredictions = "final", classProbs = TRUE,
40                             sampling=NULL, verboseIter = FALSE,
                                   summaryFunction = twoClassSummary,
41                             returnResamp = "final")
42
43 train_model <- function(method_name, family_name=NULL, training_data
       =training) {
44     t1 <- proc.time()
45     if (is.null(family_name)) {
46         temp_model <- train(covid~., data=training_data, method=
               method_name, trControl = fitControl,
47                             metric="ROC")
48     } else {
49         temp_model <- train(covid~., data=training_data, method=
               method_name, family=family_name,
50                             trControl = fitControl, metric="ROC")
51     }
52     t2 <- proc.time()
53     temp_model$elapsed_time <- (t2 - t1)[[3]]
54     temp_model$cm <- confusionMatrix(temp_model)
55     temp_model$default_summary <- defaultSummary(temp_model$pred)
56     temp_model$pr_summary <- prSummary(temp_model$pred, lev=levels(
           temp_model$pred$obs))
57     temp_model$log_loss <- mnLogLoss(temp_model$pred, lev=levels(
           temp_model$pred$obs))
58     temp_model$two_summary <- twoClassSummary(temp_model$pred, lev=
           levels(temp_model$pred$obs))
59     return (temp_model)
60 }
61
62 # LDA
63 set.seed(191)
64 lda <- train_model("lda")
65
66 # SVM
67 set.seed(191)
```

```
svm <- train_model("svmLinear2")

# Random Forest
set.seed(191)
forest <- train_model("rf")

# Logistic Regression
set.seed(191)
logreg <- train_model("glm", "binomial")

# KNN
set.seed(191)
knn <- train_model("knn")

# Classification Tree
set.seed(191)
tree <- train_model("rpart")
fancyRpartPlot(tree$finalModel)

# Gradient Boosting
set.seed(191)
boost <- train_model("gbm")

# Prepping final data visualization
methods <- c("Gradient Boosting", "Logistic Regression", "KNN", "LDA
    ", "Random Forest", "Classification Tree", "SVM")
roc <- c(boost$two_summary[1],
         logreg$two_summary[1],
         knn$two_summary[1],
         lda$two_summary[1],
         forest$two_summary[1],
         tree$two_summary[1],
         svm$two_summary[1])

kappa <- c(boost$default_summary[2],
           logreg$default_summary[2],
           knn$default_summary[2],
           lda$default_summary[2],
           forest$default_summary[2],
           tree$default_summary[2],
           svm$default_summary[2])

accuracy <- c(boost$default_summary[1],
              logreg$default_summary[1],
              knn$default_summary[1],
              lda$default_summary[1],
              forest$default_summary[1],
              tree$default_summary[1],
              svm$default_summary[1])

sensitivity <- c(boost$two_summary[2],
```

```
                  logreg$two_summary[2],
                  knn$two_summary[2],
                  lda$two_summary[2],
                  forest$two_summary[2],
                  tree$two_summary[2],
                  svm$two_summary[2])

specificity <- c(boost$two_summary[3],
                  logreg$two_summary[3],
                  knn$two_summary[3],
                  lda$two_summary[3],
                  forest$two_summary[3],
                  tree$two_summary[3],
                  svm$two_summary[3])

precision <- c(boost$pr_summary[2],
                 logreg$pr_summary[2],
                 knn$pr_summary[2],
                 lda$pr_summary[2],
                 forest$pr_summary[2],
                 tree$pr_summary[2],
                 svm$pr_summary[2])

recall <- c(boost$pr_summary[3],
              logreg$pr_summary[3],
              knn$pr_summary[3],
              lda$pr_summary[3],
              forest$pr_summary[3],
              tree$pr_summary[3],
              svm$pr_summary[3])

logloss <- c(boost$log_loss[1],
               logreg$log_loss[1],
               knn$log_loss[1],
               lda$log_loss[1],
               forest$log_loss[1],
               tree$log_loss[1],
               svm$log_loss[1])

time <- c(boost$elapsed_time,
            logreg$elapsed_time,
            knn$elapsed_time,
            lda$elapsed_time,
            forest$elapsed_time,
            tree$elapsed_time,
            svm$elapsed_time)

stats <- data.frame(accuracy, kappa, roc, sensitivity, specificity,
    precision, recall, logloss, time)
rm(accuracy, kappa, roc, sensitivity, specificity, precision, recall
    , logloss, time)
```

```
167 rownames(stats) <- methods
168 print(stats)
169
170 model_list <- list(logreg, lda, knn, tree, forest, boost, svm)
171 preds_list <- list()
172 for (model in model_list) {
173     preds_list <- rbind(preds_list, model$test_pred)
174 }
175
176 # https://cran.r-project.org/web/packages/MLeval/vignettes/
      introduction.pdf
177 eval_objects <- evalm(list(boost, logreg, knn, lda, forest, tree,
      svm), gnames=methods, positive="X1", silent=T, rlinethick = 0.8)
178
179 newpartition <- createDataPartition(holdout$covid, p = .01, list =
      FALSE)
180 holdout_small <- holdout[ newpartition,]
181 boostpred <- as.factor(ifelse(predict(boost, newdata=holdout_small,
      type="prob")$X1>0.2, "X1", "X0" ))
182 boostcm <- confusionMatrix(boostpred, holdout_small$covid, mode="
      everything", positive="X1")
183
184 knnpred <- as.factor(ifelse(predict(knn, newdata=holdout_small, type
      ="prob")$X1>0.5, "X1", "X0" ))
185 knn <- confusionMatrix(knnpred, holdout_small$covid, mode="
      everything", positive="X1")
```

Código 4: Código de pré-tratamento dos dados da base 3.

```
1 set.seed(20)
2 rm(list = ls())
3
4 library(randomForest)
5 library(caTools)
6 library(caret)
7 library(ggplot2)
8 library(MLeval)
9 library(rattle)
10 library(dplyr)
11
12 # IMPORTING AND PREPARING DATA
13
14 original_data <- read.csv("Databases/covid_dados.csv", header = TRUE
      ) #2.650.459 obs
15
16 original_data <- subset(original_data, select=c(a002,a003,a004,a005,
      a006,a006b,
17                                                  b0011,b0012,b0013,
                                                        b0014,b0015,
                                                        b0016,
18                                                  b0017,b0018,b0019,
```

```
                                                    b00110,b00111,
                                                    b00112,
19                                                b00113,b008,b009a,
                                                    b009b,b009c,
                                                    b009d,b009e,
                                                    b009f))
20
21 # transformando b008 -> b009f em um dado conjunto sobre
22 # se o entrevistado teve covid ou não
23 original_data <- mutate(original_data, covid = case_when(
24     #se fez teste e qualquer um dos resultados deu positivo, seta 1
25     b008==1 & (b009b==1 | b009d==1 | b009f==1) ~ 1,
26     # a partir daqui, todos os casos que fizeram testes tiveram um
           mix de
27     # resultados negativos, inconclusivos ou não fizeram algum teste
            ou ignoraram
28     # ou seja, não tem nenhum caso de resultado positivo entre os
           testes
29     # vamos considerar que se ALGUM deu negativo, não é covid
30     b008==1 & (b009b==2 | b009d==2 | b009f==2) ~ 0
31 ))
32
33 # remove people who:
34 # don't have a value for the new covid variable
35 # didn't answer race
36 data <- original_data[!(is.na(original_data$covid)) & !(original_
       data$a004==9),] #181.876 obs
37 # don't know if they had one of the possible symptoms
38 data <- data[!(data$b0011==3 | data$b0012==3 | data$b0013==3 |
39                    data$b0014==3 | data$b0015==3 | data$b0016==3 |
40                    data$b0017==3 | data$b0018==3 | data$b0019==3 |
41                    data$b00110==3 | data$b00111==3 | data$b00112==3
                          |
42                    data$b00113==3),] #181.588 obs
43 # ignored one of the possible symptoms
44 data <- data[!(data$b0011==9 | data$b0012==9 | data$b0013==9 |
45                    data$b0014==9 | data$b0015==9 | data$b0016==9 |
46                    data$b0017==9 | data$b0018==9 | data$b0019==9 |
47                    data$b00110==9 | data$b00111==9 | data$b00112==9
                          |
48                    data$b00113==9),] #180.517 obs
49
50 data <- subset(data, select=c(a002,a003,a004,a005,a006,a006b,
51                                b0011,b0012,b0013,b0014,b0015,b0016,
52                                b0017,b0018,b0019,b00110,b00111,b00112
                                    ,
53                                b00113,covid))
54
55 data <- subset(data, select=-c(a006,a006b))
56
57 names(data) <- c("age", "sex", "race", "edu", "fever", "cough", "
```

```
    sorethroat",
58                  "breathless", "headache", "chestpain", "nausea", "
                        runny",
59                  "tiredness", "eyepain", "senseloss", "musclepain",
60                  "diarrhea", "covid")
61
62
63 write.csv(data, "Databases/covid_dados_editada.csv")
```

Referências bibliográficas

[Base dos Dados 2022] DAHIS, R.. **Base dos dados**, 2022. [Online em <https://basedosdados.org/dataset/br-ibge-pnad-covid>; acessado em 14 de Junho de 2022].

[Branco et al. 2015] BRANCO, P.; TORGO, L. ; RIBEIRO, R. P.. **A survey of predictive modelling under imbalanced distributions**. CoRR, abs/1505.01658, 2015.

[Breiman 1996] BREIMAN, L.. **Bagging predictors**. Machine Learning, 24(2):123–140, 1996.

[Breiman et al. 1983] BREIMAN, L.; FRIEDMAN, J.; J. STONE, C. ; OLSHEN, R. A.. **Classification and Regression Trees**. Chapman and Hall/CRC, 1984.

[Buntine 1991] BUNTINE, W.. **Learning classification trees**, 1991.

[Corsil et al. 2008] CORTEZ, P.; SILVA, A. M. G.. **Using data mining to predict secondary school student performance**, 2008. [Dados disponíveis online em <https://data.world/data-society/student-alcohol-consumption>; acessado em 14 de Junho de 2022.].

[Cunningham et al. 2007] CUNNINGHAM, P.; DELANY, S.. **k-nearest neighbour classifiers**. Mult Classif Syst, 54, 04 2007.

[Friedman 2001] FRIEDMAN, J. H.. **Greedy function approximation: a gradient boosting machine**. Annals of statistics, p. 1189–1232, 2001.

[Guo et al. 2003] GUO, G.; WANG, H.; BELL, D.; BI, Y. ; GREER, K.. **Knn model-based approach in classification**. In: Meersman, R.; Tari, Z. ; Schmidt, D. C., editors, ON THE MOVE TO MEANINGFUL INTERNET SYSTEMS 2003: COOPIS, DOA, AND ODBASE, p. 986–996, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Hastie et al. 2001] HASTIE, T.; TIBSHIRANI, R. ; FRIEDMAN, J.. **The Elements of Statistical Learning**. Springer, New York, NY, USA, 2001.

[Ismael et al. 2018] ISMAEL, M. R.; ABDEL-QADER, I.. **Brain tumor classification via statistical features and back-propagation neural network**. In: 2018 IEEE INTERNATIONAL CONFERENCE ON ELECTRO/INFORMATION TECHNOLOGY (EIT), p. 0252–0257, 2018.

[James et al. 2013] JAMES, G.; WITTEN, D.; HASTIE, T. ; TIBSHIRANI, R.. **An Introduction to Statistical Learning: with Applications in R**. Springer, 2013.

[John 2020] JOHN, C. R.. **Mleval**, 2020. [Online; acessado em 30 de Maio de 2022].

[Kuhn 2008] KUHN, M.. **Building predictive models in r using the caret package**. Journal of Statistical Software, 28(5):1–26, 2008.

[Kuhn 2019] KUHN, M.. **The caret package documentation**, 2019. [Online; acessado em 09 de Junho de 2022].

[Liang et al. 2016] LIANG, D.; LU, C.-C.; TSAI, C.-F. ; SHIH, G.-A.. **Financial ratios and corporate governance indicators in bankruptcy prediction: A comprehensive study**. European Journal of Operational Research, 252(2):561–572, 2016. [Dados disponíveis online em <https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction>; acessado em 14 de Junho de 2022.].

[Louzada et al. 2016] LOUZADA, F.; ARA, A. ; FERNANDES, G. B.. **Classification methods applied to credit scoring: Systematic review and overall comparison**. Surveys in Operations Research and Management Science, 21(2):117–134, 2016.

[Rosset 2004] ROSSET, S.. **Model selection via the auc**. In: PROCEEDINGS OF THE TWENTY-FIRST INTERNATIONAL CONFERENCE ON MACHINE LEARNING, ICML '04, p. 89, New York, NY, USA, 2004. Association for Computing Machinery.

[Sun et al. 2011] SUN, Y.; WONG, A. ; KAMEL, M. S.. **Classification of imbalanced data: a review**. International Journal of Pattern Recognition and Artificial Intelligence, 23, 11 2011.

[Wolpert 1996] WOLPERT, D. H.. **The lack of a priori distinctions between learning algorithms**. Neural Computation, 8(7):1341–1390, 1996.

[Yang et al. 2020] YANG, L.; LYU, K.; LI, H. ; LIU, Y.. **Building climate zoning in china using supervised classification-based machine learning**. Building and Environment, 171:106663, 2020.