Lucas Meireles Tomaz de Alvarenga

Reconhecendo números e seus padrões

Niterói - RJ, Brasil

17 de Março de 2016

Universidade Federal Fluminense

Lucas Meireles Tomaz de Alvarenga

Reconhecendo números e seus padrões

Trabalho de Conclusão de Curso

Monografia apresentada para obtenção do grau de Bacharel em Estatística pela Universidade Federal Fluminense.

Orientadora: Profa. Jessica Quintanilha Kubrusly

Niterói - RJ, Brasil

17 de Março de 2016

Universidade Federal Fluminense

Lucas Meireles Tomaz de Alvarenga

Reconhecendo números e seus padrões

Monografia de Projeto Final de Graduação sob o título *"Reconhecendo números e seus padrões"*, defendida por Lucas Meireles Tomaz de Alvarenga e aprovada em 17 de Março de 2016, na cidade de Niterói, no Estado do Rio de Janeiro, pela banca examinadora constituída pelos professores:

Profa. Dra. Jessica Quintanilha Kubrusly
Orientadora
Departamento de Estatística – UFF

Prof. Dr. Jony Arrais Pinto Junior
Departamento de Estatística – UFF

Prof. Dr. Wilson Calmon Almeida dos Santos
Departamento de Estatística – UFF

Niterói, 17 de Março de 2016

A 473 Alvarenga, Lucas Meireles Tomaz de
Reconhecendo números e seus padrões/Lucas Meireles Tomaz de Alvarenga. - Niterói: [s. n.], 2016.
43f.

Trabalho de Conclusão de Curso – (Bacharelado em Estatística) – Universidade Federal Fluminense, 2016.

1. Estatística não-paramétrica. 2. Regressão (Estatística). 3. Análise matemática. 4. Processo estocástico. 5. Inteligência artificial. 6. Análise numérica. I. Título.

Resumo

Nesse trabalho, nós aplicamos técnicas que servem para o reconhecimento de imagens, essa tarefa faz parte do campo de estudo de Inteligência Artificial. O objetivo desse trabalho é comparar métodos de classificação quando aplicados no reconhecimento de números manuscritos, cada um desses métodos será feito com e sem a Análise de Componentes Principais. Os métodos que serão utilizadas nesse trabalho são: *Random Forest*, *K-Means* e *k-Nearest Neighbor*. Ao longo desse trabalho, vimos que a técnica que consegue uma melhor performance é a *Random Forest* , mas com um tempo de execução muito longo. Outra técnica que consegue uma alta taxa de assertividade é o *k-Nearest Neighbor*, chegando a fazer somente $8,54\%$ de classificações erradas. Também vimos que com o uso de Análise de Componentes Principais é possível reduzir em muito o tempo de execução dos métodos, sem que tenha uma perda grande de assertividade.

Dedicatória

Dedico este trabalho ao Futuro, porque não tem nada tão incerto e animador como ele.

Agradecimentos

Agradeço primeiramente a minha orientadora, que me aturou ao longo do projeto final. Ao Jony e ao Wilson, por terem aceitado fazer parte da minha banca de avaliação. A minha família, por não terem ligado pelo meu caminho ter sido muito mais longo do que precisava. Agradecimento especiais ao Professor Chico e ao Professor José Rodrigo vocês são ótimos. Também quero agradecer ao Bruno, Cissa, Paulista, Acelerado, Nadine, Pablo, Moura, Luciana, Rafael Roger, Tiago de Sales, Rosana, Barrientos, Bertinho, Albano, Veve, W, Ralado, Beto, Fernando, Igor, Paola, Vitor Miranda, JP, Cidade, Rapha e China, por terem conhecido vocês. Por fim, agradecer a UFF por proporcionar todos esses momentos.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

A melhor maneira de introduzir esse trabalho é falando de um filme, O Jogo da Imitação. Nele, temos que o governo britânico precisa decifrar o meio de comunicação nazista, que todo dia é uma combinação diferente. As maiores mentes são reunidas para decifrarem esse código, mas não era humanamente possível traduzir mais do que algumas palavras antes do fim do dia. A partir daí montam a primeira maquina para fazer o processamento da informação e reconhecer os padrões da comunicação alemã. Esse filme é importante para esse trabalho porque conta a história de Alan Turing, considerado por muitos o pai da inteligência artificial. A inteligência artificial é o campo de estudo voltado para que uma maquina, ou software, possa fazer uma tomada decisões que maximizem a chance de sucesso. Uma forma de fazer isso é pelo reconhecimento de padrões.

Como Alan Turing utilizou o reconhecimento de padrões para decifrar a criptografia alemã na segunda guerra mundial, é possível utilizá-la para diversas outras atividades, como, por exemplo, o reconhecimento de voz. Alguns atendimentos telefônicos de banco, já utilizam esse reconhecimento de voz para que você possa passar todas as suas informações por voz e diretamente para uma máquina. No nosso caso, vamos utilizar o reconhecimento de padrões para reconhecer números manuscritos por pessoas.

Essa tarefa de reconhecimento de números manuscritos, tem como principal dificultador que são pessoas reais que estão escrevendo o número, não saem de uma forma idêntica, variando assim tanto a inclinação na qual o número é escrito, quanto a forma e o tamanho. Outra dificuldade que essa tarefa encontra, é a semelhança entre alguns números, muita vezes até nós nos confundimos se um número é o “4”ou o “9”, dependendo de quem escreveu. No nosso trabalho existe um facilitador, os números estão escrito em branco com um fundo preto, ou seja, não vamos ter problemas com fundos e números escritos de diferentes cores. Para que essa tarefa seja completa, nós vamos utilizar métodos de classificação.

Na seção a seguir, apresentamos o objetivo deste trabalho. Na Seção 3 introduzimos

os materiais e métodos que irão ser aplicados neste trabalho. Após a seção de materiais e métodos, apresentamos a seção de resultados, onde será apresentado o resultado das aplicações dos métodos. Por fim, teremos a seção de conclusão, onde será apresentado a conclusão final sobre os resultados.

2 Objetivo

O objetivo desse trabalho é comparar diversas técnicas estatísticas quando utilizadas para reconhecer números manuscritos. Tais técnicas serão comparadas a fim de destacar vantagens e desvantagens de cada uma delas. Também será feita a comparação dessas técnicas com e sem o método de análise de Componentes Principais. As técnicas estatísticas que serão estudadas neste trabalho são: *K-means*, *Random Forest* e *K-Nearest Neighbor*.

3 Materiais e Métodos

Nesse capítulo iremos apresentar o banco de dados, explicar a forma como as imagens são apresentadas, introduzir as técnicas que iremos utilizar nesse trabalho e apresentar como faremos as comparações dos modelos.

3.1 Materiais

O banco de dados que vamos utilizar foi retirado do site Kaggle[1][2], ele é uma modificação do banco oferecido pelo *National Institute of Standards and Technology* [1]. Essa base que utilizamos traz a informação de 42.000 números manuscritos, vamos nos referir a cada uma dessas informações como unidades experimentais. Na Figura 1 vemos exemplos dos números manuscritos, retirados diretamente desse banco de dados. Podemos perceber que alguns números podem estar escrito com inclinação, por exemplo, o número 1.

Figura 1: Números Manuscritos

Para que possamos trabalhar com imagens é preciso que o banco de dados contenha

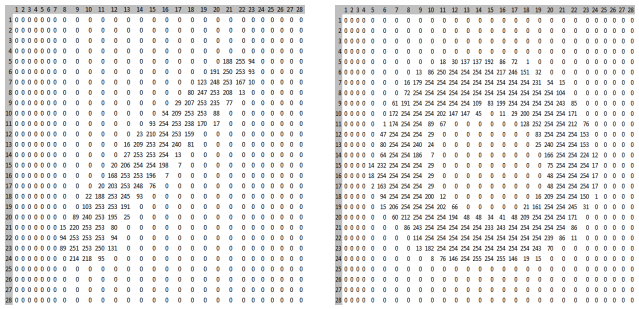
[1]Esse banco de dados estava disponível em `https://www.kaggle.com/c/digit-recognizer/data` em fevereiro de 2016.

[2]O Site do Kaggle, é um site voltado para a competição de modelos matemáticos, para que a pessoa possa treinar seus conhecimentos ou até mesmo competir por dinheiro.

as informações da cor de cada pixel. No nosso banco, cada imagem virá representada por um vetor contendo as informações de cores, essa cor vem descrita de forma numérica e padronizada. O processo de transformação de uma imagem para um vetor será visto nas seções a seguir.

3.1.1 Transformando a imagem para um vetor

No nosso banco, cada imagem tem o tamanho de 28×28 pixels, ou seja, temos para cada imagem 784 informações de cores. Podemos pensar que essa imagem pode ser interpretada como uma matriz de mesmo tamanho, onde cada posição de linha i e coluna j guarda um número referente a sua cor, em uma escala de cinza, que varia de 0 até 255. Na Figura 2 podemos ver os dois primeiros números, do nosso banco de dados, na sua forma matricial.



(a) Número 1 (b) Número 0

Figura 2: Números Manuscritos em forma de matriz numérica

3.1.2 Banco de dados

Cada imagem, após ser transformada em matriz numérica, será modificada para um vetor onde na primeira posição estará o pixel (1,1) e na septuagésima oitava quarta estará o pixel (28,28). Dessa maneira, cada imagem será representada em uma linha desse banco. No total, o banco de dados possui 785 colunas, 784 dessas colunas são compostas por variáveis numéricas correspondentes ao vetor de cores dessa imagem e a coluna restante é a variável categórica, que guarda um número de 0 até 9 que indica a classe, em outras palavras, o número real da imagem em questão.

3.1.3 Notação para as seções de Metodologia

Nas seções a seguir serão apresentadas diferentes metodologias que buscam reconhecer os números manuscritos a partir do banco de dados descrito nesta seção. Para isso é preciso definir algumas notações usadas ao longo deste capítulo.

Considere de forma genérica um banco de dados, chamado de treino, com N unidades experimentais, M variáveis numéricas e uma variável categórica, chamada de classe. Seja A uma matriz $N \times M$ tal que a posição (i, j) dessa matriz é o valor da j-ésima variável para a i-ésima unidade experimental.

3.2 Análise de Componentes Principais

Segundo Johnson e Wichern [2] o objetivo do método de Análise de Componentes Principais (*Principal Components Analysis - PCA*) é explicar a variância e covariância de um conjunto de variáveis utilizando combinações lineares destas variáveis, com isso, é possível selecionar as variáveis pela sua importância e chegar em um resultado com uma boa qualidade fazendo o uso de um número menor de variáveis. Pelo nosso banco de dados possuir um grande número de variáveis, queremos comparar a acurácia dos métodos sem e com *PCA*, por isso iremos utilizar essa técnica para redimensionar o nosso banco de dados.

Antes de explicar como esse método funciona é preciso passar por dois conceitos: autovetor e autovalor.

Definição 3.2.1 *Seja $v \in \mathbb{R}^M$ um vetor não nulo e B uma matriz quadrada $M \times M$. O vetor v é um autovetor de B se $Bv = \lambda v$ para algum $\lambda \in \mathbb{R}$. Nesse caso λ é chamado de autovalor de B associado ao autovetor v.*

3.2.1 Por dentro do *PCA*

Seja $\mathbf{X} = (X_1, X_2, ..., X_M)^\top$ um vetor aleatório e Σ a sua matriz de covariância. Sejam $e_1, e_2, ..., e_M$ autovetores de Σ associados aos autovalores $\lambda_1, \lambda_2, ..., \lambda_M$ de forma que $\lambda_1 > \lambda_2 > ... > \lambda_M$. Como Σ é uma matriz simétrica positiva definida, podemos afirmar que existem M autovetores linearmente independentes e M autovalores reais positivos, para a demonstração desse resultado veja a Seção 2.3 do livro de Johnson [2].

Seja

$$Y_i = e_i{}^\top \mathbf{X} = e_{i1}X_1 + e_{i2}X_2 + ... + e_{iM}X_M,$$

para $i = 1, 2, ..., M$. Cada variável aleatória do vetor $\mathbf{Y} = (Y_1, Y_2, ..., Y_M)^\top$ é chamada de componente principal, com $\text{Var}(Y_i) = \lambda_i$ e $\text{Cov}(Y_i, Y_k) = 0$, para $k = 1, 2, ..., M$ e $k \neq i$.

A proporção da variação total explicada pelo i-ésimo componente principal é dada por

$$p_i = \frac{\lambda_i}{\lambda_1 + \lambda_2 + ... + \lambda_M}. \tag{3.1}$$

Ou seja, o primeiro componente é aquele que mais explica a variação total. O método nos permite trabalhar com $\mathbf{Y}$ em vez de $\mathbf{X}$, dessa forma podemos descartar os componentes que pouco explicam o modelo para reduzir a sua dimensão, sem perder as informações significantes agregadas a cada variável.

3.2.2 Desenvolvendo o *PCA*

Utilizando como base de raciocínio o banco descrito na Subseção 3.1.3, seja A uma matriz $N \times M$, cuja colunas guardam uma amostra de tamanho N de cada variável X_j, para $j = 1, 2, ..., M$. Seja S a matriz de covariância amostral, das variáveis aleatórias X_i, dada por $S = A^\top A$. Sejam $v_1, v_2, ..., v_M$ autovetores de S associados aos autovalores $\lambda_1 > \lambda_2 > .... > \lambda_M$.

Temos que a nova matriz de dados, com amostras de Y_i é definida por

$$\tilde{A} = AP,$$

sendo P uma matriz $M \times m$ cuja colunas são os autovetores $(v_1, v_2, ..., v_m)$ escolhidos, para $m \leqslant M$. A escolha de m é feita a partir da variação total acumulada por $v_1, v_2, ..., v_m$.

Quando quiser implementar uma nova unidade experimental no banco de dados, será necessário que esta seja transformada de forma a ter o mesmo número de colunas que as variáveis do banco. Seja $a^\top$ a nova unidade experimental, a sua transformação será feita por: $\tilde{a}^\top = a^\top P$.

3.3 K-means

Nessa seção vamos apresentar o método de agrupamento *K-means*, esse método é conhecido como uma técnica não supervisionada, por não fazer o uso de uma variável

resposta.

Métodos de agrupamento, também podem ser chamadas de métodos de agrupamentos, tem como objetivo agrupar as unidades experimentais de uma forma que cada grupo seja mais semelhante entre si do que com alguma variável de outro grupo. Existem diversas técnicas que fazem análise de *cluster*, mas como dito anteriormente, para esse trabalho iremos utilizar o *K-means*. Cada *cluster* construído utilizando essa técnica, são agrupamentos que tem como semelhança a menor distância entre o ponto médio do *cluster* e a posição de suas unidades experimentais.

Para continuarmos a explicar o *K-means* é necessário considerar um banco de dados com N unidades experimentais e com M variáveis numéricas. O método irá particionar esse banco de dados em k *clusters*, sendo k um número fixo escolhido por quem estiver utilizando o método. No livro de Johnson e Wichern [2] e no artigo de Wagstaff et. al [3] os autores explicam esse processo de partição em 3 passos, para melhor entendimento neste trabalho irei explicar o processo em 4 passos:

1. Escolher k centroides[3]: $C_1, C_2, \ldots, C_k$. Essa escolha pode ser feita de forma aleatória ou não.

2. Particionar o espaço $\mathbb{R}^M$ em k *clusters*, cada um associado a um centroide. O *cluster* associado ao centroide C_i é definido pelos pontos mais próximos de C_i do que de qualquer outro centroide. Para isso normalmente é usada a distância euclidiana.

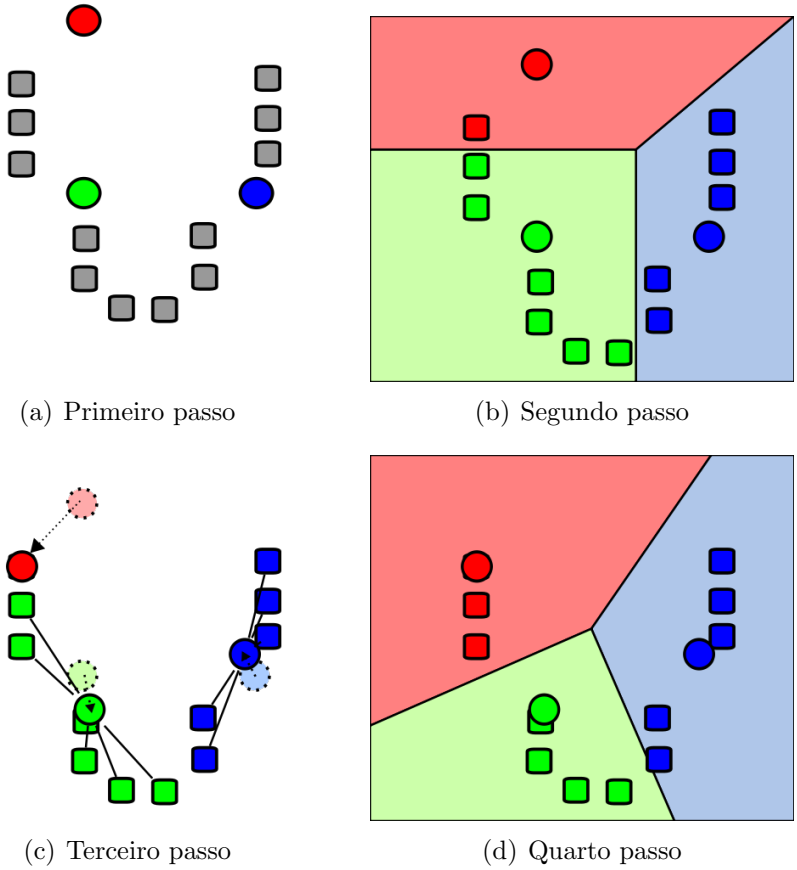
3. Recalcular os centroides de forma que o novo centroide de cada *cluster* passe a ser a média dos pontos que pertencem a ele.

4. Voltar para o passo 2 até que não haja mais alterações nos pontos que compõe cada *cluster*.

Como explicado anteriormente, k é um número escolhido por quem estiver utilizando o método. Como neste trabalho estamos falando de dígitos e sabemos que no máximo teremos 10 diferentes possibilidades de números, vamos escolher para que k seja igual a 10. Vale ressaltar que para o banco desse trabalho, o agrupamento será feito no espaço $\mathbb{R}^{784}$, pois cada unidade experimental é formada por 784 variáveis, desconsiderando a coluna de label. A seguir, na Figura 3, temos uma ilustração[4] no espaço $\mathbb{R}^2$, com 12 unidades experimentais e com o valor do k igual a 3.

[3]Ponto no espaço $\mathbb{R}^M$, que será o centro do *cluster*

[4]A Figura 3 estava disponível em `http://en.wikipedia.org/wiki/K-means_clustering` em maio de 2015.



(a) Primeiro passo (b) Segundo passo

(c) Terceiro passo (d) Quarto passo

Figura 3: Exemplo do processo de K-means

Podemos observar que na Figura 3(a) o centroide foi gerado de maneira aleatória, não sendo preciso coincidir com a posição de alguma das unidades experimentais. Na Figura 3(b) vemos que já foram feitas as divisões dos pontos em *clusters*, formados cada um pelos pontos mais próximos de cada centroide. Agora na Figura 3(c) os centroides são recalculados, tendo como novo centroide o ponto médio de todos os pontos que pertencem ao *cluster*. No grupo vermelhos o ponto do novo centroide se encontra exatamente onde estava a única unidade experimental desse *cluster*, isso acontece porque o centroide não é utilizado para fazer a conta do novo ponto médio. Por fim, na quarta figura 3(d) é possível ver que todo processo já foi repetido até não ser possível mais realocar os pontos.

Para fazermos a previsão de uma nova unidade experimental seria preciso ver a que *cluster* essa unidade pertence e depois disso, a moda da categoria daquele *cluster*.

3.4 *Random Forest*

Nessa seção vamos apresentar o método de classificação *Random Forest*, mas para conseguirmos criar um bom entendimento desse método primeiro teremos que explicar o método Árvore de Classificação.

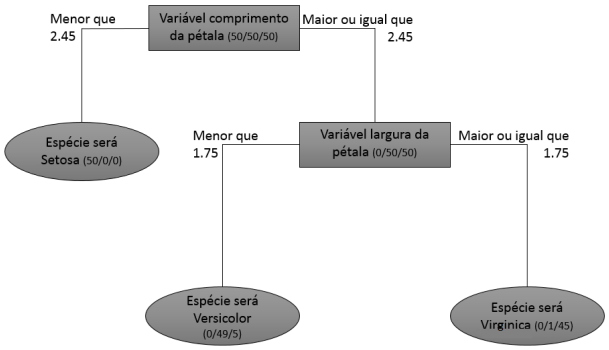
Considere um banco de dados, chamado de treino, de tamanho N formado por M variáveis numéricas e uma variável categórica, chamada de classe. O objetivo dos dois métodos apresentados a seguir é a partir desse banco de treino criar uma estrutura, Árvore de Classificação ou Random Forest, capaz de prever a classe de uma nova unidade experimental que possui M variáveis.

3.4.1 Árvore de Classificação

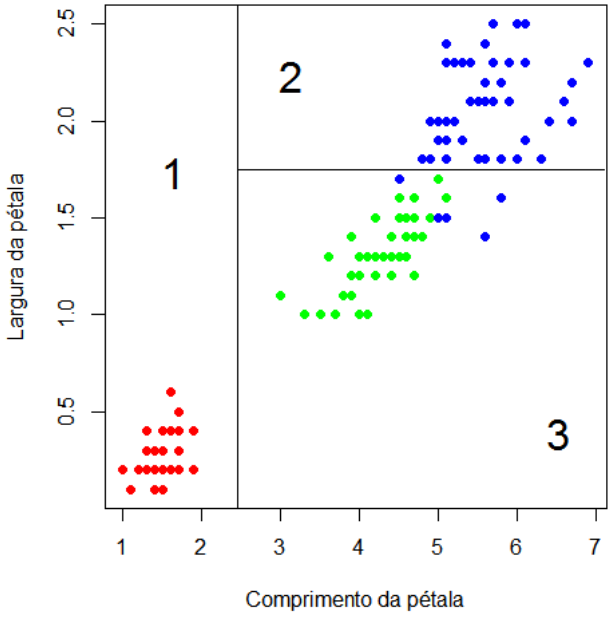
O autor Quinlan [4] usa o nome de Árvore de Decisão para se referir a tal método, outros autores utilizam Árvore de Regressão, como nós estamos utilizando esse método para fazer uma classificação sempre usaremos Árvore de Classificação para se referir a esse método. Esse método utiliza de um algoritmo que busca particionar o espaço $\mathbb{R}^M$, a fim de isolar áreas que tenham uma concentração maior de uma única classe. Veja a seguir um exemplo aplicado no espaço $\mathbb{R}^2$:

Nesse exemplo foi usada informações do banco de dados Iris, que tem informações sobre pétalas de 3 espécies diferentes: *Setosa*, *Versicolor* e *Virginica*. A amostra utilizada nesse exemplo tem tamanho 150 e cada espécie representa um terço da amostra. Na Árvore de Classificação 4(a) a informação que se encontra entre parenteses é a quantidade da espécie que está concentrada naquela parte. No gráfico 4(b) os números em caixa alta são referentes as divisões feitas na Árvore de Classificação. A parte onde ocorre a separação do espaço $\mathbb{R}^2$ é conhecida como nó.

O processo de partição do $\mathbb{R}^M$ ocorre da seguinte forma: Em cada nó é selecionada uma variável x_i para particionar os pontos em dois conjuntos, $x_i < k$ e $x_i \geq k$; Tanto a variável x_i quanto o valor k são escolhidos da forma que melhor se faça a divisão das classes; Um vez feita a divisão em um nó, o processo anterior é repetido nessas novas partições. O processo termina quando não há mais possível particionar o nó em questão ou até se chegar em algum critério de parada, como por exemplo, a profundidade da árvore. Essa parte da árvore, que não se ramifica, é conhecida como folha; Após o processo ter chegado ao fim, pode ser feita uma etapa de poda, no qual é possível retirar nós da árvore para diminuir o seu tamanho.



(a) Árvore de Classificação



(b) Gráfico de dispersão

Figura 4: Exemplo de Árvore de Classificação

Uma vez a estrutura da árvore construída, para realizarmos uma previsão, isto é, descobrir a que classe um novo valor pertence fazemos com que ele percorra os nós da árvore começando pela raiz (primeiro nó) e seguindo os passos que foi explicado anteriormente. Será atribuído ao novo valor a classe da folha em que ele chegar. Por exemplo, se quisermos usar a árvore da Figura 4 para prever a espécie de uma flor com comprimento de pétala 4 e largura de pétala 10 teremos como reposta que essa flor é da espécie *Versicolor*.

3.4.2 *Random Forest*

O nome *Random Forest* vem do fato de ser uma técnica que utiliza um conjunto de Árvores de Classificações. Para explicar como funciona o processo, que teve como base o artigo de Breiman [5], é importante lembrar que N representa a quantidade de unidades experimentais e M o número de variáveis.

Serão construídas n árvores de decisões, para compor o *Random Forest*. Quanto maior o número de variáveis no banco de dados, maior deve ser o n.

A construção de cada uma destas árvores, que farão parte do *Random Forest*, será realizado com o seguinte processo:

- É selecionado de forma aleatória e com reposição uma amostra de tamanho N do banco de dados original;
- Para cada nó da árvore será escolhido m variáveis aleatoriamente, sendo m um número menor do que M;
- A árvore irá crescer até o máximo possível, ou seja, só irá parar quando não for mais possível particionar o nó, ou atingir algum critério de parada;
- Não existe a fase de poda quando estamos tratando de *Random Forest*.

Como podemos ver no processo explicada, as variáveis são escolhidas aleatoriamente, por isso foi recomendado que quanto maior o número de variáveis no banco de dados, maior deve ser o n. Para que dessa forma, não ocorre o risco de não termos alguma variável que não seja utilizada, ou que seja utilizada pouquíssimas vezes.

Os valores de m e o número de árvores são determinados antes do início do processo. No artigo de Breiman [5], para a escolha do tamanho de m ele utiliza $\log_2 M + 1$, porque se m for um número grande aumenta a correlação entre as árvores, o que aumenta a taxa de erro. Em contra partida, se escolher um valor muito pequeno para m, esse pode aumentar o erro individual de cada árvore, sendo assim a raiz de M é um ponto de equilíbrio entre esses dois fatos. Para descobrirmos a que classe pertence um novo valor será preciso que ele percorra todas ás árvores de classificação seguindo o caminho visto na seção 3.4.1, ou seja, a classe que mais for comum para esse valor dentre todas as árvores será a classe que será atribuído a ele.

3.5 *k-Nearest Neighbor (k-NN)*

k-NN é um método que pode ser utilizado tanto para a classificação quanto para a regressão. O objetivo desse método, quando utilizado para a classificação, é classificar a unidade experimental por intermédio dos seus vizinhos mais próximos, ou seja, a unidade experimental irá receber a classe mais comum entre os k vizinhos mais próximos, para k um inteiro positivo.

A escolha de k é crítica, porque para um k pequeno qualquer ruído pode ter grande influencia e para um k muito grande estaríamos indo contra o objetivo do método, que é classificar unidades experimentais por semelhança aos seus vizinhos mais próximos. Uma forma de escolher k é igualando ele a raiz da quantidade de unidades experimentais, como Duda et al. [6] sugere em seu livro.

Seja A a matriz $N \times M$ definida na Subseção 3.1.3. Cada linha dessa matriz é uma unidade experimental em $\mathbb{R}^M$ associada a uma classe. Para classificar uma nova unidade experimental a a partir do método *k-NN*, buscamos as k unidades experimentais mais próximas de a e suas respectivas classes. Atribuímos como classe de a a classe predominante entre as encontradas. Podemos também aferir peso para as classes, uma maneira de fazer isso é criar um peso proporcional ao inverso da distância entre a e as unidades experimentais. Em geral a distância utilizada é a euclidiana, mas essa não é a única medida possível de distância.

Na ilustração[5] a seguir, temos um exemplo de como a escolha de k é crucial:

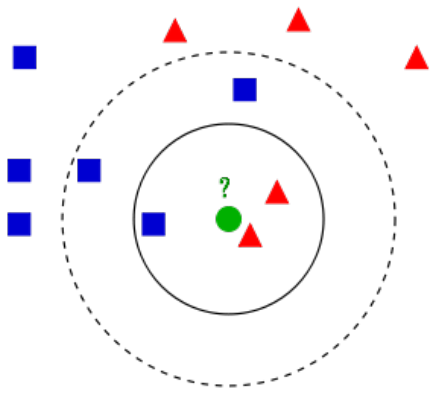


Figura 5: Exemplo do efeito da escolha do k

Podemos notar que na Imagem 5 temos pontos distribuídos no espaço $\mathbb{R}^2$, com duas classes diferentes e uma nova unidade experimental que não sabemos a classe. Para

[5] A Figura 5 estava disponível em `https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm` em junho de 2015.

$k = 3$, representado pelo circulo interior não tracejado, podemos notar que ele a nova unidade receberia triangulo como sua classe. Se escolhermos $k = 5$, representado pelo circulo tracejado, e não considerarmos nenhuma técnica de atribuição de peso, a nossa nova unidade experimental receberia quadrado como sua classe. Embora essa imagem faça com que o método pareça muito volátil, não podemos esquecer que esse exemplo tem um número pequenos de varáveis com a classe conhecida e no livro do Duda et al. [6] é demonstrado que quando o tamanho da amostra tende para o infinito e k é um número fixo, todos os k números vão convergir para a classe correta.

Uma leitura mais leve, porém mais informal, que gostaríamos de referenciar sobre esse método é a do site do Saravanam[6], ele condensa bem as informações do livro do Duda et al. de uma maneira mais prática.

3.6 Validação Cruzada

Normalmente para analisar a qualidade do modelo, a base de dados é separada em base de teste e base de treino. A primeira parte é utilizado para rodar o modelo e a outra para testar a qualidade. Nós vamos fazer o uso de uma técnica conhecida como Validação Cruzada, ela segue o mesmo principio, mas sem o ônus de perder uma parte da base para teste definitivamente. A proposta é dividir sua base em l pedaços, juntar $l - 1$ para compor a base de treino e a base restante será a base de teste. Esse processo terá que ser feito l vezes, em cada vez um dos l pedaços, que ainda não foram base de teste, terá que ser a base de teste. Para cada ciclo nós rodaremos o modelo na base de treino e faremos a classificação do modelo na base de teste. Considerando a classificação de cada uma das bases de teste teremos a classificação da base total.

No nosso trabalho foi escolhido $l = 10$, como cada pedaço deve ser composto de forma aleatória, nós fizemos uma amostragem estratificada sem reposição para criarmos cada um dos pedaços. Na Tabela 1 temos como ficou a distribuição nos l_i pedaços, sendo $i = 1, 2, 3..., 10$. Por exemplo, no primeiro pedaço, l_1, temos que a base possuí 413 "0"e na coluna total é o número na base completa, ou seja, temos 4132 "0"na base completa.

[6] `https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/`

Tabela 1: Divisão da base

	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	Total
0	413	413	413	413	414	414	413	413	413	413	4132
1	468	469	468	468	469	468	468	469	468	469	4684
2	418	418	417	417	417	418	418	418	418	418	4177
3	435	435	435	436	435	435	435	435	435	435	4351
4	407	408	407	407	407	407	407	408	407	407	4072
5	379	379	380	379	379	379	380	380	380	380	3795
6	413	413	414	414	414	414	413	414	414	414	4137
7	440	440	440	440	440	440	440	441	440	440	4401
8	406	406	406	406	407	406	407	406	407	406	4063
9	419	419	419	419	418	419	418	419	419	419	4188
Total	4198	4200	4199	4199	4200	4200	4199	4203	4201	4201	42000

3.7 Medidas de Qualidade

Nas próximas seções serão apresentados os resultados e teremos algumas tabelas com algumas medidas de qualidade para podermos fazer as comparações. Para exemplificar as medidas, foi construída a Tabela 2, onde $a_{i,j}$ é um inteiro positivo, sendo $i = 1, 2, \ldots, 9$ e $j = 1, 2, \ldots, 9$.

Tabela 2: Exemplo das medidas de qualidade.

Classe \ Previsão	0	1	...	9
0	$a_{0,0}$	$a_{0,1}$	$\cdots$	$a_{0,9}$
1	$a_{1,0}$	$a_{1,1}$	$\cdots$	$a_{1,9}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
9	$a_{9,0}$	$a_{9,1}$	$\cdots$	$a_{9,9}$

3.7.1 Base de treino

Na base de treino vamos apresentar análises descritivas do tempo, em minutos, que cada iteração da Validação Cruzada demorou para a construção do modelo. Também teremos a taxa de erro total de cada iteração, agrupada de uma forma descritiva. Essa taxa de erro total é calculado por:

$$1 - \frac{\sum_{i=0}^{9} a_{i,i}}{\sum_{i=0}^{9}\sum_{j=0}^{9} a_{i,j}}, \tag{3.2}$$

seguindo a Tabela 2. Em outra palavras, a diagonal principal dividida pelo total de unidades experimentais.

3.7.2 Base de teste

Na base de teste temos três medidas de qualidade, duas delas calculadas por classe e a taxa de erro total. Também é usada a taxa de erro total, como apresentada na Equação 3.2. As outras duas medidas são:

Taxa de Falso Negativo (TxFN) - Quantidade de classificações erradas da classe de interesse, pelo total de unidades experimentais dessa classe. Se fossemos calcular essa medida para a classe "i", seria utilizada a seguinte equação:

$$\frac{(\sum_{j=0}^{9} a_{i,j}) - a_{i,i}}{\sum_{j=0}^{9} a_{i,j}}. \tag{3.3}$$

Taxa de detecção Falsa (TxDF) - Quantidade de previsões erradas do número de interesse, pelo total de vezes que esse número foi previsto.Se fossemos calcular essa medida para a previsão "j", seria utilizada a seguinte equação:

$$\frac{(\sum_{i=0}^{9} a_{i,j}) - a_{j,j}}{\sum_{i=0}^{9} a_{i,j}}. \tag{3.4}$$

4 Resultados

Nesse capítulo iremos explicar quais métricas foram utilizados para medir a performance do modelo e iremos apresentar os resultados de cada uma das técnicas, sem e com *PCA*. Vale ressaltar que todos as técnicas foram executadas na mesma máquina, para não haver diferença no tempo de execução por questão de desempenho de computador.

4.1 Métodos com *PCA*

Como todos os métodos serão rodados com e sem o *PCA*, resolvemos criar essa seção para explicar como foi feito esse processo. Primeiro nós fizemos a Validação Cruzada, separando a base como está na Tabela 1. Em cada uma das 10 interações, antes de rodar o método, o *PCA* foi aplicado na base de treino e foram selecionadas o número de variáveis até o acumulo da proporção da variação total chegar a 95%. O número de variáveis que ficaram em cada uma das interações pode ser visto na Tabela 3.

Tabela 3: Número de variáveis na base de treino após o *PCA*, por iteração.

iteração	Variáveis restantes na base de treino
1	153
2	153
3	154
4	153
5	153
6	153
7	153
8	153
9	153
10	153

4.2 *Random Forest*

Vale ressaltar que para construirmos as medidas de qualidade nós utilizamos a Validação Cruzada, que falamos na Seção 3.6. Como adiantamos na Seção 3.7, na Tabela 4 podemos ver um resumo do tempo de execução do modelo na base de treino. O tempo mínimo (Min) é o melhor tempo de execução em uma das interações, enquanto o máximo (Max) é o pior tempo. Temos também o primeiro, o segundo e o terceiro quartil, essas medidas nos ajudam ver melhor como está a dispersão entre todos os tempos de execução na base. Podemos ver, que teve iteração que demorou mais de 43 minutos para o modelo rodar, sendo que a média de todas as interações é de aproximadamente 40 minutos. Totalizando em mais de 6 horas de duração, somente para o *Random Forest*. Nessa mesma Tabela 4 temos o percentual de erro da base de treino, ele segue a mesma formatação que o tempo de execução. Temos que nosso erro máximo foi de $3,81\%$ e a média ficou em $3,48\%$, lembrando que essa medida de erro é calculada pela Equação 3.2. Vale ressaltar, que não temos uma variação tão grande entre os erros, isso é um bom sinal para o modelo.

Tabela 4: Resumo da performance do Random Forest nas bases de Treino

	Min	1º Qu.	Mediana	Média	3º Qu.	Max
Tempo de execução (min)	36,62	37,28	40,48	40,12	42,22	43,92
Erro (%)	3,07	3,33	3,52	3,48	3,70	3,81

Seguindo para a avaliação da qualidade na base de teste, podemos ver a Tabela 5, as linhas são as classe e as colunas são as classificações do modelo, por exemplo, se formos olhar para a segunda coluna e para a segunda linha, temos que 4078 "0"foram realmente classificados como "0". Na Tabela 6 temos as medidas de qualidade Taxa de Falso Negativo (TxFN) e Taxa de Detecção Falsa (TxDF), como explicados na Seção 3.7.

A taxa de erro total do *Random Forest* foi de $3,49\%$. Mesmo com esse valor de erro baixo, podemos perceber pelas outras medidas de qualidade que temos problemas em classificar certas classes. Como por exemplo o número "9", teve uma Taxa de Falso Negativo de $5,44\%$ e a Taxa de Detecção Falsa foi de $5,65\%$. Ou seja, além do número "9"ser o pior número para ele classificar é o número que ele mais se confunde quando vai classificar outras classes, esse prolema está mais concentrado no número "4". Em contrapartida, com o número "9", temos o número "0"com uma Taxa de Falso Negativo de $1,31\%$ e uma Taxa de Detecção Falsa de $2,35$, mesmo que o "0"possua a Taxa de Falso Negativo mais baixa, a Taxa de Detecção Falsa do número "1"é mais baixa. Em outra palavras, quando o modelo for fazer uma classificação e classificar como "1", é mais

confiável do que quando ele falar que é "0". No entanto, o número "1"ele acaba por classificar mais como outros números do que o "0".

Tabela 5: Classificação do Random Forest nas bases de teste

Classe \ Previsão	0	1	2	3	4	5	6	7	8	9
0	4078	0	1	1	4	3	24	0	19	2
1	0	4614	22	13	8	3	7	7	7	3
2	16	10	4032	22	19	1	12	34	28	3
3	6	5	63	4128	0	44	6	30	45	24
4	3	7	8	1	3941	1	16	7	9	79
5	18	4	3	57	8	3629	37	1	17	21
6	21	5	2	0	5	24	4067	0	13	0
7	4	21	45	2	21	0	0	4233	8	67
8	9	22	18	53	20	32	14	5	3852	38
9	21	11	6	64	50	6	2	38	30	3960

Tabela 6: Medidas de qualidade do Random Forest

Classe	TxFN (%)	TxDF (%)
0	1, 31	2, 35
1	1, 49	1, 81
2	3, 47	4, 00
3	5, 13	4, 91
4	3, 22	3, 31
5	4, 37	3, 05
6	1, 69	2, 82
7	3, 82	2, 80
8	5, 19	4, 37
9	5, 44	5, 65

4.2.1 *Random Forest com PCA*

Para a utilização do *PCA* seguimos o mesmo procedimento que na seção anterior, mais a informação que explicamos na Seção 4.1.

A Tabela 7, segue o mesmo modelo da tabela da seção anterior, mas tem uma informação a mais, o tempo de execução do processo de *PCA*. O tempo de execução do *PCA*, tem o valor que cada iteração demorou para fazer o processo, separadas nas mesmas descrições que o tempo de execução do modelo. Com a aplicação do *PCA*, nós diminuímos o tempo de processamento de uma maneira bem substancial. Nosso tempo máximo de execução passou a ser um pouco mais de 8 minutos, mesmo se adicionarmos o tempo que o *PCA* demora para rodar, chega a ser quase 4 vezes mais rápido. Olhando

para a taxa de erro total na Tabela 8, podemos perceber que o erro aumentou. Como o Random Forest já tinha uma taxa de erro baixa e nós diminuímos o número de informações, não foi surpresa o aumento do erro. Mas dependendo do objetivo da pessoa, o ganho no tempo de processamento pode superar o aumento de 2 pontos percentuais na taxa de erro total.

Tabela 7: Resumo da performance do *Random Forest* com *PCA* nas bases de Treino

	Min	1º Qu.	Mediana	Média	3º Qu.	Max
Tempo de execução (min)	7,70	7,93	7,97	7,95	8,00	8,05
Tempo de execução PCA (min)	2,21	2,22	2,22	2,22	2,23	2,25
Erro (%)	5,45	5,50	5,55	5,55	5,58	5,68

Seguindo para a avaliação da qualidade na base de teste, não existe diferença entre essa tabela e a vista na seção anterior, então a forma de ler as informações é a mesma.

Com a inserção do *PCA* a taxa de erro total subiu para 5,21%, o erro na base geral utilizando a Validação Cruzada, foi menor do que o erro médio nas bases de treinos. Além disso, foi uma boa taxa de erro para um processo mais de 4 vezes mais rápido.

Olhando para os erros na classe na Tabela 8, temos que o número "8"é agora o número com a maior Taxa de Falso Negativo, chegando a 8,91%, e também é o número com a maior Taxa de Detecção Falsa, com um percentual de 6,75%. O número "1"que já tinha a menor Taxa de Detecção Falsa, passou a ter também a menor Taxa Falsa Negativa. Já o número "3"teve um aumento considerável na Taxa de Detecção Falsa, chegando a 8,5%, sendo assim, o número "3"é o que possuí a menor confiança, caso não tivéssemos os valores da classe.

Tabela 8: Classificação do *Random Forest* com *PCA* nas bases de teste

Classe \ Previsão	0	1	2	3	4	5	6	7	8	9
0	4040	0	9	12	5	9	41	2	11	3
1	0	4582	35	19	5	9	7	9	16	2
2	23	9	3944	38	27	4	8	37	80	7
3	8	1	75	4056	2	61	15	33	66	34
4	4	17	20	3	3892	1	26	8	16	85
5	22	2	10	114	24	3515	52	6	28	22
6	29	4	10	1	10	43	4034	0	6	0
7	9	24	43	5	35	3	2	4186	13	81
8	15	26	36	115	24	74	20	12	3701	40
9	24	6	14	70	87	13	3	78	32	3861

Tabela 9: Medidade de qualidade do *Random Forest* com *PCA*

Classe	TxFN (%)	TxDF (%)
0	2, 23	3, 21
1	2, 18	1, 91
2	5, 58	6, 01
3	6, 78	8, 50
4	4, 42	5, 33
5	7, 38	5, 81
6	2, 49	4, 13
7	4, 89	4, 23
8	8, 91	6, 75
9	7, 81	6, 63

4.3 *K-Means*

Como o *K-Means* é um método não supervisionado, utilizamos o ponto de partida como a classe que era esperada para aquele *cluster*, fazendo assim com que não fosse preciso uma pessoa para classificar cada um dos *cluster*. Para escolher o ponto de partida de cada um dos *clusters*, foi pego o primeiro número, do nosso banco de dados, de cada uma das nossas classes como *cluster* inicial. Fazendo com que ele se adequasse com a base que era utilizada em cada uma das interações da Validação Cruzada.

Vale ressaltar que para construirmos as medidas de qualidade nós seguimos a Validação Cruzada, que falamos na Seção 3.6. Como adiantamos na Seção 3.7 e vimos na Seção do *Random Forest* a Tabela 10 se comporta da mesma maneira.

Esse modelo tem um tempo de execução impressionante, como podemos ver na Tabela 10. Sendo o tempo máximo de um pouco mais de 3 minutos e ainda parece ser um ponto discrepante em relação as outras interações. A média do tempo de execução não chega nem a dois minutos de execução. Infelizmente em relação ao erro total, a média da taxa de erro total é de $46, 28\%$.

Tabela 10: Resumo da performance do *K-Means* nas bases de Treino

	Min	1º Qu.	Mediana	Média	3º Qu.	Max
Tempo de execução (min)	1, 18	1, 33	1, 37	1, 72	1, 77	3, 19
Erro (%)	42, 52	43, 10	43, 26	46, 28	44, 25	66, 98

A Tabela 11 segue o mesmo conceito apresentado nas duas seções anteriores. Para esse método, temos que a taxa de erro total foi de $46, 22\%$, um percentual de erro bem maior que o apresentado pelo *Random Forest*. Na Tabela 12, podemos perceber que o número

com a menor Taxa de Falsa Negativo é o número "1", chegando até $6,9\%$, sendo assim muito menor do que as outras classes do mesmo método. No entanto a taxa de Detecção Falsa é de $35,24\%$. Em outras palavras, quando ele prevê que é o número "1", ocorre que o método arrisca muito o número "1"para classificar outras classes, não sendo assim um resultado confiável para a previsão do número "1". Já o o número "0"é o número com a Menor Taxa de Detecção Falsa e uma Taxa de Falsa Negativo de $39,11$. Podemos ter uma confiança maior quando ele está prevendo o zero, do que quando prevê outro número, mas não necessariamente ele classfica bem o "0". Em outras palavras, quando o modelo falar que é "0", provavelmente é "0", mas não necessariamente ele previu todos os "0"corretamente. Infelizmente, esse modelo chega a ter números com mais de 60% de erro, seja a Taxa de Falso Negativo ou a Taxa de Detecção Falsa.

Tabela 11: Classificação do *K-Means* nas bases de teste

Classe \ Previsão	0	1	2	3	4	5	6	7	8	9
0	2516	8	19	329	26	1082	95	6	47	4
1	0	4361	23	6	6	2	11	257	15	3
2	17	456	2654	186	122	117	390	51	138	46
3	15	243	397	2533	67	316	39	119	601	21
4	3	161	5	0	1655	33	72	984	7	1152
5	38	479	127	1234	151	810	55	185	562	154
6	73	296	41	19	118	725	2815	20	29	1
7	6	263	27	4	550	9	5	1581	17	1939
8	23	354	94	649	78	102	40	113	2483	127
9	26	113	5	58	1216	12	9	1534	34	1181

Tabela 12: Medidadas de qualidade do *K-Means* nas bases de teste

Classe	TxFN (%)	TxDF (%)
0	$39,11$	$7,40$
1	$6,90$	$35,24$
2	$36,46$	$21,76$
3	$41,78$	$49,52$
4	$59,36$	$58,51$
5	$67,48$	$90,15$
6	$31,96$	$20,28$
7	$63,37$	$68,37$
8	$38,89$	$36,87$
9	$55,94$	$58,10$

4.3.1 *K-Means com PCA*

Para a utilização do *PCA* seguimos o mesmo procedimento que na seção anterior, mais a informação que explicamos na Seção 4.1.

Com o *PCA*, estávamos esperando uma melhora grande na taxa de erro, removendo boa parte da variável poderia diminuir o ruído para essa técnica. Mas o que o ocorre é a média da taxa de erro total continuar alta, chegando a $46,75\%$, como podemos ver na Tabela 13. As outras variáveis dessa tabela, seguem a mesma dinâmica da tabela vista na seção do *Random Forest com PCA*. O tempo de execução do modelo caiu, chegando a média de $0,28$ minutos. Mas se formos adicionar o tempo de execução do *PCA* na conta, acaba por ficar mais demorado que o modelo anterior.

Tabela 13: Resumo da performance do *K-Means* com *PCA* nas bases de Treino

	Min	1º Qu.	Mediana	Média	3º Qu.	Max
Tempo de execução (min)	$0,23$	$0,24$	$0,26$	$0,28$	$0,27$	$0,49$
Tempo de execução PCA (min)	$2,18$	$2,18$	$2,22$	$2,25$	$2,31$	$2,35$
Erro (%)	$42,53$	$43,11$	$43,26$	$46,75$	$44,25$	$71,61$

Seguindo para a avaliação da qualidade na base de teste, não existe diferença entre essa tabela e a vista na seção anterior, então a forma de ler as informações é a mesma.

Com a inserção do *PCA* a taxa de erro total subiu para $46,72\%$. Na Tabela 15 podemos perceber que o número "1"continua sendo o número com a menor Taxa Falsa Negativa, mas chegou a quase dobrar esse número, chegando a $12,40\%$, e uma Taxa de Detecção Falsa de $35,46\%$. O número "0"permaneceu praticamente com a mesma Taxa de Detecção Falsa, $7,32\%$, sendo a menor de todos os números. Mas se formos comparar com os resultados do *Random Forest*, até mesmo o menor erro é mais alto que o erro mais alto de lá, como podemos conferir na Tabela 5.

4.4 *k-NN*

Vale ressaltar que para construirmos as medidas de qualidade nós seguimos a Validação Cruzada, que falamos na Seção 3.6. Como adiantamos na Seção 3.7 e vimos na Seção do *Random Forest*, a Tabela 18, que apresenta as medidas de qualidade, se comporta da mesma maneira das seções anteriores.

Mesmo com a simplicidade do k-NN, nós temos resultados impressionantes. Uma taxa

Tabela 14: Classificação do *K-Means* com *PCA* nas bases de teste

Classe \ Previsão	0	1	2	3	4	5	6	7	8	9
0	2581	6	22	275	25	1084	93	7	36	3
1	0	4103	23	6	6	1	7	520	15	3
2	19	442	2645	185	124	118	386	68	144	46
3	15	222	391	2560	76	312	33	132	590	20
4	3	161	5	4	1720	34	72	891	7	1175
5	37	452	126	1313	157	806	53	193	517	141
6	73	266	41	61	115	698	2807	51	24	1
7	7	251	27	4	634	9	6	1429	19	2015
8	23	345	84	658	82	102	35	118	2489	127
9	27	109	5	57	1315	13	10	1378	38	1236

Tabela 15: Medidadas de qualidade *K-Means* com *PCA*

Classe	TxFN (%)	TxDF (%)
0	37, 54	7, 32
1	12, 40	35, 46
2	36, 68	21, 49
3	41, 16	50, 03
4	57, 76	59, 57
5	78, 76	90, 18
6	32, 15	19, 85
7	67, 53	71, 21
8	38, 74	35, 83
9	70, 49	57, 73

de erro média, na base de treino, de 8, 39%, como podemos ver na Tabela 16. A taxa de erro total máximo foi de 8, 47% e o mínimo de 8, 32%, ou seja, o erro foi muito próximo em todas as interações. No entanto, o tempo de execução dessa técnica é alto, chegando em média a 42, 90 minutos por iteração, com um máximo de 44, 87 minutos. Sendo assim mais demorado do que o *Random Forest*.

Tabela 16: Resumo da performance do k-NN nas bases de Treino

	Min	1º Qu.	Mediana	Média	3º Qu.	Max
Tempo de execução (min)	41, 50	41, 93	42, 57	42, 75	42, 90	44, 87
Erro (%)	8, 32	8, 37	8, 39	8, 39	8, 42	8, 47

A Tabela 17 volta a ser como era na Seção do *Random Forest*. As linhas são as classe e as colunas são as classificações do modelo, por exemplo, se formos olhar para a segunda coluna e para a segunda linha, temos que 4042 "0"foram realmente classificados como "0". Na última coluna temos Taxa de Falso Negativo (TxFN) e na última linha a Taxa

de Detecção Falsa (TxDF), como explicados na Seção 3.7.

Temos que o erro percentual total foi de $8,54\%$, um pouco maior que a média da base de treino, mas ainda uma boa taxa. Na Tabela 18, podemos ver que temos problemas concentrados em alguns números como o "8"e o "2", suas Taxa de Falso Negativo são respectivamente $18,39\%$ e $16,02\%$. O número "1"possuí a menor Taxa de Falso Negativo, com uma taxa de somente $0,51\%$. Em contra partida, a classe "1"tem a maior Taxa de Detecção Falsa, chegando a $19,81\%$. O que pode explicar esse comportamento é o tamanho do nosso k, lembrando que estamos pegando os k vizinhos mais próximos para classificar aquele número. O número "1"deve estar bem concentrado entre si, mas o número "2"e "8"devem estar mais espaçados entre si, ou seja, acabam por serem mais influenciados por outros números, por exemplo, o número "1"deve ser um grande influenciador. No caso, outro grande influenciador seria o número "9", com uma Taxa de Detecção Falsa de $14,83\%$. Provavelmente modificando o k, conseguiríamos uma Taxa de erro ainda menor, mas a proposta do trabalho foi seguir o parâmetros indicados por livros ou artigos.

Tabela 17: Classificação do *k-NN* nas bases de teste

Classe \ Previsão	0	1	2	3	4	5	6	7	8	9
0	4042	4	3	1	2	19	48	3	5	5
1	0	4660	6	4	2	0	3	6	1	2
2	73	291	3508	37	26	10	25	144	41	22
3	15	107	27	3962	4	60	18	53	46	59
4	3	129	0	0	3630	2	29	14	2	263
5	28	90	0	105	18	3381	80	8	9	76
6	54	56	0	0	12	24	3988	2	1	0
7	5	202	10	0	16	1	1	4057	0	109
8	31	214	7	166	29	105	33	24	3316	138
9	25	58	4	54	40	5	4	119	8	3871

4.4.1 *k-NN com PCA*

Para a utilização do *PCA* seguimos o mesmo procedimento que na seção anterior, mais a informação que explicamos na Seção 4.1.

A Tabela 19, não tem nenhuma diferença das seções, com o PCA, anteriores. Diferente do *Random Forest*, tivemos uma redução na taxa de erro total, chegando a uma média de $7,88\%$. A taxa de erro total máximo foi de $7,95$ minutos e o mínimo de $7,83$, ou seja, está muito bem distribuído entre todas as interações. O tempo de execução também melhorou

Tabela 18: Medidas de qualidade do *k-NN*

Classe	TxNF (%)	TxDF (%)
0	2, 18	5, 47
1	0, 51	19, 81
2	16, 02	1, 60
3	8, 94	8, 48
4	10, 85	3, 94
5	10, 91	6, 27
6	3, 60	5, 70
7	7, 82	8, 42
8	18, 39	3, 30
9	7, 57	14, 83

consideravelmente, caindo para uma média de 7, 00 minutos, mesmo adicionando o tempo de execução do *PCA* ainda é um tempo 4 vezes mais rápido do que o original.

Tabela 19: Resumo da performance do *k-NN* com *PCA* nas bases de Treino

	Min	1º Qu.	Mediana	Média	3º Qu.	Max
Tempo de execução (min)	6, 85	6, 90	6, 92	7, 00	6, 99	7, 66
Tempo de execução PCA (min)	2, 15	2, 16	2, 17	2, 17	2, 17	2, 21
Erro (%)	7, 83	7, 84	7, 90	7, 88	7, 90	7, 95

Seguindo para a avaliação da qualidade na base de teste, não existe diferença entre essa tabela e a vista na seção anterior, então a forma de ler as informações é a mesma.

Como podemos perceber na Tabela 21, tivemos uma melhora na taxa de erro total da base de teste, chegando a 8, 02%. O número "1"chegou a um Taxa de Falso Negativo de 0, 60%, somente um pouco maior do que a versão sem o *PCA*. Enquanto o número "2"e "8"tiveram uma melhora mais considerável em sua Taxa de Falso Negativo, chegando a respectivamente 14, 36 e 17, 03. A Taxa de Detecção Falsa continua com os mesmo prolemas da Seção anterior. Um fato interessante é que o *Random Forest* comete erros mais parecidos com erros humanos, como confundir "4"com o "9"e o "9"com o "4", como podemos ver na Tabela 6. Já o *k-NN*, o maior problema dele é com o número "1", ele acaba por confundir o "8"com o número "1", um erro que dificilmente um humano cometeria.

Tabela 20: Classificação do *k-NN* com *PCA* nas bases de teste

Classe \ Previsão	0	1	2	3	4	5	6	7	8	9
0	4042	3	3	1	1	18	52	2	5	5
1	0	4656	8	5	2	0	4	6	1	2
2	67	242	3577	33	22	10	27	138	41	20
3	14	88	31	3977	3	58	21	58	49	52
4	3	123	1	0	3645	2	30	15	2	251
5	27	82	0	104	13	3399	79	8	9	74
6	46	45	0	0	8	24	4012	1	1	0
7	4	189	11	0	16	1	1	4077	0	102
8	30	190	7	157	25	99	34	24	3371	126
9	25	55	4	55	40	5	4	118	8	3874

Tabela 21: Medidas de Qualidade do *k-NN* com *PCA*

Classe	TxFN (%)	TxDF (%)
0	2, 18	5, 07
1	0, 60	17, 93
2	14, 36	1, 78
3	8, 60	8, 19
4	10, 49	3, 44
5	10, 43	6, 00
6	3, 02	5, 91
7	7, 36	8, 32
8	17, 03	3, 33
9	7, 50	14, 03

4.5 Comparando Resultados do *PCA*

Neste trabalho, a técnica de *PCA* foi utilizada para fazer o redimensionamento da base, conseguindo cumprir com esse objetivo perfeitamente. Devido a esse fato e não termos tido uma perda mais significativa de qualidade, decidimos fazer uma comparação mais direta com todos os modelos rodados com *PCA*. Criamos a Tabela 22, que contém as informações de Taxa de Falso negativo e as informações de Taxa de Detecção Falsa de cada uma das técnicas com o *PCA*.

Se formos comparar as informações por classe, o *Random Forest* sempre esterá com uma Taxa de Detecção Falsa menor do que as outras duas técnicas e a Taxa de Falso Negativo, na maioria das vezes, também sera menor. Por exemplo, se formos olhar para a classe "2"podemos ver que é a única classe que o *k-NN* tem uma taxa melhor do que o *Random Forest*.

Tabela 22: Comparação das Medidas de Qualidade dos resultados com *PCA*

	Random Forest		*K-means*		*k-NN*	
Classe	TxFN	TxDF	TxFN	TxDF	TxFN	TxDF
0	2,23	3,21	37,54	7,32	2,18	5,07
1	2,18	1,91	12,40	35,46	0,60	17,93
2	5,58	6,01	12,40	35,46	14,36	1,78
3	6,78	8,50	41,16	50,03	8,60	8,19
4	4,42	5,33	57,76	59,57	10,49	3,44
5	7,38	5,81	78,76	90,18	10,43	6,00
6	2,49	4,13	32,15	19,85	3,02	5,91
7	4,89	4,23	67,53	71,21	7,36	8,32
8	8,91	6,75	38,74	35,83	17,03	3,33
9	7,81	6,63	70,49	57,73	7,50	14,03

Mesmo que olhando os resultados por classe o *Random Forest* seja melhor em todas, vemos que se formos olhar de uma forma geral, o *k-NN* tem resultados melhores do que alguns dos resultados do *Random Forest* e se pudêssemos usar isso para criar um melhor modelo?

Com essa pergunta, surgiu a ideia de fazer um novo modelo, este modelo será uma mescla desses métodos, utilizando o que tem a melhor performance no resultado na classe que foi prevista pelo modelo. Para escolhermos quem tem a melhor performance na classe, será utilizada a Taxa de Detecção Falsa, ela nos traz o quanto o modelo é confiável naquela classificação. Ou seja, se eu tiver três previsões diferentes, cada uma feita por um método diferente, seria aconselhável escolher a que tiver com a menor Taxa de Detecção Falsa. Por exemplo, *Random Forest* classifica a primeira unidade experimental como "9", o *k-NN* classifica como "1"e o *K-Means* classifica como "9"também, as Taxas de Detecção Falsas são $5,65\%$, $1,91\%$ e $57,73$ respectivamente. Isso significa que $5,65\%$ das classificações de "9"do *Random Forest* estão erradas, $57,73\%$ das classificações de "9"do *K-means* estão erradas, enquanto somente $1,91\%$ dos números "1"classificados pelo *k-NN* estão errados. Nós escolheríamos o número "1"como classificação final para essa unidade experimenta.

4.5.1 Resultado para o modelo Mescla

Como dito anteriormente, nós iremos fazer as classificações e olharemos o resultado de cada uma das unidades experimentais e escolheremos como resultado final o que tiver a melhor Taxa de Detecção Falsa. Vale observar que o *K-Means* não foi utilizado para fazermos a mescla, foi observado que até o menor valor da Taxa de Detecção Falsa do *K-Means* é maior do que o maior valor do *Random Forest*, sendo assim, nunca seria escolhida

alguma classificação do *K-Means*.

A Tabela 23, não tem nenhuma diferença das seções anteriores. Em média, cada iteração demorou 16, 86 minutos para rodar, menos da metade do tempo que o *Random Forest* ou o *k-NN* demoram para rodar. O tempo de execução do *PCA* subiu um pouco, comparado as outras vezes, essa diferença pode ter acontecido por algum fator que diminui a potencia do computador. Tivemos um erro médio total de 1, 84%, uma ótima taxa e com uma variação muito pequeno, o valor mínimo foi 1, 80% e o máximo foi 1, 87%.

Tabela 23: Resumo da performance da mescla de modelos nas bases de Treino

	Min	1º Qu.	Mediana	Média	3º Qu.	Max
Tempo de execução (min)	16, 28	16, 39	16, 44	16, 86	17, 13	18, 68
Tempo de execução PCA (min)	2, 48	2, 50	2, 51	2, 58	2, 55	3, 12
Erro (%)	1, 80	1, 82	1, 84	1, 84	1, 86	1, 87

A Tabela 24 guarda as informação de classificação de cada uma das classes. Obtemos uma taxa de erro total na base de teste de 5, 35%, sendo um número bem maior do que o encontrado na base de treino. Na Tabela 25 podemos ver que o número "0"chegou a um Taxa de Falso Negativo de 1, 60%, enquanto sua Taxa de Detecção Falsa chegou a 5, 57%. Se formos comparar com o *k-NN* com *PCA*, temos que os valores ficaram com um menor variação entre si, mesmo que alguns tenham subido, outros valores diminuíram muito, como por exemplo a classe "8"que teve uma redução na Taxa de Falso Negativo de 17, 03% para 9, 97%. Enquanto sua Taxa de Detecção Falsa subiu bem menos, indo de 3, 33% para 5, 84%, como podemos ver na Tabela 21. Como o *Random Forest* tinha, de uma maneira geral, umas taxas de erro melhores que o *k-NN* s resultados estão bem mais próximos dos valores dele, tendo melhoras em alguns números, mas piorando bastante em outros, como por exemplo a classe "9"que a Taxa de Detecção Falsa era de 6, 63% e subiu para 8, 29%.

Tabela 24: Classificação do modelo Mescla nas bases de teste

Classe \ Previsão	0	1	2	3	4	5	6	7	8	9
0	4066	0	3	2	3	7	42	2	6	1
1	0	4585	27	16	6	9	12	10	17	2
2	69	10	3906	17	27	8	28	42	62	8
3	17	2	66	3986	3	89	27	55	76	30
4	5	18	7	1	3915	3	22	9	12	80
5	34	2	6	63	31	3569	54	5	15	16
6	40	4	2	0	12	39	4037	0	3	0
7	9	26	43	3	38	0	3	4188	11	80
8	35	24	17	91	34	121	34	11	3658	38
9	31	5	12	65	108	15	3	83	25	3841

Tabela 25: Medidas de Qualidade do Modelo Mescla, em comparação com *Random Forest* e *k-NN* com *PCA*

	Random Forest		**Mescla**		*k-NN*	
Classe	TxFN	TxDF	TxFN	TxDF	TxFN	TxDF
0	$2,23$	$3,21$	$1,6$	$5,57$	$2,18$	$5,07$
1	$2,18$	$1,91$	$2,11$	$1,95$	$0,60$	$17,93$
2	$5,58$	$6,01$	$6,49$	$4,48$	$14,36$	$1,78$
3	$6,78$	$8,50$	$8,39$	$6,08$	$8,60$	$8,19$
4	$4,42$	$5,33$	$3,86$	$6,27$	$10,49$	$3,44$
5	$7,38$	$5,81$	$5,96$	$7,54$	$10,43$	$6,00$
6	$2,49$	$4,13$	$2,42$	$5,28$	$3,02$	$5,91$
7	$4,89$	$4,23$	$4,84$	$4,93$	$7,36$	$8,32$
8	$8,91$	$6,75$	$9,97$	$5,84$	$17,03$	$3,33$
9	$7,81$	$6,63$	$8,29$	$6,23$	$7,50$	$14,03$

5 Conclusões

O objetivo desse trabalho foi comparar técnicas estatísticas quando utilizadas para reconhecer números manuscritos. As técnicas que foram utilizadas foram *Random Forest*, *K-means* e *k-NN*, para cada uma dessas técnicas o modelo foi feito com e sem *PCA*. Os melhores resultados, pensando em taxa de erro, foi alcançado pelo *Random Forest*, chegando a ter $3,49\%$ de taxa de erro total, sendo $5,44\%$ maior Taxa de Falso Negativo e $5,65\%$ a maior Taxa de Detecção Falsa.

Todavia, esse método possui um tempo longo de processamento, se formos olhar o resultado por esse caminho, temos o método *K-Means* como o mais rápido, mas a taxa de erro do mesmo chega a ser $46,22\%$ não compensando o uso do método. Como opção para escolhermos uma técnica com uma baixa taxa de erro e um bom desempenho computacional temos o *Random Forest* ou o *k-NN*, ambos com *PCA*. O *Random Forest* com *PCA* tem uma taxa de erro de $5,21\%$ e demora em média $7,95$ minutos, por iteração, enquanto o *k-NN* tem uma taxa de erro total de $8,02\%$ e demora em média 7 minutos por iteração. Podemos dizer, que no *k-NN* ainda temos uma grande oportunidade, nós escolhemos k seguindo a sugestão do livro de Duda et al. [6], ou seja, ainda podemos ter grandes oportunidades de escolher o melhor k para esse método, o mesmo pode valer para o *Random Forest*.

Como dito anteriormente, nós não obtivemos bons resultados *K-Means*, não acreditamos que uma outro escolha de parâmetros poderia melhorar em muito o resultado, sendo assim, acreditamos que esse não é um modelo adequado para fazer esse tipo de análise, não encontramos também nenhum trabalho relacionando *K-Means* a tarefa de reconhecimento de números manuscritos.

Nosso modelo Mescla teve uma taxa de erro total de $5,35\%$, sendo um pouco maior do que o do *Random Forest* sem *PCA* e menor do que o *k-NN* com *PCA*. Um fator importante desse método é a diferença entre a taxa de erro na base de treino e na base teste, a taxa de erro total chegou a uma média de $1,84\%$, sendo assim a melhor taxa de erro. Com esse

fato podemos constatar que nosso modelo Mescla sofre *overfitting*, em outras palavras, nosso modelo está ajustado em demasiado para o nosso banco de dados, não conseguindo manter a qualidade em dados que não foram usados para o treino do modelo.

Referências

[1] LECUN, Y.; CORTES, C. The MNIST database of handwritten digits. Disponível em: <http://yann.lecun.com/exdb/mnist/>.

[2] JOHNSON, R. A.; WICHERN, D. W. *Applied Multivariate Statistical Analysis*. 6. ed. [S.l.]: Pearson Education, 2007.

[3] WAGSTAFF, K. et al. Constrained k-means clustering with background knowledge. *Proceedings of the Eighteenth International Conference on Machine Learning*, p. 577–584, 2001.

[4] QUINLAN, J. Induction of decision trees. *Machine Learning*, n. 1, p. 81–106, 1986.

[5] BREIMAN, L. Random forests. *Machine Learning*, n. 45, p. 5–32, 2001.

[6] DUDA, R. O.; HART, P. E.; STORK, D. G. *Patter Classification*. 2. ed. [S.l.]: John Wiley & Sons, Inc, 2000.