FACULDADE DE TECNOLOGIA DE AMERICANA
Curso de Tecnologia em Segurança da Informação

Tulio Cruvinel Gomes

MACHINE LEARNING APLICADO A ESTEGANÁLISE

Americana, SP
2020

FACULDADE DE TECNOLOGIA DE AMERICANA
Curso de Tecnologia em Segurança da Informação

Tulio Cruvinel Gomes

MACHINE LEARNING APLICADO A ESTEGANÁLISE

Projeto de Iniciação Científica desenvolvido na FATEC Americana no Curso de Tecnologia em Segurança da Informação, sob orientação do(a) Prof. Me. Rossano Pablo Pinto.

Área de concentração: Segurança da Informação

Americana, SP
2020

RESUMO

Este projeto visa explorar a construção de um sistema de detecção de imagens esteganografadas a partir da habilidade de aprendizagem utilizando técnicas da área de Machine Learning (aprendizado de máquina). A construção do sistema engloba o desenvolvimento de software à instalação e configuração de plataformas existentes. O projeto visa utilizar frameworks open source para o projeto, como o Scikit-Learn.

Palavras-chave: Machine Learning, Esteganálise, Esteganografia.

LISTA DE QUADROS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A área de Inteligência Artificial (IA) foi tida como promessa para a construção de sistemas inteligentes em diversas ocasiões (NORVIG; INTELLIGENCE, 2009). Em tais ocasiões, as promessas foram maiores do que a realidade e a IA acabou não sendo utilizada de forma ampla pela indústria (ver Inverno da IA - (NORVIG; INTELLIGENCE, 2009). Com o surgimento da Internet, e com ela uma oferta grande de dados gerados pela sociedade, um ramo bastante importante da IA, conhecido como Machine Learning (ML) (HARRINGTON, 2012; NORVIG; INTELLIGENCE, 2009), passou a apresentar resultados reais para diversos ramos da indústria, principalmente no contexto da indústria 4.0, que faz uso de sistemas cyber-fisicos IoT, computação cognitiva e computação em nuvem. Exemplos de aplicações que utilizam ML são: veıculos autônomos, assistentes pessoais comandados por voz (Siri da Apple; Alexa da Amazon; e Google Assistant da Google), sistemas antifraude, reconhecimento de objetos em fotos e filmes, classificação de objetos, mercado financeiro (tendências da bolsa de valores, mercado consumidor),robôs domésticos e industriais, robôs cozinheiros, redes sociais como twitter e facebook - dentre vários outros.

1.1 JUSTIFICATIVA

Tarefas antes realizadas apenas por humanos são realizadas hoje por máquinas guiadas por técnicas de aprendizagem de máquina, tais como carros autônomos e reconhecimento de padrões diversos. O estudo e a popularização do conceito e a aplicabilidade de ML em diversos problemas mostra-se bastante relevante em um tempo em que se fala da automatização da tomada de decisão, indústria 4.0 e descentralização.

1.2 OBJETIVOS GERAIS

Este projeto tem como objetivo geral explorar - no sentido de testar, usar ou programar - algoritmos e modelos de ML.

1.3 OBJETIVOS ESPECÍFICOS

Explorar (e/ou aperfeiçoar) algoritmos de ML supervisionados e não-supervisionados aplicados a problemas reais em ambientes reais, como uma análise comparativa entre os arquivos originais de imagem e respectivos arquivos com mensagens escondidas.

- Entender a estrutura dos algoritmos de classificação e suas aplicações
- Treinar os algoritmos de classificação para reconhecerem o padrão de uma imagem esteganografada.

- Desenvolver um software que percorra o diretório de arquivos de um sistema operacional (Windows/Linux) em busca de arquivos jpeg e png.
- O software irá copia-los para uma pasta e separar pelas extensões.
- O software irá análisar cada imagem e gerar uma porcentagem de precisão da imagem esteganografada para cada algoritmo de classificação já treinados.
- Realizar uma relatório sobre o método dos algoritmos e classificar a imagem em uma porcentagem baseada na analise de todos os algoritmos envolvidos.
- O software irá copiar os arquivos de imagem analisados com índices superiores a 75 porcento de esteganografia para outra pasta, a fim de facilitar para uma próxima etapa de análise de quarentena.

2 REVISÃO DA LITERATURA

Neste capítulo serão abordados os principais trabalhos que constituem o desenvolvido da pesquisa e da exploração dos algoritmos mais populares para aprendizado de máquina, que posteriormente serão utilizados para o desenvolvimento da aplicação final.

2.1 MACHINE LEARNING

Machine Learning é um campo de estudo da Inteligencia Artificial onde a proposta é desenvolver um software que tem a capacidade de aprender. Esse aprendizado é feito a partir de um treinamento, onde são inseridos os dados que através de um processo os transforma em conhecimento para executar alguma tarefa ou ser aplicado a resolução de problema através do reconhecimento de padrões (SHARIFZADEH et al., 2018; MOHAMMED; KHAN; BASHIE, 2016). Em outros estudos, Nilsson (1998) complementa que podemos considerar o aprendizado como a partir do momento que sua estrutura é alterada, baseada nos dados inseridos e isso impacta nos resultados obtidos da aplicação. ML é sobre resolver problemas e garantir os resultados através de modelos matemáticos, fluxo de dados e modelos de problemas determinísticos (SMOLA; VISHWANATHAN, 2008).

Segundo Harrington (2012) ML é um encontro entre as áreas de Ciências da Computação, Engenharia e Estatística. Tem sido aplicada em diversas áreas do conhecimento humano e devido a sua eficácia, tem se popularizado pela mídia a partir de 2010 e aplicada por várias empresas como Amazon, Netflix, Google e Facebook. Suas aplicações finais mais comuns são motores de busca, reconhecimento facial, reconhecimento de voz, detecção de fraudes, sistemas de recomendações, reconhecimento de manuscrito, processamento de linguagem natural e dentre tantas outras aplicações. Qualquer situação onde se tenha um problema e que ele possua um padrão, então ele pode ser classificado e solucionado por ML.

No aprendizado de máquina é comum escrevermos um modelo a partir do problema, inserir os dados de entrada e os resultados desejados, porém este processo de aprendizagem pode se tornar difícil para exemplificar pela linguagem e através dos modelos matemáticos e probabilísticos podemos reconhecer os padrões usados e isolar as variáveis para trazer um maior entendimento sobre o modelo do problema. É esse o grande trunfo que o ML traz e que o torna valioso, já que ele chega aos resultados dos problemas do modelo porém ele pode também reconhecer outros padrões e problemas que não foram reconhecidos devido a barreira da linguagem humana (MOHAMMED; KHAN; BASHIE, 2016; HARRINGTON, 2012).

Por isso possuímos alguns modelos de aprendizado, que facilitam a automatizam da construção de modelos analíticos, que são: Aprendizado supervisionado, aprendizado não supervisionado, aprendizado semi-supervisionado e aprendizado por reforço. Como nosso foco nesta pesquisa é explorar os modelos supervisionados e não supervisionados, os outros modelos

de classificação serão abordados superficialmente.

2.1.1 APRENDIZADO SUPERVISIONADO

No aprendizado supervisionado prevemos onde uma instância de dados pode ser encaixar dentro de um conjunto de dados já rotulados. Este tipo de modelo torna claro para o algoritmo o que queremos prever. Neste tipo de aprendizado, (SHALEV-SHWARTZ; BEN-DAVID, 2013) diz que semelhante ao processo de aprendizagem humano, onde acontece a interação de aluno e ambiente. Podemos considerar o ambiente como o professor que "Supervisiona"o aluno e fornece um conjunto de informações significativas (labels) e aplicando o processo de aprendizagem, podemos treinar o algoritmo aluno para "transformar a experiência em expertise". Frequentemente os supervisores são humanos, porém máquinas e também podem ser usadas para essa rotulagem, porém os dados rotulados manualmente tendem a ser um recurso mais preciso e confiável para o aprendizado de máquina, o que naturalmente torna os julgamentos humanos mais caros do que as máquinas Mohammed, Khan e Bashie (2016).

Em aprendizado supervisionado, temos duas categorias de algoritmos: Classificação e Regressão.Algoritmos de classificação são aplicados onde a variável de saída é uma categoria que já foi rotulada e consta como um atributo do dataset. Como exemplos de algoritmos de classificação, temos: kNN, Naive Bayes e Decision Trees. Os algoritmos de Regressão são quando precisamos fazer uma predição de um valor real ou numérico, como exemplos de algoritmos de regressão, temos: Linear Regression e SVM.

2.1.2 APRENDIZADO NÃO SUPERVISIONADO

No aprendizado não supervisionado é o treinamento da máquina usando informações que não são classificadas nem rotuladas, permitindo assim que o algoritmo processe as informações sem orientação ou supervisão. Neste momento, a tarefa da máquina é agrupar informações não classificadas de acordo com semelhanças, padrões e diferenças sem nenhum treinamento prévio de dados (MOHAMMED; KHAN; BASHIE, 2016).

Por não possuir supervisão, nenhum treinamento será dado à máquina, restringindo a ela encontrar os padrões ocultos nos dados não rotulados.

Em aprendizado não supervisionado, temos duas categorias de algoritmos: Clustering e Associação. Algoritmos de Clustering são aplicados quando precisamos encontrar grupos distintos dentro do nosso dataset. Algoritmos de Associação são quando precisamos encontrar uma relação ou dependência dentro do conjunto de dados do dataset. Regras de clustering e associação são essenciais para a mineração de dados para a extração de conhecimento.

2.2 ALGORITMOS

Nesta seção será abordado uma introdução, modelo matemático e a construção de cada algoritmo e os resultados obtidos utilizando datasets públicos durante esta etapa da pesquisa.

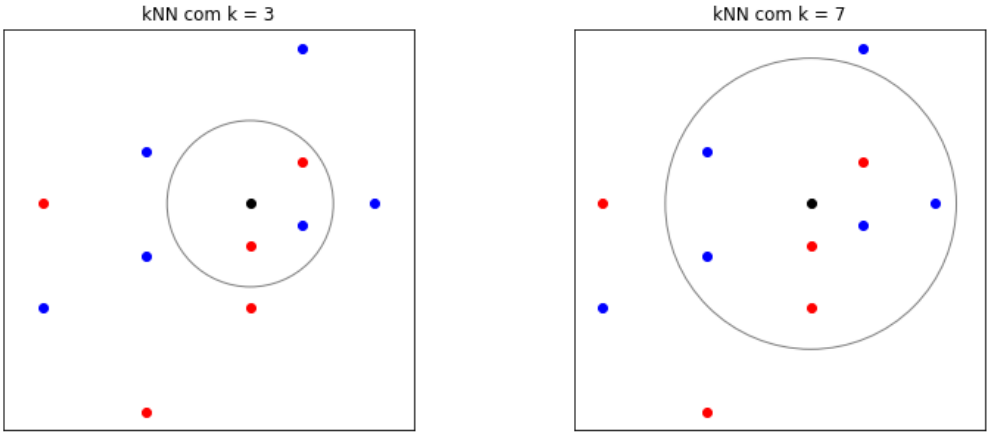
2.2.1 K - Nearest Neighbors

Considerado como o algoritmo mais simples de machine learning, *k*-Nearest Neighbors (kNN) é um método não paramétrico usado para classificação e regressão. Partindo de um padrão de entrada desconhecido e observando os padrões mais próximos ao redor dela, os vizinhos, podemos calcular a distância entre eles para atribuir uma classificação a partir de k vizinhos. O resultado depende se o *kNN* está aplicado à classificação ou regressão.

No *kNN* aplicado em classificação binária, a saída será uma associação à classe pertencente do voto majoritário do conjunto de vizinhos mais próximos. Se for definido $k = 1$, o vizinho mais próximo dará a classificação correspondente à entrada. Caso o *kNN* seja aplicado em regressão, a saída será a propriedade de valor do objeto, sendo ele composto pela média dos valores dos vizinhos.(MOHAMMED; KHAN; BASHIE, 2016; SMOLA; VISHWANATHAN, 2008).

A Figura 1 à esquerda ilustra o conceito onde o ponto preto é nosso dado de entrada. Se $k = 3$, este ponto pertencerá a classe vermelha. A Figura 1 à direita retrata o mesmo caso, porém com $k = 7$, atribuindo então a classe azul ao dado de entrada. Fica claro que o voto majoritário em ambas as figuras.

Figura 1 – K - Nearest Neighbors



Fonte: Autor

Por ser um algoritmo muito flexível, um dos seus problemas é que sua estimativa pode apresentar grande ruído e isso é reflexo de um dataset que possui também muito ruído, é necessário ter seus dados corretamente classificados para não gerar falsos-positivos na aplicação. Como exemplo, se um carro é erroneamente classificado como possuir a cor Verde e por consequência, todos os carros semelhantes a ele serão categorizados da mesma forma.

O *kNN* explora o *lazy learning*, sendo que o algoritmo não constrói um modelo até o instante em que uma predição é necessária. Isto traz o benefício de incluir apenas os dados

relevantes para a análise do novo padrão de entrada. Como desvantagem, todos os dados de treinamento precisam ser armazenados e consultados para se identificarem os vizinhos mais próximos, o que demanda um maior poder de processamento.

Podemos então observar a composição do *kNN* em três partes:

- Uma métrica de distância a ser calculada a fim de determinar os vizinhos mais próximos.
- Um valor para o parâmetro *k*, que determina a quantidade de vizinhos que serão comparados ao valor de entrada.
- Um dataset com o menor índice de ruído, reduzindo assim o impacto da estimativa.

2.2.1.1 Calculando a Distância

A matemática formaliza a ideia da distância como o comprimento entre dois pontos. A distância nos permitirá agrupar indivíduos semelhantes e separar aqueles que não se parecem. Existem diversas formas de calcular a distância entre dois pontos e o mais comum em *kNN* é a distância Euclidiana. Dado dois pontos como A e B, que representam nossos vetores de atributos $A = (x_1,x_2,\cdots,x_n)$ e $B = (y_1,y_2,\cdots,y_n)$ e onde *n* representa a dimensão do nosso vetor de atributos, a Equação (1) descreve a *Métrica Euclidiana*.

Distância Euclidiana:

$$d(A,B) = \sqrt{(A_1 - B_1)^2 + (A_2 - B_2)^2 + \cdots + (A_i - B_i)^2} = \sqrt{\sum_{r=1}^{n} (A_{ir} - B_{ir})^2} \quad (1)$$

Outros métodos comuns de calculo de distância são as variações de Manhattan e Minkowski, onde a distância Euclidiana é um caso especial da métrica de Minkowski, que é representado como $p = 2$ assim como Manhattan é outro caso, onde $p = 1$ (AHMEDMEDJAHED; Ait Saadi; BENYETTOU, 2013).

Distância de Minkowski, onde $p = 1,2,...,\infty$ representada pela Equação (2).

$$d(A,B) = \sqrt{\sum_{r=1}^{n} (A_{ir} - B_{ir})^p} \quad (2)$$

Distância de Manhattan representada pela Equação (3).

$$d(A,B) = \sqrt{\sum_{r=1}^{n} (A_{ir} - B_{ir})} \quad (3)$$

2.2.1.2 Definindo *k*

Não existe um único valor para a constante, é importante entender que ela pode variar de acordo com seu dataset. É comum se utilizar um número integro e de pequeno valor, números ímpares ou primos, para a denominação de *k* (FUKUNAGA; NARENDRA, 1975). Conforme a aplicação que você estiver explorando, você pode utilizar um algoritmo de otimização e encontrar um melhor valor para *k* ou testar um conjunto de valores de forma empírica e encontrar um valor mais apropriado.

2.2.1.3 Data Set

O *Iris Dataset* é um conjunto de dados multivariados, introduzido pelo estatístico e biólogo Fisher (1936) em seu artigo. O dataset foi criado a fim de quantificar a variação morfológica das flores da íris de três espécies relacionadas. O conjunto de dados consiste em 50 amostras de cada uma das três espécies de Iris (Iris setosa, Iris virgínica e Iris versicolor). Quatro características foram medidas em cada amostra: o comprimento e a largura das sépalas e pétalas, em centímetros. Com base na combinação dessas quatro características, Fisher (1936) desenvolveu um modelo discriminante linear para distinguir as espécies umas das outras.

Tabela 1 – Iris Dataset - Header

	sepal_length	sepal_width	petal_length	petal_width	species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	7.0	3.2	4.7	1.4	Iris-versicolor
4	6.4	3.2	4.5	1.5	Iris-versicolor
5	6.3	3.3	6.0	2.5	Iris-virginica
6	5.8	2.7	5.1	1.9	Iris-virginica

Fonte: Fisher (1936)

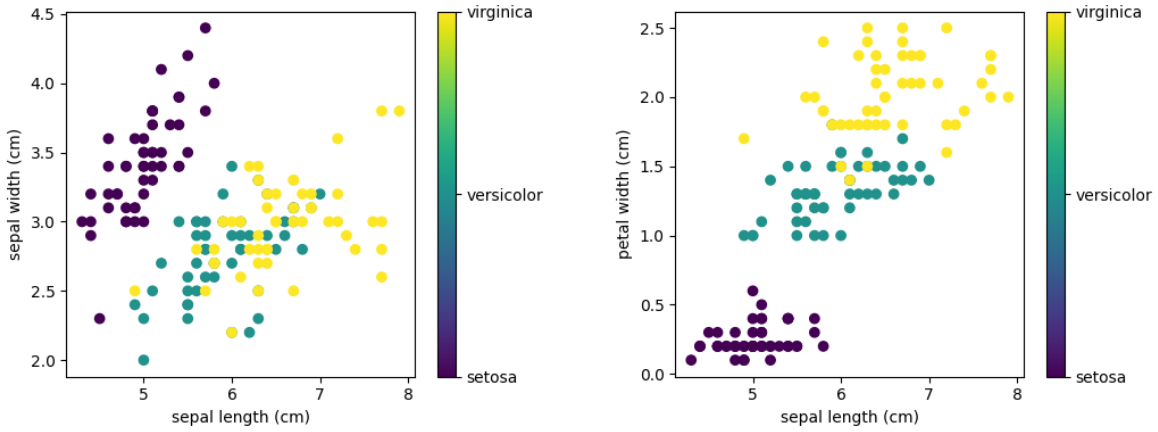
À esquerda da Figura 2 retrata dois atributos do dataset, *sepal_length* e *sepal_width*, enquanto à direita novamente retrata dois atributos, *sepal_length* e *petal_width*, onde ambos possuem marcadores coloridos para cada classificação por espécie. É possível identificar as classes à esquerda, porém não apresenta uma conclusão de predição apenas observando o dataset, enquanto à direita se torna visível a área de classificação por espécias. A escolha do Iris Dataset se deu por ser um dataset pequeno e sólido, trazendo assim resultados claros durante a aplicação do *kNN*.

2.2.1.4 Reprodução do Algoritmo

O estudo do algoritmo *kNN* consiste em utilizá-lo para classificação binária aplicado ao *Iris Dataset* e usando como referência bibliográfica o livro de Harrington (2012). A função de classificação *clasify0* foi dividida em etapas, sendo então a *calcDistance* para calcular a distância entre os vizinhos e *findMajorityClass* para calcular os votos a fim de classificar o dado de entrada. Para mensurar o desempenho do algoritmo e explorar sua taxa de erro de acordo com *k* vizinhos próximos, podemos calcular a partir do número de erros divididos pelo número de vetores testados (*ErrorCount* / *NumTestVecs*), sendo menor a taxa de erro quando é próxima a 0 e a maior é próxima a 1. Em algoritmo 1 consta o pseudocódigo do *kNN*.

Durante a reprodução do algoritmo, o dataset foi particionado em três porções distintas de dados para o treinamento, respectivamente 15, 37 e 75 amostras para validação que correspondem respectivamente à 10%, 25% e 50% do dataset.

Figura 2 – Iris Dataset



Fonte: Autor

Algoritmo 1: kNN - Pseudocódigo

Input: Dado de entrada inX a ser classificado.
Output: Retornar a predição de classe majoritária para a classe desconhecida de inX
for *cada ponto em nosso dataset* **do**
  calcular a distância entre inX e ponto atual
  ordenar as distâncias em ordem crescente
  pegar k itens que possuem a menor distância para inX
  definir a classe majoritária entre esses itens
end

Fonte: Harrington (2012)

A fim de explorar resultados consistentes, utilizei a biblioteca *Pandas* para embaralhar o dataset baseado em seu index, mantendo assim seus valores e somente alterando a ordem de posição deles e utilizando um vetor de vizinhos como k = [1,3,5,7,9], permitindo então visualizar os diversos estados de precisão do algoritmo. O código utilizado para a reprodução do algoritmo consta na Seção A.1.

2.2.1.5 Resultados e Conclusão

Podemos observar a Tabela 2 os resultados obtidos conforme o proposto na exploração do algoritmo. Fica claro que em ambas as porções de dataset, sendo 10%, 25% e 50% ao se aplicar $k = 9$, a taxa de erro aumenta consideravelmente ao ponto máximo de 20% de imprecisão. Da mesma maneira podemos observar $k = 1$ tendo as menores taxas de erro, porém é importante entendermos que não se enquadra como a melhor decisão caso seja aplicado em um ambiente real, já que o algoritmo associa a classe mais próxima e ignora o próprio contexto do dataset apresentado, classificando assim de forma duvidosa apesar da alta taxa de precisão.

Os resultados de $k = 5$ e $k = 7$ em todas as porções de datasets são muito próximos, se apresentando então como resultados não conclusivos dentre as outras porções de amostras da tabela. Excluindo então $k = 1$, podemos concluir que os melhores resultados são obtidos quando $k = 3$ aplicado à porção de 25% do dataset que obtém aproximadamente 8% de taxa de erro, a menor de todos os resultados apresentados. Os resultados provaram a alta precisão atribuída ao algoritmo *kNN*.

Tabela 2 – kNN - Taxa de Erro conforme os k vizinhos para diferentes porções do dataset

k	Dataset - 10%	Dataset - 25%	Dataset - 50%
1	0.066667	0.054054	0.080000
3	0.133333	0.081081	0.120000
5	0.133333	0.108108	0.160000
7	0.133333	0.135135	0.173333
9	0.200000	0.162162	0.200000

Fonte: Autor

2.2.2 Naive Bayes

O Naive Bayes (NB) é um algoritmo do tipo probabilístico para classificação binária e de múltiplas classes, baseado no *Teorema de Bayes* que é representado pela Equação (4). Sua aplicação é mais utilizada em classificação de texto, como análise de sentimento, ou problemas onde existem múltiplas classes. É chamado Naive, ou Ingênuo, porque os cálculos das probabilidades para cada classe são simplificados e parte da suposição que todos os atributos são independentes dado a classe alvo, ignorando então as possíveis dependências, correlações e grupos de problemas. Em aplicações no mundo real, esta característica o torna muito efetivo devido ser recorrente que os atributos que compõem o dataset não tenham dependências entre si (MOHAMMED; KHAN; BASHIE, 2016; HARRINGTON, 2012).

Teorema de Bayes:

$$P(A/B) = \frac{P(B/A)P(A)}{P(B)} \tag{4}$$

Uma vantagem do *NB* é que ele precisa de um pequeno número de dados de treinamento para estimar a classificação, o que resulta em um menor consumo de memória de processamento. Há três variáveis que influenciam consideravelmente no poder de precisão do NB, sendo: ruido no dataset de treino, a tendência e a variação. O impacto pelo dataset de treino pode ser reduzido ao se escolher um bom dataset com a menor quantidade de ruido, a tendência é os erros atribuídos aos grupos de classe do próprio dataset de treino e erros de variança são atribuídos quando o dataset é muito pequeno ou que os grupos de classes não possuem instâncias o suficiente (MUKHERJEE; SHARMA, 2012; SHALEV-SHWARTZ; BEN-DAVID, 2013).

2.2.2.1 Classificação Bayesiana

Em classificação de texto, todo o documento é considerado uma instância e os atributos são as palavras relevantes presentes no texto ou até mesmo emojis e outros caracteres. O *NB* cria um vetor de vocabulário, também chamado de *Bag of Words*, selecionando todas as palavras únicas presentes no documento. O treinamento do NB é feito através do método de *Maximum Likelihood Estimation*, onde as probabilidades são calculadas através da frequências das palavras do nosso vocabulário (ANDRÉS-FERRER; JUAN, 2010).

Dado a Equação (5), a C_i representa as *i* classes alvos e $Data$ representa um vetor de dados, no nosso caso, um vetor de palavras e P representa a probabilidade condicional. Dado um conjunto de *i* classes $C = (c1, c2, c3, ...,c_i)$, queremos atribuir uma classe *C* para um documento D. O NB calcula a probabilidade de $P(C|D)$ de um documento pertencer a determinada classe a partir da probabilidade $P(C)$ do documento ser da mesma classe e das probabilidades condicionais de cada palavra ocorrer em um documento. O objetivo do algoritmo é encontrar a melhor classe para um documento caso ela maximize sua probabilidade do documento *D* (MOHAMMED; KHAN; BASHIE, 2016; NILSSON, 1998).

Classificação Bayesiana:

$$P(C_i|D) = \frac{P(D|C_i)P(C_i)}{P(D)} \tag{5}$$

Para evitar o *Underflow*, um resultado de um cálculo onde o número de valor absoluto é menor do que o computador pode realmente representar na memória, o resultado das probabilidades é substituído pela soma dos logaritmos das probabilidades (HARRINGTON, 2012).

2.2.2.2 Data Set

O dataset utilizado é o "Twitter US Airline Sentiment", que foi baixado do Kaggle como um arquivo do tipo *CSV*, porém sua fonte original era da biblioteca *Data for Everyone* da Crowdflower, que hoje não se encontra mais disponível. O dataset é categorizado como *Open-Source* e licença **CC BY-NC-SA 4.0**. Os tweets foram retirados da rede social Twitter em fevereiro de 2015 sobre cada uma das principais companhias aéreas dos EUA. Os colaboradores então classificaram cada tweet como "positivo", "neutro"ou "negativo"e citaram o motivo de uma classificação negativa.

Existem 14.640 instâncias e 15 atributos, sendo uma delas o atributo de classificação. Os atributos incluídos são: tweet id, sentiment, sentiment confidence score, negative reason, negative reason confidence, airline, sentiment gold, name, retweet count, tweet text, tweet coordinates, time of tweet, date of tweet, tweet location, and user time zone.

2.2.2.3 Reprodução do Algoritmo

O estudo do algoritmo NB consistem em explorar a classificação de texto baseando-se na literatura de Harrington (2012), porém utilizando o dataset *Twitter US Airline Sentiment*, tornando o estudo mais próximo de uma aplicação no mundo real. O NB foi dividido em funções, sendo da seguinte forma: Dentro de *testingNB*, temos o *textParse* que transforma nossa instância em um conjunto de vetores para cada palavra e eliminando as palavras com menos de dois caracteres, a fim de explorar diferentes resultados, inseri um *regex* a fim de remover os emojis e urls inseridos nas instâncias e comparar taxa de precisão. O *createVocabList* recebe a saída de *textParse* e cria uma lista apenas com as palavras únicas no documento. Com o vocabulário pronto, podemos calcular as probabilidades e realizar a classificação com o *classifyNB* que recebe como parâmetros o documento teste, conjunto de classes, o vocabulário e as probabilidades estimadas durante o treinamento. Caso não seja classificado correto, um contador de erros será incrementado, assim podemos calcular a taxa de precisão. Em nossa aplicação, utilizamos apenas as 200 primeiras instâncias do dataset e 75 instâncias para a validação. Em algoritmo 2 consta o pseudocódigo do *NB*. O código utilizado para a reprodução do algoritmo consta na Seção A.2.

Algoritmo 2: Naive Bayes - Pseudocódigo

```
Input: Dado de entrada inX a ser classificado.
Output: Retorna a probabilidade para cada classe
for Cada Documento em treinamento do
    for Cada Classe do
        if Se o token apareceu no documento then
            Incremente o contador para aquele token
        for Cada Classe do
            for Cada Token do
                Divida a contagem de tokens pela contagem total de tokens para
                 obter as probabilidades condicionais
            end
        end
    end
end
```

Fonte: Harrington (2012)

2.2.2.4 Resultados e Conclusão

Durante a reprodução do NB, foi estipulado um *Regex* para visualizar as diferentes taxas de precisão a partir de determinadas palavras que foram eliminadas do vocabulário. O *Regex* pode remover todos os emojis, urls e as palavras: *VirginAmerica*, *you* e *your*. Dentre várias combinações, a remoção de todas as características acima elevou para 45% a taxa de erro. Os melhores resultados obtidos foi utilizando o *Regex* apenas para eliminar as urls e a

palavra *VirginAmerica*, onde foi possível alcançar 21% de taxa de erro. A Tabela 3 representa as saídas do algoritmo, onde cada resultado contém o vocabulário que foi incorretamente classificado. Como as instâncias de treinamento e validação são randômicas, para considerar a taxa de erro, foram utilizadas 10 amostras dos resultados e realizado uma média simples.

Tabela 3 – NB - Vocabulário classificado incorretamente

	Erro de Classificação
1	['well', 'didn', 'but', 'now', 'elevate']
2	['hold', 'times', 'call', 'center', 'are', 'bit', 'much']
3	['your', 'beautiful', 'front', 'design', 'down', 'right', 'now']
4	['everything', 'was', 'fine', 'until', 'you', 'lost', 'bag']

Fonte: Autor

O Quadro 1 contém um trecho do vocabulário construído com as 200 instâncias do dataset onde seu tamanho final consta com 897 palavras na lista.

Quadro 1 – NB - Lista do Vocabulário

NB - Lista do Vocabulário
['twitter', 'silicon', 'wonked', 'vabeatsjblue', 'view', 'needs', 'numofpointsavailable', 'impending', 'los', 'pressure', 'aesthetics', 'fine', 'much', 'improve', 'able', 'texas', 'diehardvirgin', 'pros', 'late', 'giving', 'minutes', 'dude', 'friday', 'recline', 'feet', 'heard', 'browsers', 'hands', 'taking', 'asap', 'appointments', 'gentleman', 'account', 'soreback', 'redesign', 'front', 'options', 'worse', 'policy', 'yall', 'face', 'happened', 'there', 'both', 'again', 'food', 'coffee', 'though', 'only', 'cant', 'francisco', 'open', 'rep', 'premium', 'worst', 'transformative', 'great', 'new', 'cheapflights', 'during', 'shaker', 'doing', 'center', 'want', 'via', 'yesterday', 'sweet', 'left', 'while', 'based', 'header', 'awaiting', 'hipster', 'iced', 'precipitation', 'persisting', 'running', 'hand', 'due', 'supp', 'password', 'the', 'painless', 'won', 'next', 'flying', 'change', ...]

Fonte: Autor

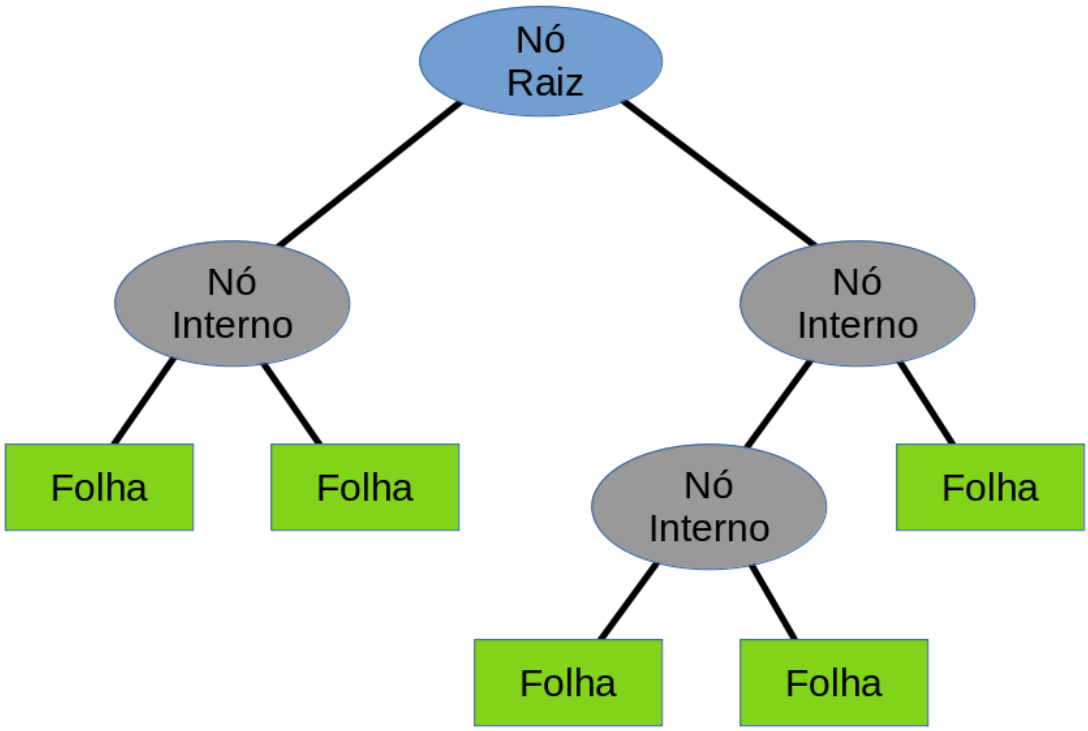
2.2.3 Decision Trees

O algoritmo de Decision Tree (DT) é definido como um modelo estatístico usado para classificação. Sua abordagem é classificar os dados em classes e representar os resultados em fluxogramas, semelhante a uma estrutura de uma árvore, percorrendo do nó raiz até as folhas. A raiz (root) representa o atributo principal da classificação, seus ramos (branches) representam a direção das decisões e suas folhas (leaf) representam a classe que será atribuída ao dado de entrada. Cada sucessor do nó raiz é definido pelas regras de divisão e percorrem caminho raiz $\rightarrow$ folha a fim de determinar uma característica específica (MOHAMMED; KHAN; BASHIE, 2016; SHALEV-SHWARTZ; BEN-DAVID, 2013).

A Figura 3 exibe um exemplo do fluxograma de *DT*. O nó raiz é a primeira característica que divide o dataset, o valor de entrada é comparado a ele e suas respostas são direcionadas

pelos ramos até um próximo nodo interno. Este ciclo se repete pelos nós internos até encontrar uma folha, do qual é possível atribuir uma classe ao valor de entrada. Uma grande vantagem é que o fluxograma gerado pelo *DT* pode ser compreendido com facilidade por humanos, mesmo não tendo um prévio conhecimento estatístico ou estar familiarizado com o algoritmo (HARRINGTON, 2012).

Figura 3 – Fluxograma - Decision Tree



Fonte: Autor

Uma das características que compõem o *DT* é ele ser capaz de utilizar dados multivalorados, que podem ser numéricos e não numéricos. Outra característica é que os resultados do aprendizado devem possuir duas ou mais saídas, sendo que se houver apenas duas saídas para classificação, podemos chamar este modelo de *Binary Decision Tree*. Apesar do algoritmo *DT* aceitar mais de duas classes, existe uma categoria específica chamada de *Boolean Decision Tree*, que se caracteriza quando possuímos duas classes binárias de entrada (NILSSON, 1998).

O maior problema de *DT* aplicado a atributos binários, é escolher qual é a melhor ordem para se iniciar os testes, diminuindo assim o tamanho da árvore e consequentemente o poder computacional necessário. Existem diversos métodos para selecionar a melhor ordem do atributo divisor ao se construir o *DT*, sendo que elas podem ser encontradas em áreas como: Teoria da informação, análise discriminante e técnicas de codificação. Os mais comuns são algoritmos *Interative Dichotomiser 3 (ID3)* e *C4.5*, sucessor do *ID3*. O *ID3* utiliza-se do conceito da *Entropia* para extrair do *Information Gain* de todos os atributos, assim como o

C4.5 utiliza-se do *Gain Ratio* (MOHAMMED; KHAN; BASHIE, 2016; RAILEANU; STOFFEL, 2004).

Podemos então observar que o DT percorre as seguintes etapas:

- Insere todos os exemplos de treino como raízes.
- Seleciona o atributo do nó raiz seguindo algum método de métrica.
- Seus ramos direcionam a um novo nó, o qual o particionamento recursivo continua até que nenhum exemplo de treinamento reste, ou até que nenhum atributo reste, ou os exemplos de treinamento restantes pertençam à mesma classe.

2.2.3.1 Entropia de Shannon

A entropia é usada na teoria da informação para determinar a pureza ou impureza de determinado conjunto. Se a amostra é completamente homogênea, a entropia é zero e se a amostra é igualmente dividida, possui uma entropia igual a um (MOHAMMED; KHAN; BASHIE, 2016; SHALEV-SHWARTZ; BEN-DAVID, 2013). Dado uma tabela e obtendo $p(X_i)$, que representa a probabilidade de escolher aquela classe e dado *n* como o número de classes, podemos utilizar a Equação (6) para calcular a entropia de Shannon.

Entropia de Shannon:

$$H(X) = -\sum_{i=1}^{n} p(X_i) \log_2 p(X_i) \tag{6}$$

Um método popular ao se utilizar os algoritmos ID3 e C4.5 é o *Information Gain* dado pela Equação (7), onde ele é a diferença entre a entropia da classe antes e depois da divisão, buscando encontrar atributos que retornem resultados mais homogêneos para a divisão do dataset.

Information Gain:

$$IG(X,Y) = H(X) - H(X|Y) \tag{7}$$

Como um exemplo prático, a entropia da Tabela 4 pode ser calculada da seguinte forma: $Entropia(Gostam\_de\_ML) = Entropia(4,11)$. Alterando então os valores pela probabilidade dos eventos, podemos dar continuidade a equação como: $Entropy(0.27,0.73) = -(0.27log_20.27) - (0.73log_20.73) = 0.8366$.

Tabela 4 – DT - Exemplo Entropia - Gostam de ML

Sim	Não
11	4

Fonte: Autor

2.2.3.2 Data Set

O *Lenses Dataset* é um conjunto de dados multivariados, introduzido por Cendrowska (1987) em seu artigo que explora um novo algoritmo baseado em ID3, propondo melhoras de compreensão e manipulação, bem como outros problemas associados ao *DT*. O conjunto de dados consiste em 24 instâncias que classificam qual tipo de lente será necessária para o paciente, como: *Hard Lenses*, *Soft Lenses* e *No Lenses*. Quatro atributos foram exibidos em cada instância: A idade do paciente (Young, Pre-presbyopic, Pre-presbyopic), Prescrição de óculos (Myope ou Hypermyope), Astigmático (Yes ou No) e Taxa de produção de lágrimas (Reduced ou Normal). Todos os exemplos do dataset estão completo e livres de ruido, onde as instâncias são classificadas e subdivididas em: 4 hard lenses, 5 soft lenses e 15 no lenses.

Tabela 5 – Lenses Dataset - Header

	Age	Prescript	Astigmatic	Tear Reate	Lenses
1	young	myope	yes	reduced	no lenses
2	young	myope	no	normal	soft
3	pre	myope	no	reduced	no lenses
4	pre	myope	no	normal	soft
5	presbyopic	hyper	no	reduced	no lenses

Fonte: Cendrowska (1987)

2.2.3.3 Reprodução do Algoritmo

O estudo do algoritmo *DT* consiste em utilizá-lo para classificação aplicado ao *Lenses Dataset* que foi utilizado no livro de Harrington (2012). O *DT* foi dividido em um ciclo de funções com suas respectivas etapas, sendo a primeira *calcShannonEnt*, que gera o valor da entropia de cada atributo para *n* classes. Dentre todos os valores de entropia, *chooseBestFeatureToSplit* será responsável por escolher o melhor atributo de divisão do dataset, onde esta divisão é recursiva, criando então novos nós internos e só se encerra quando não reste nenhum atributo ou os atributos pertençam à mesma classe. O resultado ao se criar o *DT* é uma lista do tipo dicionário (Dict) que contém todo o caminho do nó raiz $\rightarrow$ folhas que pode ser salvo em um arquivo de texto, a fim de que em uma próxima aplicação ou classificação, não tenha a necessidade de percorrer o *DT* novamente e recalcular todos os nós, ramos e folhas. O algoritmo 3 consta o pseudocódigo de *createBranch*, onde ele representa o ciclo do *DT*. Os resultados foram exportados em um dicionário e uma imagem. O código utilizado para a reprodução do algoritmo consta na Seção A.3.

2.2.3.4 Resultados e Conclusão

No Quadro 2, podemos observar o resultado do dicionário após calcular todo o caminho do *DT* aplicado ao dataset Lenses, esclarecendo o porque ele não é de fácil entendimento

Algoritmo 3: Decision Tree - createBranch - Pseudocódigo

```
Input: Dado de entrada inX a ser classificado.
Output: Retorna a tupla que representa o caminho de predição do DT
for Verifique em cada instância do dataset do
    if Se o conjunto de dados está na mesma classe then
        Retorne o rótulo da classe
    else
        Encontre o melhor atributo para dividir os dados
        Divide o dataset
        Cria um nó de ramificação (branch)
        for cada divisão do
            Chame createBranch e adicione o resultado ao nó do ramo
        end
        Retorne o nó da ramificação
    end
end
```

Fonte: Harrington (2012)

humano. Portanto a melhor opção é exibirmos ele em forma de fluxograma utilizando a biblioteca Matplotlib do Python3 a fim de visualizar suas saídas, seus ramos de decisões e os nós internos.

Quadro 2 – DT - Dicionário com o caminho nó raiz → folhas .

Resultado da Dicionário - Caminho DT nó raiz → folhas
tupla_saida = {'tearRate': {'normal': {'astigmatic': {'no': {'age': {'presbyopic': {'prescript': {'myope': 'no lenses', 'hyper': 'soft'}}, 'young': 'soft', 'pre': 'soft'}}, 'yes': {'prescript': {'myope': 'hard', 'hyper': {'age': {'presbyopic': 'no lenses', 'young': 'hard', 'pre': 'no lenses'}}}}}}}, 'reduced': 'no lenses'}}

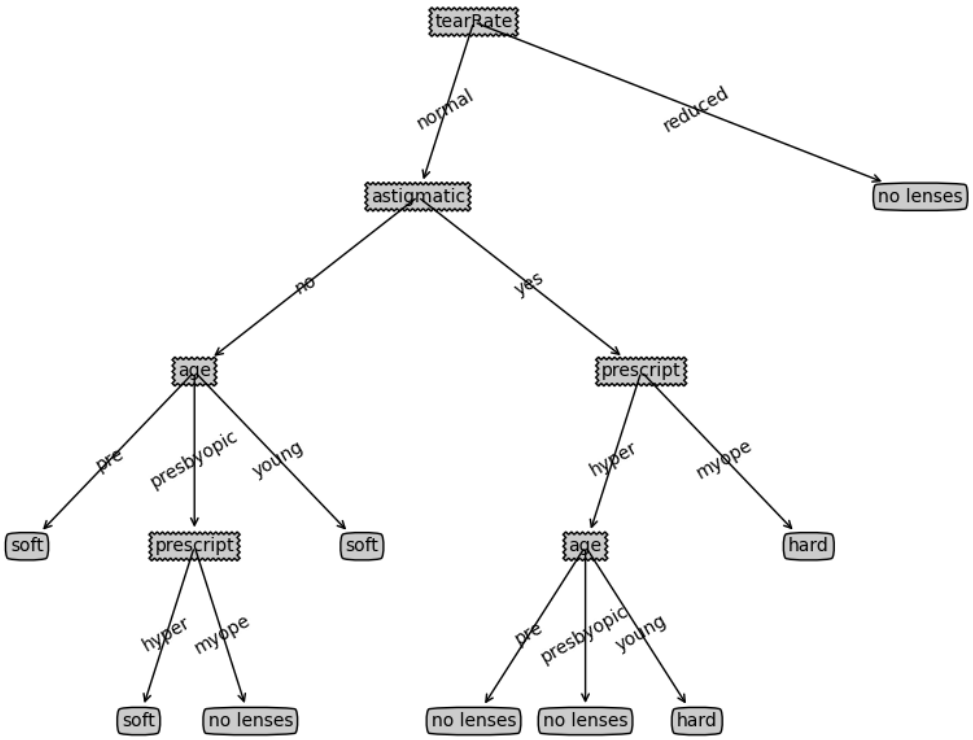
Fonte: Harrington (2012)

Podemos então observar na Figura 4 que a profundidade do *DT* é de 5 e quanto maior a profundidamente, mais complexo é o nosso modelo, demostrando então que o estudo conseguiu alcançar 9 folhas para classificar mas apenas os atributos *Astigmatic* e *Prescript* demonstraram ser nós internos com saídas binárias. Os resultados provaram o fácil entendimento humano dos resultados pela visualização da imagem e a alta precisão atribuída ao algoritmo durante o estudo.

2.2.4 Regressão Logística

A Regressão Logística (*RLOG*) é um algoritmo de classificação usado de forma binária ou multilinear em um conjunto discreto de classes. A *RLOG* utiliza um método de otimização e sua saída será redirecionada para a função sigmoide, que retorna um valor de probabilidade entre o intervalo $(0,1)$. O objetivo é aprender os pesos ou parâmetros da equação que melhor

Figura 4 – Fluxograma - Decision Tree aplicado ao Dataset Lenses



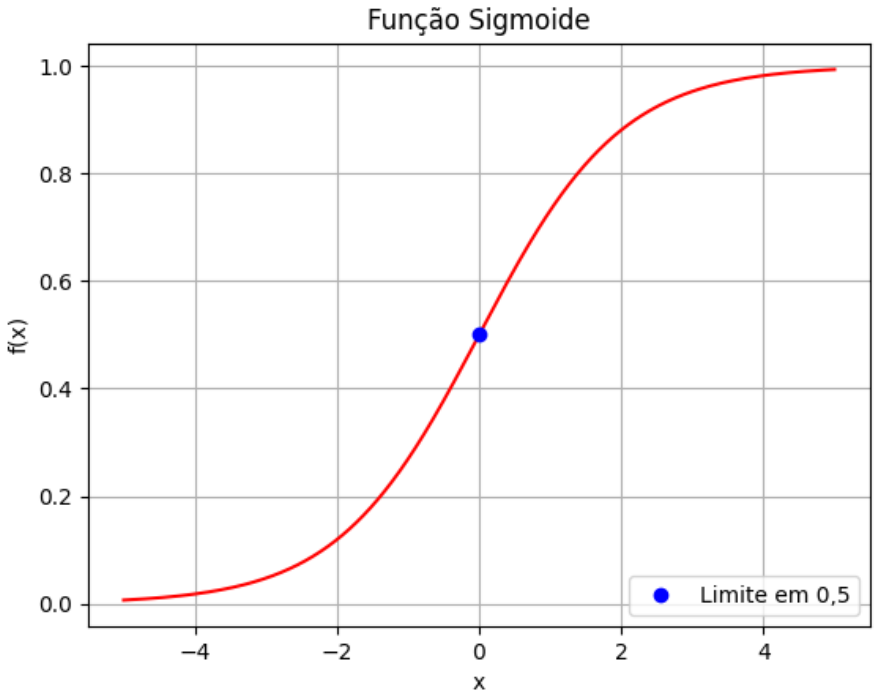
Fonte: Harrington (2012)

descrevem a relação entre os preditores e a variável de resposta para utilizá-los em uma nova observação (KUHN; JOHNSON et al., 2013). O conjunto de saídas ou classes retorna uma pontuação de probabilidade que deve estar limitada ao intervalo $(0,1)$ e apresentando então uma relação direta com os preditores mensurados para cada observação do dataset. Apesar de muitos comparativos, muitos se questionam entre a diferença entre regressão linear e *RLOG*, sendo que para a regressão linear, a constante Y sempre é contínua (BRZEZINSKI; KNAFL, 1999).

Em *RLOG* é necessário multiplicar cada atributo por um peso e então adicionar esta saída à *Função Sigmoide*. Com isso podemos compreender a *RLOG* como uma estimativa de probabilidades (HARRINGTON, 2012). Também é necessário definirmos o valor limite ou a fronteira de decisão, que define a fronteira entre as classes dado *f(x)*. Como exemplo podemos observar a Figura 5, temos o valor limite de 0,5 e as classes *c0* e *c1*. Se a função de previsão retornasse um valor de 0,8, então classificaríamos esta observação como *c1*. Se nossa previsão retornasse um valor de 0,4, então classificaríamos a observação como *c0*.

Podemos então observar a composição de *RLOG* em quatro partes:

Figura 5 – Função Sigmoide



Fonte: Autor

- Aplicação da *Função Sigmoide* ou também referenciada como de *Função Logística*.
- Um método de otimização ou regressão de coeficientes.
- Um valor para a fronteira de decisão.
- Uma grande porção do dataset para a etapa de treinamento.

2.2.4.1 Função Sigmoide

O nome "Sigmoide"significa "em forma de S", referindo-se ao gráfico da função mostrado na Figura 5 (SHALEV-SHWARTZ; BEN-DAVID, 2013). O valor de retorno de uma função sigmoide, sendo o eixo *y*, se categoriza por estar no intervalo de $(0,1)$ ou de $(-1,1)$. A função sigmoide é dada pela Equação (8) , onde podemos entender a variável *z* como os pesos gerados por algum método de otimização e tornando então em nosso valor de entrada para a função sigmoide.

Função Sigmoide:

$$\alpha(z) = \frac{1}{1 + e^{-(z)}} \tag{8}$$

2.2.4.2 Método Gradiente Estocástico

O Gradiente Estocástico (GE) é um método de iterativo de otimização, onde o parâmetro selecionado é atualizado a cada iteração para cada instância do dataset, buscando o ponto

máximo ou mínimo da função, se direcionando assim de uma melhor forma em direção ao gradiente (KUHN; JOHNSON et al., 2013; HARRINGTON, 2012). Em *ML* é comum se utilizar o método *Descendente* em Redes Neurais e em nosso caso, utilizar o método *Ascendente* aplicado em *RLOG*. A diferença entre utilizar o *Método Gradiente* ou o *GE*, seja aplicado de forma *Ascendente* ou *Descendente*, é que o *Método Gradiente* utiliza o dataset completo à cada atualização de pesos, enquanto o *GE* utiliza apenas uma instância por vez para atualizar os pesos (HARRINGTON, 2012). Podemos visualizar as equações de ambos os métodos, onde fica visível que a diferença está no sinal aplicado em cada função, sendo assim, se estamos procurando o ponto mínimo da função utilizando o método *Descendente*, utilizamos a Equação (10) e para o ponto máximo da função utilizando o método *Ascendente*, utilizaremos a Equação (9).

Gradiente Estocástico Ascendente:

$$w := w + \alpha \nabla_w f(w) \tag{9}$$

Gradiente Estocástico Descendente:

$$w := w - \alpha \nabla_w f(w) \tag{10}$$

Dado *z*, representado na função sigmoide como nosso valor de entrada, precisamos de um vetor dado por *x* e nosso dataset. É necessário que esses dois vetores de números sejam multiplicados para cada elemento e adicionados para gerar um número final, nosso *z*. Podemos simplificar por $z = w^T x$, ou de forma mais explícita, $z = w_0 x_0 + w_1 x_1 + \cdots + w_n x_m$, sendo então um incremento de pesos por iteração.

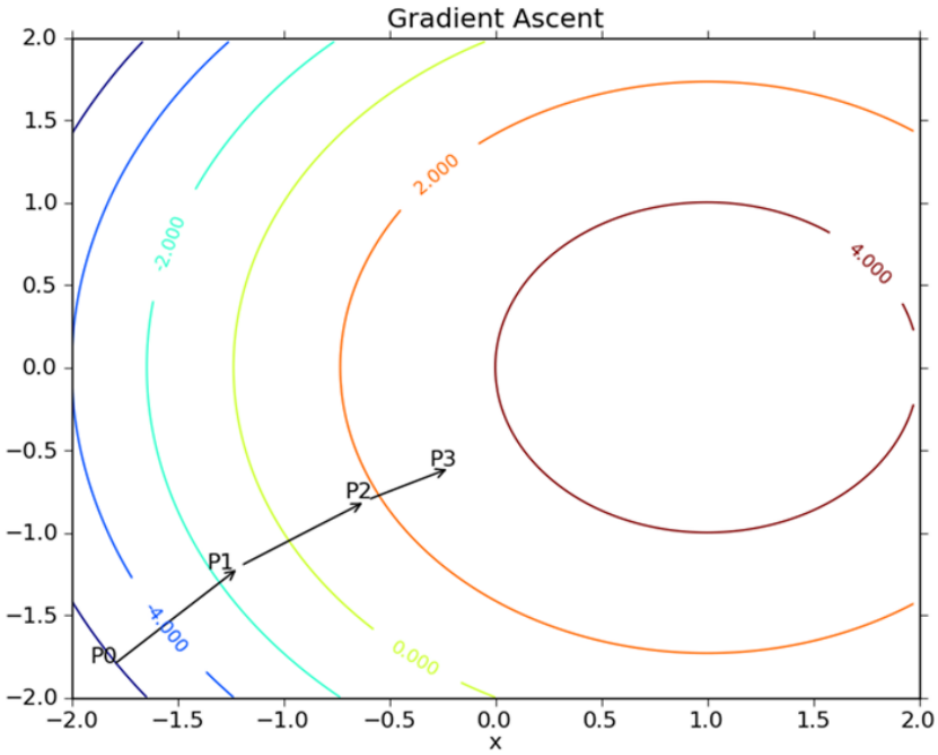
A Figura 6 exibe a direção dos pontos, sendo que a partir de cada atualização, os pesos são alterados e é possível visualizar a direção tomada a partir da otimização do método. Esse ciclo se repete até a condição ser atingida.

2.2.4.3 Data Set

O *Horse Colic Dataset* é um conjunto de dados multivariados que apresenta atributos inteiros, reais e categóricos. Foi introduzido por McLeish, Cecile e Lopez-Suarez (1989) em seu artigo que explora o desenvolvimento de um protótipo utilizando IA e ML que ajudaria na tomada de decisão se um cavalo seria ou não transportado para cirurgia, baseado nos sintomas de cólica apresentados como dado de entrada a ser classificado e no dataset que contém 10 anos de dados que foram coletados do hospital veterinário.

O conjunto original de dados consiste em 368 instâncias que foram particionadas como 300 instâncias para treinamento e 68 instâncias para validação e composto por 27 atributos e uma classe alvo, onde o cavalo pode ser classificado como: *Lived*, *died* e *euthanized*. Durante a reprodução do dataset, as classes alvo foram simplificadas em *Lived* e *Non-Lived*, conforme consta na mesma reprodução de Harrington (2012).

Figura 6 – Gradiente Ascendente



Fonte: (HARRINGTON, 2012)

Tabela 6 – Horse Colic Dataset - Header

	Surgery	Age	Pulse	Pain	Abdomen
1	2	1	20	4	42.00
2	2	1	36	3	44.00
3	1	1	28	4	45
...	...	...	...	...	...
367	1	1	12	1	37.00
368	2	1	36	3	44.00

Fonte: McLeish, Cecile e Lopez-Suarez (1989)

2.2.4.4 Reprodução do Algoritmo

O estudo do algoritmo consiste em utilizá-lo para a classificação aplicado ao *Horse Colic* Dataset que foi utilizado no livro de Harrington (2012). A função *multiTest* define o valor de 10 como o número de instâncias que serão testadas para validação e inicializa o *RLOG* como um todo chamando a função *colicTest*. Dentro de *colicTest*, o dataset é separado entre instâncias de treinamento e de validação, em seguida são obtidos os pesos de treino pela variável *trainWeights* dado os vetores de classes e de atributos. Para cada linha do nosso dataset de treino, *classifyVector* recebe os valores da instância mais os pesos já gerados pela função *stocGradAscent1*, onde será direcionado e aplicado na função *sigmoid*. Com a fronteira

de decisão definida por 0.5, a função *sigmoid* retorna como 1 caso seja acima e 0 caso o valor seja abaixo da fronteira de decisão. Ao final, conseguimos uma média da taxa de erro dado pela soma das taxas de erro dividida por o número total de testes. O algoritmo 4 representa o pseudocódigo do *stocGradAscent1* que calcula os pesos para cada instância do dataset, onde estes pesos serão um valor de entrada para o algoritmo *sigmoid*. O código utilizado para a reprodução do algoritmo consta na Seção A.4.

Algoritmo 4: Regressão Logística - stocGradAscent1 - Pseudocódigo

Input: Dado de entrada inX a ser classificado.
Output: Retorna o peso dos vetores
Todos os valores de pesos são igual a 1
for *Cada pedaço de data no dataset* **do**
    Calcule o gradiente para um pedaço do dataset
    Atualize os pesos dos vetores por alpha * gradiente
end

Fonte: Harrington (2012)

2.2.4.5 Resultados e Conclusão

Como citado em seu livro, o algoritmo utilizado possui uma taxa de erro de 35%, porém com algumas alterações é possível atingir até 20% de taxa de erro. No decorrer da reprodução do *RLOG*, foi testado diferenças valores para alpha como também diferentes valores de iterações na função *colicTest*, se tornando possivel otimizar mais ainda a taxa de precisão do algoritmo conforme cita Harrington (2012). Apesar de não atingir resultados próximos de 20% de taxa de erro conforme citado em seu livro, consegui atingir 27% de taxa de erro dado as sugestões de alteração dos valores de alpha e das quantidades de iterações.

Dados os exemplos, fica claro observar como o método *RLOG* é simples e de fácil aplicação, otimizando os pesos conforme as iterações e o quanto utilizar o método *GE* reduz a quantidade computacional necessária. Dado a um dataset com cerca de 30% de seus dados sem valores, é surpreendente a taxa de precisão apresentada durante o estudo.

2.2.5 Regressão Linear

A Regressão Linear (RL) é uma abordagem linear para modelar a relação entre uma variável dependente (Y) e uma ou mais variáveis independentes (X), sendo então nosso objetivo traçar uma linha que consiga estimar a melhor relação entre X e Y, onde o valor previsto pode ser de natureza discreta ou quantitativa. Por traçar a melhor linha, a distância total de todos os pontos de regressão deve ser mínima. Antes de tentar ajustar um modelo linear aos dados, deve-se determinar se há ou não uma relação entre as variáveis de interesse (KUHN; JOHNSON et al., 2013). Uma das medidas numéricas de associação entre duas variáveis é

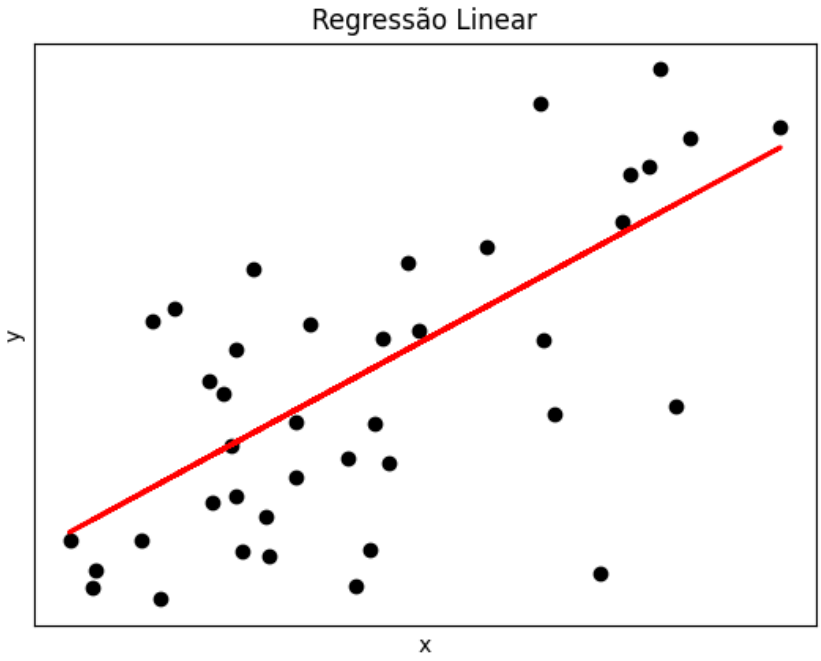
o coeficiente de correlação, que é um valor entre -1 e 1 indicando a força da associação dos dados para as duas variáveis.

Regressão Linear Simples:

$$Y_i = \beta_0 + \beta_1 X_i + e_i \tag{11}$$

Na Equação (11), Y_i é a variável dependente, X_i é a variável de resposta e e_i são os fatores de erros. β_0 e β_1 são os parâmetros a serem estimados, onde β_0 equivale ao ponto onde ocorre a interceptação da reta com o eixo vertical e o β_1 representa o coeficiente de regressão (KUHN; JOHNSON et al., 2013).

Figura 7 – Regressão Linear



Fonte: Autor

2.2.5.1 Método dos Mínimos Quadrados

O método mais comum para ajustar uma linha de regressão é o Método dos Mínimos Quadrados (MMQ). Essa técnica de otimização gera o melhor ajuste para um conjunto de dados tentando minimizar a soma dos quadrados das diferenças entre o valor estimado e os dados observados, calculando então uma linha de melhor ajuste, minimizando a soma dos quadrados dos desvios verticais de cada ponto de dados para a linha (KUHN; JOHNSON et al., 2013). Escrevendo como notação de matriz, podemos observar na Equação (12) e aplicar a zero a função para obtermos $\hat{w}$ como a nossa melhor estimativa (HARRINGTON, 2012).

Método dos Mínimos Quadrados:

$$\hat{w} = (X^T X)^{-1} X^T y \tag{12}$$

2.2.5.2 Métodos de Encolhimento

Os métodos de encolhimento são técnicas onde não selecionamos as variáveis de forma explícita, mas sim ajustamos um modelo contendo todos os preditores ou regularizando as estimativas de coeficiente, reduzindo assim para zero as estimativas em relação ao estimativas de mínimos quadrados . Esses métodos não usam mínimos quadrados completos para ajustar, mas sim critérios diferentes que têm uma penalidade cada modelo por ter um grande número de coeficientes ou um grande tamanho de coeficientes e que reduzirá os coeficientes normalmente a zero. Esses podem ser aplicados a datasets grandes ou em situações onde você tem mais atributos do que pontos de dados (HARRINGTON, 2012). Dentre os diversos métodos de encolhimento, temos a *Regressão de Ridge*, *Forward Stagewise Linear*,*Lasso* e dentre outros. Abaixo abordaremos apenas os dois primeiros citados.

Na *Regressão de Ridge* (RR) é adicionado uma matriz λI, onde I é dado por n x m onde todos os elementos em diagonal são 1 e o restante são 0. Com a *RR*, a função de perda de *MMQ* é aumentada para minimizamos a soma dos resíduos quadrados e também penalizamos o tamanho das estimativas dos parâmetros, a fim de reduzi-los para zero. Originalmente, a *RR* foi desenvolvida para lidar com problemas onde existam mais atributos do que pontos de dados(KUHN; JOHNSON et al., 2013). Apesar de adicionarmos *bias* para as estimativas, podemos também utilizar λ para impor um valor máximo para a soma de todos os pesos (HARRINGTON, 2012). A Equação (13) descreve *RR* em notação de matriz.

Regressão de Ridge:

$$\hat{w} = (X^T X + \lambda I)^{-1} X^T y \tag{13}$$

Outro método também de encolhimento é o *Forward Stagewise Linear* (FSL), um método de regressão linear que gera um vetor de coeficientes a cada iteração e atualiza apenas o coeficiente correspondente à variável mais correlacionada com o vetor de resíduos, onde a cada passo ele faz a decisão que melhor reduza o erro para o próximo passo (KUHN; JOHNSON et al., 2013; SHALEV-SHWARTZ; BEN-DAVID, 2013). O *FSL* é muito similar ao método de encolhimento *Lasso*, porém computacionalmente é mais simples e efetivo em datasets com maiores quantidades de atributos e instâncias (HARRINGTON, 2012). O pseudocódigo do *FSL* consta no algoritmo 5.

2.2.5.3 Data Set

O *Boston House Prices* é um conjunto de dados numéricos e foi introduzido por Harrison e Rubinfeld (1979) em seu artigo que explora os problemas metodológicos associados ao uso de dados do mercado imobiliário na área metropolitana de Boston, onde foi gerado as estimativas

quantitativas a respeito da disposição de se pagar por uma melhor qualidade do ar. Este dataset é de comum utilização em estudos e artigos de ML que abordam problemas de regressão.

O conjunto de dados consiste em 506 instâncias que são compostas por 14 atributos, sendo um deles o nosso alvo, que seria o *Valor médio das casas ocupadas pelo proprietário em U$ 1.000* (MEDV) e os 13 atributos restantes sendo: Taxa de crime per capita por cidade (CRIM), proporção de terreno residencial zoneado para lotes com mais de 25.000 pés quadrados (ZN), proporção de acres de negócios não varejistas por cidade (INDUS), Charles River Dummy Variable (CHAS), concentração de óxidos nítricos (NOX), número médio de quartos por habitação (RM), proporção de unidades ocupadas pelos proprietários e construídas antes de 1940 (AGE), distâncias ponderadas para cinco centros de emprego de Boston (DIS), índice de acessibilidade a rodovias radiais (RAD), taxa de imposto de propriedade de valor total por $ 10.000 (TAX), proporção aluno-professor por cidade (PTRATIO), 1000 $(Bk - 0{,}63)^2$ onde Bk é a proporção de negros por cidade (B) e percentual inferior de status da população (LSTAT).

Tabela 7 – Boston House Prices - Header

	CRIM	ZN	INDUS	...	B	LSTAT	MDEV
1	0.00632	18.0	2.31	...	396.90	4.98	24.0
2	0.02731	0.0	7.07	...	396.90	9.14	21.6
3	0.02729	0.0	7.07	...	392.83	4.03	34.7
4	0.03237	0.0	2.18	...	394.63	2.94	33.4
5	0.06905	0.0	2.18	...	396.90	5.33	36.2

Fonte: Harrison e Rubinfeld (1979)

2.2.5.4 Reprodução do Algoritmo

O estudo do algoritmo consiste em utilizar o *Boston House Prices Dataset* e aplicar a regressão a fim de prever os valores de *MEDV*. No início é utilizado a função *standRegres* a fim de encontrar a linha de melhor ajuste, onde este parâmetro é armazenado na variável *ws*. A função *crossValidation* recebe os arrays de *x,y* e um número para delimitar quantas validações cruzadas serão efetuadas. Dividindo o dataset como 90% para treino e 10% para validação, essas listas serão randomizadas. Criando então uma matrix para armazenar todos os coeficientes de *Ridge Regression*, em nossa reprodução utilizamos 30 valores para λ com diferentes pesos. Inicia o loop para executar os 30 diferentes pesos e todos os erros são utilizados pela função *rssError* para posteriormente calcularmos a taxa de erro. Após todas as validações cruzadas, *errorMat* é a variável que armazena o número de erros estimados e então calculados. Então podemos comparar as saídas com *standRegress*.

A fim de explorar diferentes resultados, também foi utilizado o método *Forward Stagewise Linear* para observar diferentes formas de otimização e como é o diferente comportamento de pesos para cada atributo, mas durante o decorrer dos testes os resultados com *Regressão de*

Ridge foram mais satisfatórios. O algoritmo 5 consta o pseudocódigo para *Forward Stagewise Linear* que foi utilizado no estudo. O código utilizado para a reprodução do algoritmo consta na Seção A.5.

Algoritmo 5: Regressão Linear - Forward Stagewise Linear - Pseudocódigo

```
Input: Dado de entrada inX
Output: Retorna o MelhorPeso
Regularize os dados para ter a média 0 e variação unitária
for Cada iteração do
    Defina lowerError para + Inf
    for Para cada atributo do
        for Cada t do
            Mude um coeficiente para obter um novo Peso
            Calcule o erro com o novo Peso
            if O erro for menor do que lowerError: then
                Definir MelhorPeso para o Peso atual
            end
            Atualizar conjunto de Pesos para MelhorPeso
        end
    end
end
```

Fonte: Harrington (2012)

2.2.5.5 Resultados e Conclusão

Durante a reprodução de *RL*,foi utilizado o método *Forward Stagewise Linear* primeiramente para ter um melhor entendimento do modelo e então seguimos utilizando o método de encolhimento de *Regressão de Ridge* e consequentemente tivemos bom resultados, tanto pela escolha do próprio método quanto pela utilização do *Boston House Prices*, que é um dataset sem ruídos e sem valores nulos. Dado nosso termo de constante ser o valor médio esperado de Y quando todo $X = 0$, o resultado do termo de constante foi de *33.1532* e chegamos no melhor modelo de coeficientes apresentado no Quadro 3.

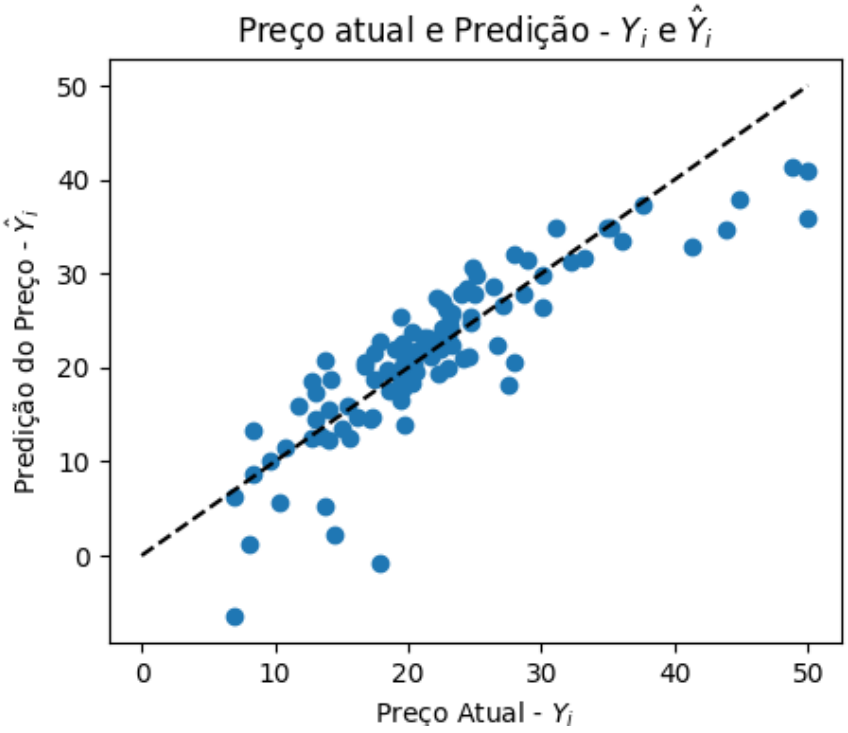
Quadro 3 – RL - Melhores Constantes - Ridge Regression

Melhores Constantes - Ridge Regression
[-1.28119023e-01 5.11339113e-02 -2.96980335e-02 2.48951428e+00 -8.48187731e+00 3.42539775e+00 -4.02323845e-03 -1.37995760e+00 3.49347811e-01 -1.39133697e-02 -8.79583093e-01 1.19877320e-02 -6.08894194e-01]

Fonte: Autor

Na Figura 8 podemos visualizar as variáveis Y_i para os preços atuais e $\hat{Y}_i$ para a predição dos preços. As melhores predições se concentram na diagonal do gráfico acompanhando a linha tracejada, indicando então um bom ajuste para o método de regressão.

Figura 8 – Preço atual e Predição - Y_i e $\hat{Y}_i$



Fonte: Autor

2.2.6 K-Means

O algoritmo *K-means* é um modelo não supervisionado que aloca matematicamente cada ponto de dados a um agrupamento, também chamado de centroide ou cluster, e assim consegue distinguir os padrões que um determinado conjunto de dados possui. Ao se inicializar o modelo, devemos decidir previamente a quantidade de centroides utilizados. Uma das desvantagens de utilizar o *K-means* é por ele ser computacionalmente lento, principalmente em datasets extensos (ALSABTI; RANKA; SINGH, 1997).

A Figura 9 nos exibe uma simplificação de como ocorre a classificação automatizada. Dado o número de centroides ser $k = 3$ e *n* pontos do conjunto de dados, o algoritmo encontrou um padrão baseado um uma métrica de cálculo de distâncias. Quanto maior *k*, maior a quantidade de padrões que o *K-means* consegue descobrir.

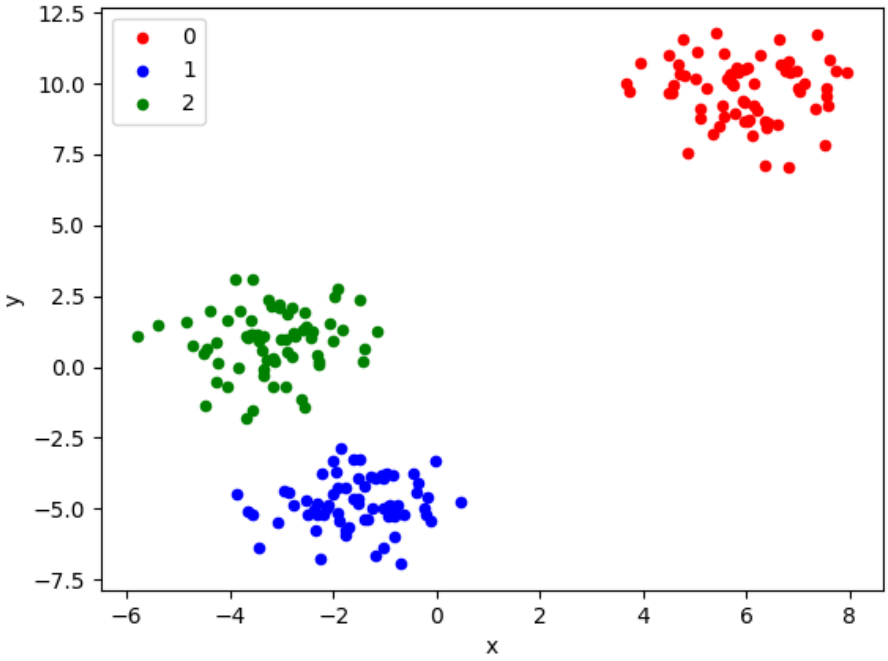
Cada ponto $x_i = x_1,...,x_n$ pertence a um centroide $C_j = C_1,...,C_k$ que foi centralizado no ponto $\mu_j = \mu_1,...,\mu_k$. O *K-means* é descrito pela Equação (14).

K-means:

$$E = \sum_{j=1}^{k} \sum_{i_t \epsilon C_j} (x_i - \mu_i)^2 \tag{14}$$

O método de agrupamento de k-médias é um método iterativo, que começa com uma

Figura 9 – K-means



Fonte: Autor

seleção aleatória de *k* ou μ_1,μ_2,...μ_k. Então a cada iteração, os pontos de dados são agrupados em k centroides, de acordo com a média mais próxima de cada um dos pontos, e a média é atualizado de acordo com os pontos dentro do centroide. Este processo de agrupar os pontos de dados de acordo com as médias dos centroides e a atualização do cluster significa que de acordo com o conjunto de pontos no centroide, continua até que não haja mais mudanças nos centroides já alterados, nem reste centroides Mohammed, Khan e Bashie (2016), Shalev-Shwartz e Ben-David (2013).

O algoritmo *K-means* procura soluções locais com relação ao erro de agrupamento, sendo iterativo e mais rápido, tornando-se então amplamente usado. É um método de agrupamento baseado em pontos que começa com os centros do cluster inicialmente colocado em posições arbitrárias e prossegue movendo a cada passo os centros do cluster para minimizar o erro de agrupamento. A principal desvantagem desse método está em sua sensibilidade às posições iniciais dos centros do cluster. Cada vez que executamos o *K-means*, ele pode fornecer diferentes meios e clusters dado pela seleção aleatória das k-médias iniciais. Para obter soluções com ótimo desempenho utilizando o *K-means*, várias execuções de centros do cluster devem ser programadas em diferentes posições iniciais a fim de buscar uma melhor otimização (MOHAMMED; KHAN; BASHIE, 2016; SHALEV-SHWARTZ; BEN-DAVID, 2013; HARRINGTON, 2012).

O critério de agrupamento mais amplamente usado é a soma das distâncias euclidianas quadradas entre cada ponto de dados x_i para cada centroide do subconjunto C_k que contém

X_i. Este critério é denominado erro de agrupamento e depende dos centros do agrupamento já calculados (HARRINGTON, 2012; MOHAMMED; KHAN; BASHIE, 2016).

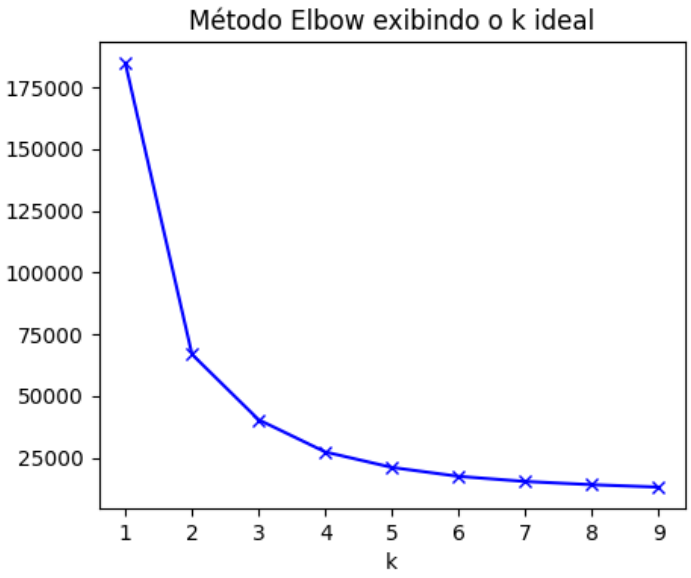
2.2.6.1 Calculando a Distância

Na Subsubseção 2.2.1.1, foram apresentados diversos métodos de cálculo de distância, inclusive a *Distância Euclidiana* e suas variações. Em *K-means*, utilizamos a *Distância Euclidiana* dado com $p = 2$.

2.2.6.2 Escolhendo K

Um método de escolha do valor *k* é o método *Elbow*. Neste método, o *K-means* é executado para um intervalo de *K* valores, como por exemplo de 1 à 10 e se calcula então a *Soma do Erro Quadrático* (SSE). O *SSE* é calculado como a distância média entre os pontos de dados e o seu centroide do cluster e em seguida gerando um gráfico de linha para valores SSE para cada *k*. Se o gráfico de linha se parece com um braço flexionado, então o cotovelo no braço é o melhor valor para *k* Mohammed, Khan e Bashie (2016), Shalev-Shwartz e Ben-David (2013). Na Figura 10 podemos observar que o cotovelo do gráfico seria o valor de $k = 3$, sendo então nossa melhor escolha de *k*.

Figura 10 – Método Elbow - Exibindo o *k* ideal



Fonte: Autor

2.2.6.3 Data Set

O *Wine Dataset* é um conjunto de dados numéricos disponibilizado pelo UCI Machine Learning Repository, que é uma coleção de datasets entre outros e amplamente usado pela

comunidade para a análise empírica de algoritmos de aprendizado de máquina. O arquivo foi criado como um arquivo ftp em 1987 por David Aha e seus colegas estudantes de graduação na UC Irvine. O dataset trata sobre os resultados de uma análise química de vinhos cultivados na região da Itália, mas derivados de três cultivares diferentes. A análise determinou as quantidades de 13 constituintes encontrados em cada um dos três tipos de vinhos e assim classificando a qualidade do vinho.

Existem 1600 instâncias com 13 atributos, sendo uma delas o atributo de classificação alvo *quality*. Os outros 12 atributos são: Fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH sulphates e alcohol.

Tabela 8 – Wine Dataset - Header

	Fixed Acidity	Volatile Acidity	Citric Acid	...	Sulphates	Alcohol
1	7.4	0.700	0.00	...	0.56	9.4
2	7.8	0.880	0.00	...	0.68	9.8
3	7.8	0.760	0.04	...	0.65	9.8
4	11.2	0.280	0.56	...	0.58	9.8
5	7.4	0.700	0.00	...	0.56	9.4

Fonte: UCI Machine Learning Repository

2.2.6.4 Reprodução do Algoritmo

O estudo do algoritmo consiste em utilizar o *Wine Dataset* e aplicar o modelo de agrupamento *K-means* a fim de entender o relacionamento entre determinadas variáveis e assim efetuar uma classificação. O código utilizado para a reprodução do algoritmo consta na Seção A.6 e o algoritmo 6 descreve o pseudocódigo do *K-means*.

A principal função do estudo é a *cluster*, onde já consta um número pré-determinado de centroides igual a *5* e iniciamos então carregando o dataset. Chamamos a função *biKmeans* que pede como parâmetros o nosso array de dados, o número de centroides e qual o método de métrica utilizado. Se inicia a função *kmeans* que cria uma matriz para armazenar as associações dos clusters para cada ponto do dataset e outra para armazenar os erros e logo após a função *randCent* cria *k* centroides randômicos para o dataset a partir dos valores mínimos e máximos da dimensão do dataset. O erro é a distância entre o centroide para o determinado ponto, com essa medida podemos medir qual a eficácia dos clusters. Você itera até que nenhum dos pontos de dados altere seu cluster criando então uma flag chamada *clusterChanged* e caso seja *True*, a iteração continua. Após efetuar esse loop calculando as distâncias para todos os centroides, os valores são atualizados e é obtido uma média desses pontos, onde os centroides e as associações são retornadas.

Medindo então o número de clusters para o número de itens na lista do cluster, inicia a iteração em todos os clusters e encontre o melhor cluster para dividir. É necessário comparar o *SSE* após cada divisão, onde se inicializa o SSE mais baixo para o infinito e então inicia o loop

Algoritmo 6: K-means - Pseudocódigo

Input: Dado de entrada inX
Output: Retorna os agrupamentos
Crie k pontos para começar os centroides
while *Qualquer ponto que mudou a atribuição do cluster* **do**
  for *Cada ponto no dataset* **do**
    for *Cada centroide* **do**
      Calcular a distância entre o centroide e o ponto
    end
    Atribuir o ponto ao cluster com a menor distância **for** *Cada cluster calcule a média dos pontos nesse cluster* **do**
      Atribuir o centroide à média
    end
  end
end

Fonte: Harrington (2012)

em cada cluster para a lista de agrupamentos *centList*. Para cada um desses clusters, você cria um conjunto de dados com apenas os pontos daquele cluster, denominado por *ptsInCurrCluster* e é alimentado em *K-means*. O *kmeans* gera novos centroides com os erros para todo o dataset. Se essa divisão produzir o *SSE* mais baixo, ela será salva. Depois decidir qual cluster dividir, as atribuições de cluster são atualizadas e o novo centroide é anexado para *centList*. Quando o loop termina, a lista de centroides e as atribuições do cluster é retornada assim como na função *kmeans*. O algoritmo 7 descreve o pseudocódigo do *Bisecting K-means*.

Algoritmo 7: Bisecting K-means - Pseudocódigo

Input: Dado de entrada inX
Output: Retorna um agrupamento com menor erro
Comece com todos os pontos em um cluster
while *O número de clusters seja inferior a k* **do**
  for *Cada cluster* **do**
    Mensure o erro total Execute o agrupamento k-means com k $= 2$ no cluster
    Calcule o erro total após k-means dividir o cluster em dois
  end
  Escolha a divisão do cluster que que contem o menor erro
end

Fonte: Harrington (2012)

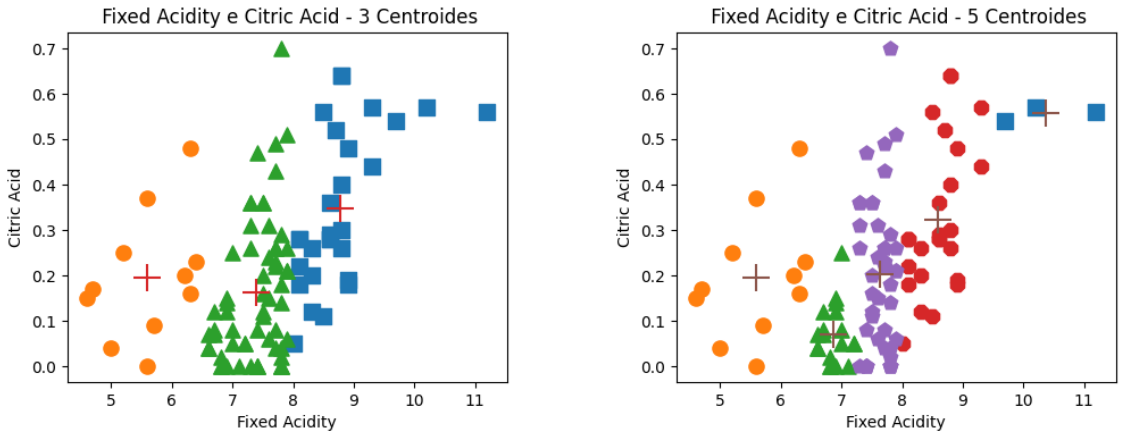
2.2.6.5 Resultados e Conclusão

Para uma melhor visualização dos gráficos gerados, a reprodução do *K-means* foram utilizadas apenas 100 instâncias do dataset. Também como método para explorar os diferentes comportamentos, alguns atributos específicos foram selecionados para executar a classificação

por agrupamentos. Foi realizada a reprodução entre os atributos na seguinte distribuição: 1 - Fixed acidity e citric acid, 2 - total sulfur dioxide e pH. Entre os 2 grupos, efetuamos a reprodução utilizando 3 e 5 centroides.

Na Figura 11, podemos visualizar que utilizando 3 centroides, as definições das similaridades são baseadas de acordo com o eixo *Fixed Acidity*, enquanto utilizando os agrupamentos com 5 centroides, fica mais claro uma característica do grupo de cor *Verde* que podemos compreender como uma classe que possui baixos índices de *Fixed Acidity* e de *Citric Acid*.

Figura 11 – Fixed Acidity e Citric Acid - 3 e 5 Centroides

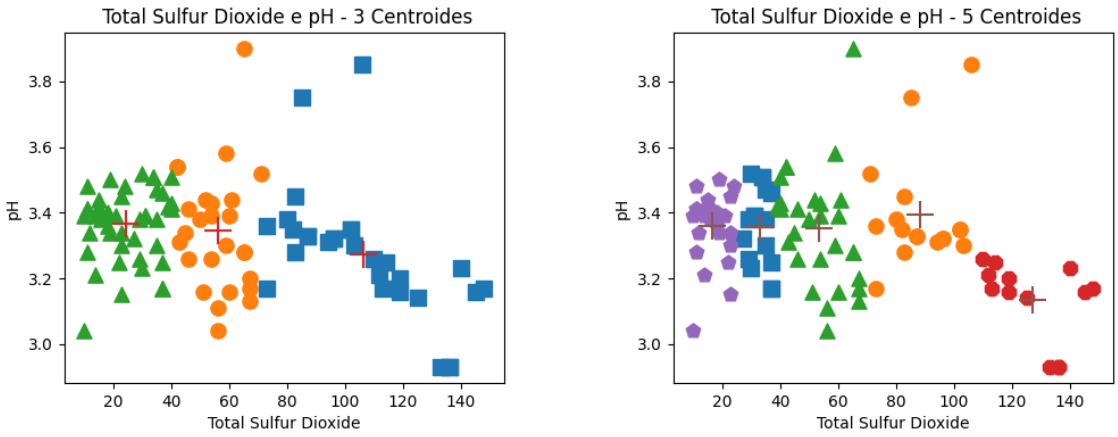


Fonte: Autor

Na Figura 12, podemos observar uma relação direta ao atributo *Total Sulfur Dioxide* quando aplicados diferentes centroides. Podemos concluir que a quantidade do atributo *pH* não é um fator de classificação, já que independente das medidas, o eixo *X - Total Sulfur Dioxide* acaba definindo as regras de similaridade. Dado o gráfico à direita exibindo os resultados com 5 centroides, podemos observar com clareza que as medidas de X se tornam mais estreitas para os três primeiros centroides, porém a margem de classificação para os outros restantes dobra comparado a eles. Se compararmos essa informação com o gráfico da esquerda que consta com 3 centroides, podemos criar a hipótese de que existam mais combinações de atributos que possam influenciar essa margem de $X = 80$ à $X = 150$.

Dados os resultados, o aprendizado de máquina não supervisionado *K-means* surpreende com seu simples entendimento de agrupamento e seu método de métrica, principalmente quando podemos observar a centralização dos centroides nas Figura 11 e Figura 12.

Figura 12 – Total Sulfur Dioxide e pH - 3 e 5 Centroides



Fonte: Autor

3 AVALIAÇÃO DO CRONOGRAMA

O cronograma desenvolvido para este projeto se dará pela Tabela 9 e Tabela 10.

3.1 PRIMEIRA ETAPA

Durante o decorrer do mês de Agosto, foi iniciado a pesquisa para as referências bibliográficas e a coleta de material para compor ideias para a aplicação final durante esta primeira etapa. Todos os autores estão citados neste presente relatório e já está inserido o resultado desta etapa se encontra no Capítulo 2.

De acordo com o cronograma inicial, a proposta seria estudar dois algoritmos de ML por mês e compartilhar os resultados obtidos com o grupo do projeto durante o decorrer da primeira etapa. Com o evento da FATEC Portas Abertas, colocamos como objetivo colocar uma aplicação de ML funcionando até a data de outubro e propor uma demonstração prática aos visitantes. Devido a esta nova etapa inserida no cronograma, decidimos inverter as metas dessa primeira fase e trabalhar a aplicação final para ser apresentada no evento.

3.1.1 Projeto de Esteganografia - FATEC Portas Abertas

No dia 27 de Outubro de 2019 ocorreu o evento FATEC - Portas Aberta, o qual trata-se de um evento anual que faz parte do programa A Universidade e as Profissões. Nesse dia, a Faculdade de Tecnologia de Americana "Ministro Ralph Biasi" abre suas portas com uma programação especial para que a comunidade conheça os cursos de graduação oferecidos pela unidade e o papel do tecnólogo, onde participam de palestras e visitas monitoradas às instalações da FATEC, como: salas de aula, laboratórios de ensino e pesquisa entre outros setores, como também projetos elaborados pelos discentes da instituição e que são demonstrados durante o evento.

Decidi por replicar o artigo de Berg et al. (2003), utilizando de um Raspberry PI 3 e uma Pi Camera, o projeto seria tirar fotos dos visitantes e esteganografa-las, demonstrando então a aplicação de detecção de esteganografia por IA. O Raspberry PI seria responsável por captar as fotos e processar a aplicação. Devido à complexidade da extração dos atributos da imagem esteganografada e do baixo poder de processamento do Raspberry PI, se tornou inviável a aplicação e partimos para uma versão simplificada do projeto.

Nesta versão simplificada, decidimos apenas capturar a foto dos visitantes, pedir para que eles formulassem uma frase para ser inserida e em seguida esteganografar a imagem, demonstrando a eficácia da técnica e a semelhança das duas imagens. Ao final do processo, a imagem esteganografada foi enviada aos visitantes como artifício para intensificar a imagem da FATEC Americana nas redes sociais. O algoritmo da aplicação para o evento se encontra no link: <https://gitlab.com/tuliocgomes/fatec_portas_abertas/>.

Após o evento, o foco foi em terminar o algoritmo de extração de atributos com base no artigo de Berg et al. (2003). Pela complexidade dos atributos, os meses de novembro a janeiro foram dedicados apenas na aplicação final. Os resultados da aplicação postergados para a segunda metade da iniciação científica, já que não haveria tempo hábil de documentar os resultados obtidos.

3.1.2 Itens da Primeira Etapa

1. Inicio do Projeto de Iniciação Científica.
2. Elaboração da Revisão Bibliográfica.
3. Estudo do algoritmo - kNN.
4. Projeto de Esteganografia - FATEC Portas Abertas.
5. Estudo do algoritmo - Naive Bayes.
6. Projeto final - Replica de um artigo de ML aplicado a Esteganálise.
7. Projeto final - Replicando a extração dos atributos necessários.
8. Projeto final - Dataset das imagens.
9. Projeto final - Treinamento e resultados do algoritmos.
10. Escrita do relatório parcial.
11. Entrega do relatório parcial.

Tabela 9 – Cronograma - 1° Etapa.

	2019					2020	
	AGO	SET	OUT	NOV	DEZ	JAN	FEV
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							

3.2 SEGUNDA ETAPA

Com a aplicação final pronta, os meses de abril a julho foram dedicados ao estudo os algoritmos propostos durante o início da iniciação científica. Todas as aplicações e resultados desse estudo podem ser visualizadas no Seção 2.2.

3.2.1 Itens da Segunda Etapa

1. Projeto final - Revisão da extração dos atributos.
2. Projeto final - Treinamento e resultados.
3. Estudo do algoritmo - Decision Tree.
4. Estudo do algoritmo - Regression.
5. Estudo do algoritmo - Linear Regression.
6. Estudo do algoritmo - Logistic Regression
7. Estudo do algoritmo - SVM
8. Estudo do algoritmo - K-Means
9. Escrita dos algoritmos
10. Escrita do relatório final.
11. Entrega do relatório final.

Tabela 10 – Cronograma - 2º Etapa.

	2020					2020	
	FEV	MAR	ABR	MAI	JUN	JULH	AGO
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							

4 Projeto Final

Com a decisão de antecipar a Etapa 2 da pesquisa, foi idealizado como aplicação final replicar o artigo *Searching For Hidden Messages Automatic Detection of Steganography* de Berg et al. (2003). O artigo se trata sobre treinar algoritmos de ML para reconhecer os padrões de um arquivo esteganografado. Anteriormente as análises de esteganografia eram visuais e realizando os ataques de forma manual, porém o seguinte artigo teve resultados positivos com imagens compactadas em comparação as técnicas anteriores.

Procurando por referência dos autores do artigo citado, não foi possível localizar repositórios compartilhados publicamente ou outras versões já replicadas deste mesmo artigo. Então decidimos elaborar a construção do mesmo e visualizar sua eficácia em esteganálise.

4.1 Extração dos Atributos

Por se tratar de uma classificação, o artigo utilizou de três algoritmos de ML: Naive Bayes, Decision Trees e Neural Networks. Como dataset, foram separadas 150 imagens em formato JPEG, divididas em: 50 flores, 50 montanhas e 50 Arvores. Por não existir um dataset público com tais especificações, utilizei de um script em python para realizar o download de 50 imagens de cada categoria utilizando o recurso do *Google Imagens*. É necessário que 25 imagens de cada grupo sejam imagens esteganografadas para o treinamento do algoritmo, então foi inserido um conjunto de caracteres de comprimento de 256 bits, ou seja, 32 bytes de caracteres alfabéticos denominado por *tokens* e utilizando o algoritmo de esteganografia *F5* para esteganografar as imagens. Em seu artigo, Berg et al. (2003) cita que inserir um curto trecho de caracteres só afirmaria a eficácia do algoritmo e que frases de maior comprimento por consequência se tornariam mais efetivas para serem detectadas. A Tabela 11 exibe as tokens utilizadas.

Tabela 11 – Dataset de caracteres inseridos via esteganografia.

	Caracteres Inseridos
1	vgtnghjnrofermwmbwhyiunttnqulfkg
2	gricuytvbzotppgseqlkdbqfffzkyjci
3	rfjtprssoghwmhufqpysxnukqaeixyso

Entre os requisitos para a extração dos atributos, é necessário dividir a imagem em blocos, no qual ele cita como uma matriz de 8x8. De acordo com o artigo, são necessários 51 atributos para o correto treinamento da aplicação. Sendo abaixo:

- Atributo 01 - Média da entropia para a imagem inteira.
- Atributo 02-03 - Média e Desvio Padrão para cada bloco da imagem.

- Atributo 04-11 - Média e Desvio Padrão para cada transição de probabilidades entre (0 $\rightarrow$ 0, 0 $\rightarrow$ 1, 1 $\rightarrow$ 0 e 1 $\rightarrow$ 1).
- Atributo 12-15 - A probabilidade média em toda a imagem para cada uma das transições (0 $\rightarrow$ 0, 0 $\rightarrow$ 1, 1 $\rightarrow$ 0 e 1 $\rightarrow$ 1).
- Atributo 16-51 - A Entropia condicional para posição que não pertença as bordas da imagem dentro do grid 8x8 DCT e calculado para a imagem inteira.

O processo de extração dos atributos teve a duração variável entre 1 segundo à 6 minutos para cada imagem, de acordo com o tamanho de cada imagem, sendo as esteganografadas e as não esteganografadas. Durante a extração, foi inserido o label *Stego*, sendo *1* para uma imagem esteganografada e *0* para uma imagem comum. A Tabela 12 exibe o cabeçalho do dataset final. O código utilizado para a extração dos atributos consta na Seção A.7.

Tabela 12 – Dataset de Imagens Esteganografadas e Não- Esteganografadas.

	Stego	EntropyImgFull	MeanEntropySlices	StdevEntropySlices	...
1	0	3.7749606609539716	5.917170446496514	4.017411166091254	...
2	0	5.509644819266748	9.341096180193254	0.1342231600545828	...
3	1	4.990205806260587	7.924566455571159	0.1947962837088572	...
4	1	4.306866090288476	4.941243399120087	1.8923277219898336	...
5	1	5.374798635879711	6.602803069637658	0.3219392853795799	...

Fonte: Autor

4.2 Particionando o Dataset

O artigo estudado não demonstra o motivo que levou a escolha dos atributos utilizados bem como os resultados de forma progressiva a fim de compor os 51 atributos citadas no estudo. A fim de explorar com mais detalhes, foi elaborado três datasets para testar a precisão de cada composição e comparar com o resultado final. Os três particionamentos são compostos da seguinte forma:

- 1º Dataset - Composto por três atributos: média da entropia para a imagem inteira, média e desvio padrão para cada bloco da imagem.
- 2º Dataset - Composto por 15 atributos, sendo então o 1º dataset somado à média e desvio padrão para cada transição de probabilidades e a probabilidade média em toda a imagem para cada uma das transições (0 $\rightarrow$ 0, 0 $\rightarrow$ 1, 1 $\rightarrow$ 0 e 1 $\rightarrow$ 1).
- 3º Dataset - Composto por 51 atributos: Dataset completo proposto pelo artigo.

Após a conclusão do dataset, foi preparado um ambiente de testes em uma nuvem pública, *Google Colab Research*, onde é possível utilizar os serviços de processamento de forma gratuita. Foi utilizado na aplicação o *Scikit Learn*, um framework opensource para ML, que utilizamos para treinar os algoritmos de Decision Trees, Navie Bayes e Perceptron substituindo Neural Network, a fim de explorar resultados diferentes do artigo proposto.

4.3 Resultados

Nessa seção discutiremos os resultados obtidos dos três datasets utilizados. A Tabela 13 possui os resultados obtidos no artigo de Berg et al. (2003) a fim de comparação. Todos os resultados foram obtidos utilizando 25% do dataset para a validação dos dados e os 75% restantes para o aprendizado. A precisão de cada algoritmo foi tirada de uma média simples a partir de seis amostras de cada resultado e observando as diferenças taxas de precisão de cada porção de instâncias para validação. É possível visualizar pelos resultados das tabelas que independente das partições de dataset utilizadas e suas quantidades de atributos explorados, sua maioria trouxe melhores resultados do que os apresentados no artigo de estudo.

Tabela 13 – Resultados obtidos no aprendizado de ataques utilizando a técnica de esteganografia Jsteg-Jpeg

	Decision Tree	Naive Bayes	Neural Network
Flower	78%	69%	81%
Mountain	54%	35%	76%
Tree	57%	33%	51%

Fonte: (BERG et al., 2003)

4.3.1 1º Dataset - 3 Atributos

Excluindo apenas os resultados do *Perceptron* aplicado as imagens *Mountain*, todos os outros algoritmos foram superiores aos resultados originais do artigo. Apesar da pequena quantidade de instâncias para treinamento e validação, as imagens do tipo *Flower* obtiveram uma alta taxa de precisão, sendo respectivamente 92% utilizando *Decision Tree* e 91% utilizando o *Perceptron*. Os demais resultados apesar de serem superiores, no mundo real não são suficientes para serem aplicados, já que a taxa de falso-positivos é alta dado os valores obtidos para gerar a média.

Tabela 14 – Resultados do 2º Dataset - 3 atributos e 50 instâncias

	Decision Tree	Naive Bayes	Perceptron
Flower	92%	76%	91%
Mountain	61%	46%	56%
Tree	69%	46%	74%

Fonte: Autor

4.3.2 2º Dataset - 15 Atributos

Todos os algoritmos retornaram altas taxas de precisão, exceto o *Naive Bayes* com o tipo de imagem *Mountain*, que apesar de compor 100% de precisão, tem fortes indícios

de que ocorreu um excesso de treinamento e comprometeu a precisão a ponto de considerar 100% eficaz. O ambiente ideal é aumentar a quantidade de instâncias e visualizar os diferentes comportamentos desse algoritmo em específico.

Tabela 15 – Resultados do 2º Dataset - 15 atributos e 50 instâncias

	Decision Tree	Naive Bayes	Perceptron
Flower	92%	76%	86%
Mountain	92%	100%	95%
Tree	92%	84%	86%

Fonte: Autor

4.3.3 3º Dataset - 51 Atributos

Apesar das grandes taxas de precisão apresentadas na tabela Tabela 16, os resultados obtidos nas imagens do tipo *Mountain* apresentaram em dois algoritmos taxa de 100% e 98% de precisão utilizando o *Perceptron* a partir do método da média descrito anteriormente. Esses dados nos revela que apesar de constar 100%, se observarmos o contexto dos datasets anteriores, se revela esses falsos-positivos ao conjunto de imagens do tipo *Mountain* e aumenta conforme maior o número de atributos. Pela pouca quantidade de instâncias para a validação dos dados, fica claro ser um erro de classificação e que não deve ser levado em consideração ao se utilizar em um ambiente no mundo real.

Tabela 16 – Resultados do 3º Dataset - 51 atributos e 50 instâncias

	Decision Tree	Naive Bayes	Perceptron
Flower	92%	76%	94%
Mountain	100%	100%	98%
Tree	84%	84%	95%

Fonte: Autor

4.3.4 Dataset completo - 51 Atributos e 150 instâncias

Como um complemente ao estudo elaborado por Berg et al. (2003), utilizei todo o ambiente já descrito anteriormente, mas realizando o aprendizado de máquina para os três algoritmos citados com todas as instâncias do dataset e com o particionamento dos atributos já mencionado. O diferencial desses resultados é observar as diferenças de precisão das tabelas Tabela 14,Tabela 15 e Tabela 16 a partir da quantidade de instâncias de treinamento e validação, bem como a influência dos atributos escolhidos. Os resultados se encontram na tabela Tabela 17.

Podemos observar que o dataset com três atributos e 150 instâncias já ultrapassou os resultados obtidos no artigo original, mas com uma variação de precisão bem inferior aos datasets particionados com 50 instâncias enquanto os outros dois conseguem uma alta taxa de precisão acima de 89%, sendo a maior utilizando o algoritmo *Decision Tree* com o dataset de 51 atributos e 150 instâncias. Esses resultados concluem que com uma maior quantidade de instâncias para treinamento e validação, a precisão tende a ser maior e mais homogênea entre os diferentes algoritmos de classificação, com uma taxa de variação de 3% entre eles.

Tabela 17 – Dataset completo - 51 atributos e 150 instâncias

features	Decision Tree	Naive Bayes	Perceptron
3	64%	57%	56%
15	90%	89%	92%
51	97%	89%	96%

Fonte: Autor

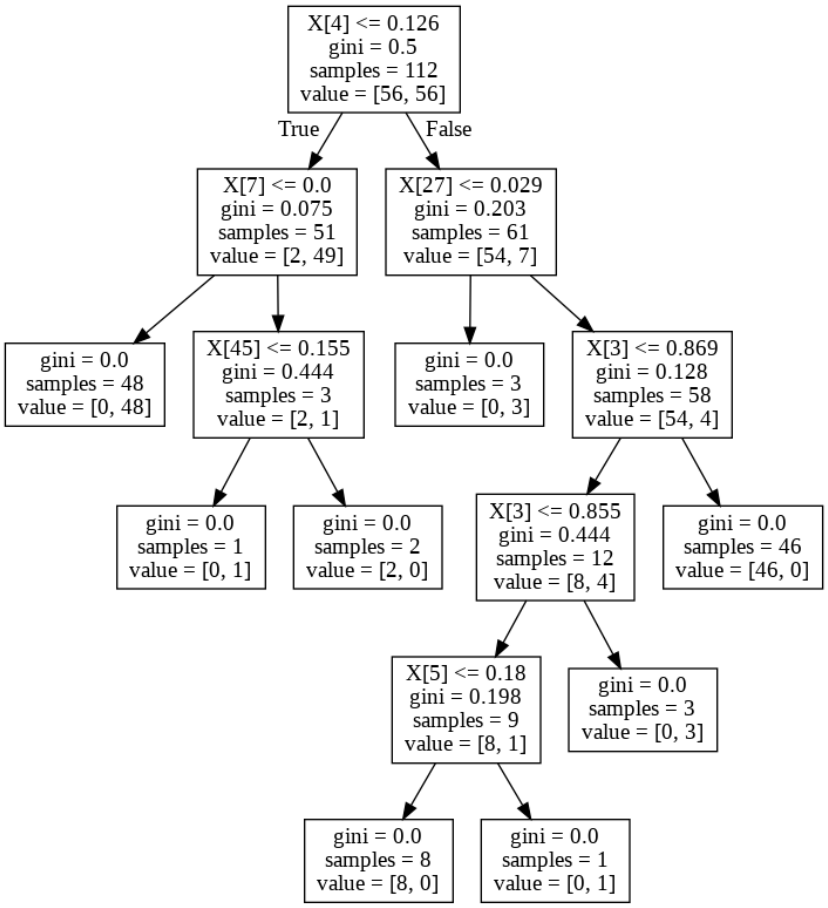
A fim de ter uma visualização mais clara do efeito do algoritmo *Decision Tree*, foi plotado a árvore de decisões para entendermos os nós internos presentes e qual os caminhos utilizados para esta classificação binária. A Figura 13 nos exibe que a média das transições das probabilidades $0 \rightarrow 0$ é o nó raiz e que o fluxograma apresenta 6 nós internos, resultando então em uma profundidamente de 6 saltos e com 8 folhas com os atributos para classificação.

4.3.5 Conclusão

Replicando o estudo de Berg et al. (2003) que utilizou o aprendizado de máquina para automaticamente construir ataques a esteganografia, podemos concluir que os atributos elaboradas pelo autor contribuíram positivamente para seus resultados em época e nos resultados contidos neste projeto. Apesar de não constar as conclusões para este tipo de mineração de dados nas imagens utilizadas, foi possível explorar a progressão dos atributos e seu comportamento a partir da quantidade de instâncias fornecidas no escopo do autor. O framework SKLearn contribuiu positivamente na aplicação e trouxe uma facilidade maior na aplicação assim como a versatilidade de selecionar os algoritmos com diversos parâmetros.

Os resultados foram acima do constatado no artigo do autor e reforçam que é possível utilizar ML para construir ataques de esteganografia baseados em algoritmos Jsteg-Jpeg e F5, já que ambos não possuem aplicação de criptografia. Como estudo futuro, elaborar o mesmo princípio do dataset apresentado no artigo, mas utilizando diferentes algoritmos de esteganografia mais modernos e com suporte a criptografia.

Figura 13 – Aplicação Final - Fluxograma - Decision Tree



Fonte: Autor

5 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Essa seção apresenta os resultados obtidos no do projeto da iniciação científica.

A Exploração dos algoritmos abordados na Seção 2.2 proporcionou uma outra visão e perspectiva da inteligência artificial, desde a construção dos algoritmos, o entendimento matemático e estatístico das aplicações, a implementação e principalmente entender a necessidade de cada modelo e aproveitando das suas vantagens e desvantagens. Por realizar os estudos práticos com datasets simples mas amplamente utilizados em estudos e artigos, foi possível visualizar com clareza a eficácia e a precisão do correto treinamento, preparação dos dados e o objetivo por trás de cada aplicação.

Durante o desenvolvimento da réplica do artigo citado na seção , foi necessário uma grande imersão em áreas como: Compressão de Imagem, células DCT, LSB, Transições de probabilidades com Markov Chains e Programação em Python, além dos algoritmos usados na exploração do problema. A réplica do artigo demandou demasiado material de apoio para o entendimento dos problemas apresentados e consequentemente compreender a necessidade de tais metodologias ou técnicas utilizas. Durante a reprodução, os resultados apresentados foram além do proposto pelo próprio autor e explorando de maneiras mais amplas comparadas ao artigo original.

Também é necessário ressaltar que a proposta do projeto para ser apresentado no FATEC Portas Abertas gerou uma dedicação e motivação, onde foi possível criar uma experiência semelhante a um projeto de cunho profissional utilizando-se de prazos e trabalho em equipe, já que era constante a troca de informações entre o grupo de orientados.

Os encontros presenciais e remotos com o orientador veio a ser uma adição excepcional, já que a troca de informações, discussão dos algoritmos em grupo e compartilhamento de experiências pessoais e profissionais veio a beneficiar durante todo o percurso. Deixo aqui meu agradecimento ao Prof. Me. Rossano Pablo Pinto pela oportunidade e por ter me auxiliar durante todas as etapas me sugerindo material.

Fica claro que, durante toda a fase da Iniciação Científica, foi possível reconhecer que a IA tem avançado grandes passos e tem muito a oferecer durante os próximos anos. É comum o termo ser constante em veículos de mídia ou em novos frameworks e modelos revolucionários de mercado, como *GPT-3*, mas em si requer um grande aprofundamento estatístico e matemático por parte do individuo a aplicar qualquer técnica ou modelo de IA. Durante o período de pesquisa, fui capaz de interagir com duas empresas AgroTech que tem utilizado massivamente frameworks como Sklearn, Keras, Pytorch e TensorFlow, porém conversando com os integrantes das equipes ficou claro eles não possuíam o mínimo de entendimento dos algoritmos de forma aprofundada, o que acabou influenciando em perdas significativas de precisão em alguns projetos por falta de otimizações ou de construções de modelos e algoritmos customizados para seus problemas. Em uma visita a uma dessas empresas e conversando com os integrantes dos grupos,

entendendo alguns conceitos simples explorados durante a etapa de extração de atributos do Seção 4.1, fiz a sugestão da adição de 6 atributos a mais para compor o dataset que estavam utilizando para o treinamento de um algoritmo de detecção de imagem, imediatamente a precisão apresentou uma melhora de 18% e sem apresentar overfitting. É gratificante ver este estudo científico proposto refletir de forma direta na indústria, com elogios e resultados positivos.

6 AVALIAÇÃO DO ORIENTADOR

Como mencionado no relatório parcial, o cronograma do trabalho realizado pelo orientado foi invertido depois de algumas semanas trabalhando no cronograma original. A proposta original tinha como parte final a elaboração de um protótipo. A alteração foi proposta por mim para que algo de concreto fosse apresentado no FATEC Portas Abertas.

Para a segunda metade da IC, a revisão da literatura foi complementada e está suficiente para uma IC.

O cronograma, alterado, foi concluído integralmente e é suficiente para aprovação nesse relatório final.

Destaco aqui que a capacidade crítica de análise, comparação de resultados e proposição de melhorias em trabalhos publicados, todos alcançados pelo Tulio, é o que se espera de um trabalho de iniciação científica. Além disso, o aluno relata a experiência de ter contribuído, com o que aprendeu, com uma empresa e os resultados alcançados pela empresa foram significativos.

O orientado utilizou ferramentas de ponta no desenvolvimento desse trabalho, como escrita em Latex utilizando Overleaf, publicação dos projetos no gitlab (um repositório git), Mendeley para organização das referências e nuvem Google Colab Research para executar os algoritmos de ML.

Sendo assim, considero o trabalho desenvolvido e este relatório final como suficientes para a aprovação da Iniciação Científica.

Referências

AHMEDMEDJAHED, S.; Ait Saadi, T.; BENYETTOU, A. Breast Cancer Diagnosis by using k-Nearest Neighbor with Different Distances and Classification Rules. **International Journal of Computer Applications**, v. 62, n. 1, p. 1–5, 2013. Citado na página 6.

ALSABTI, K.; RANKA, S.; SINGH, V. Electrical Engineering and Computer Science An efficient k-means clustering algorithm. **Electrical Engineering and Computer Science**, 1997. Citado na página 26.

ANDRÉS-FERRER, J.; JUAN, A. Constrained domain maximum likelihood estimation for naive Bayes text classification. **Pattern Analysis and Applications**, v. 13, n. 2, p. 189–196, 2010. ISSN 14337541. Citado na página 10.

BERG, G. et al. Searching for hidden messages: Automatic detection of steganography. **Computer Science Departament University at Albany**, p. 51–56, 2003. Citado 6 vezes nas páginas 33, 34, 36, 38, 39 e 40.

BRZEZINSKI, J. R.; KNAFL, G. J. Logistic regression modeling for context-based classification. **IEEE**, p. 755–759, 1999. Citado na página 17.

CENDROWSKA, J. PRISM: An algorithm for inducing modular rules. **International Journal of Man-Machine Studies**, v. 27, n. 4, p. 349–370, 1987. ISSN 00207373. Citado na página 15.

FISHER, R. A. The use of multiple measurements in taxonomic problems. **Annals of Eugenics**, n. 7, p. 179–188, 1936. Citado na página 7.

FUKUNAGA, K.; NARENDRA, P. M. A Branch and Bound Algorithm for Computing k-Nearest Neighbors. **IEEE Transactions on Computers**, C-24, n. 7, p. 750–753, 1975. ISSN 00189340. Citado na página 6.

HARRINGTON, P. **Machine Learning in Action**. [S.l.]: Manning Publications Co., 2012. ISBN 9781617290183. Citado 20 vezes nas páginas 1, 3, 7, 8, 9, 10, 11, 13, 15, 16, 17, 19, 20, 21, 22, 23, 25, 27, 28 e 30.

HARRISON, D.; RUBINFELD, D. L. Hedonic housing prices and the demand for clean air. **J. Environ. Economics Management, vol.5**, p. 81–102, 1979. Citado 2 vezes nas páginas 23 e 24.

KUHN, M.; JOHNSON, K. et al. **Applied Predictive Modeling**. [S.l.]: Springer, 2013. 615 p. ISBN 9781461468486. Citado 5 vezes nas páginas 17, 19, 21, 22 e 23.

MCLEISH, M.; CECILE, M.; LOPEZ-SUAREZ, A. Database issues for a veterinary medical expert system. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 339 LNCS, p. 177–192, 1989. ISSN 16113349. Citado 2 vezes nas páginas 19 e 20.

MOHAMMED, M.; KHAN, M. B.; BASHIE, E. B. M. **Machine learning: Algorithms and applications**. [S.l.]: Crc Press, 2016. v. 112. 112 p. ISBN 9781498705387. Citado 9 vezes nas páginas 3, 4, 5, 9, 10, 12, 14, 27 e 28.

MUKHERJEE, S.; SHARMA, N. Intrusion Detection using Naive Bayes Classifier with Feature Reduction. **Procedia Technology**, v. 4, p. 119–128, 2012. ISSN 22120173. Citado na página 9.

NILSSON, N. J. **Introduction to Machine Learning An Early Draft of A Proposed**. [S.l.]: Robotics Laboratory Department of Computer Science Stanford University, 1998. Citado 3 vezes nas páginas 3, 10 e 13.

NORVIG, P. R.; INTELLIGENCE, S. A. **A modern approach**. [S.l.]: Prentice Hall, 2009. Citado na página 1.

RAILEANU, L. E.; STOFFEL, K. Theoretical comparison between the Gini Index and Information Gain criteria. **Annals of Mathematics and Artificial Intelligence**, v. 41, n. 1, p. 77–93, 2004. ISSN 10122443. Citado na página 14.

SHALEV-SHWARTZ, S.; BEN-DAVID, S. **Understanding Machine Learning: From Theory to Algorithms**. [S.l.]: University Press, Cambridge, 2013. 1–397 p. ISBN 9781107057135. Citado 8 vezes nas páginas 4, 9, 12, 14, 18, 23, 27 e 28.

SHARIFZADEH, M. et al. Convolutional neural network steganalysis's application to steganography. **2017 IEEE Visual Communications and Image Processing, VCIP 2017**, v. 2018-Janua, p. 1–4, 2018. Citado na página 3.

SMOLA, A.; VISHWANATHAN, S. **Introduction to Machine Learning**. [S.l.]: University Press, Cambridge, 2008. 213 p. ISBN 0521825830. Citado 2 vezes nas páginas 3 e 5.

Apêndices

APÊNDICE A – Código dos Algoritmos Utilizados

Neste apêndice consta todos os códigos elaborados para o estudo dos algoritmos estudados na Seção 2.2 e também para a extração e treinamento abordados no Capítulo 5.

A.1 Código - kNN

```
# -*- coding: utf-8 -*-
from numpy import *
import numpy as np
import operator
import matplotlib.pyplot as plt
import pandas as pd

def calcDistance(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
    diffMat = tile(inX, (dataSetSize, 1)) - dataSet
    sqDiffMat = diffMat ** 2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances ** 0.5
    sortedDistIndices = distances.argsort()
    return sortedDistIndices


def findMajorityClass(inX, dataSet, labels, k, sortedDistIndices):
    classCount = {}
    for i in range(k):
        voteIlabel = labels[sortedDistIndices[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1
    return sorted(classCount.items(), key=operator.itemgetter(1),
                                        reverse=True)


def classify0(inX, dataSet, labels, k):
    sortedDistIndices = calcDistance(inX, dataSet, labels, k)
    sortedClassCount = findMajorityClass(inX, dataSet, labels, k,
                                        sortedDistIndices)
    return sortedClassCount[0][0]

def autoNorm(dataSet):
    minVals = dataSet.min(axis=0)
    maxVals = dataSet.max(axis=0)
    ranges = maxVals - minVals
    normDataSet = np.zeros(dataSet.shape)
    normDataSet = dataSet - minVals
```

```
    normDataSet = normDataSet / ranges
    return normDataSet, ranges, minVals

def file2matrix(filename):
    df = pd.read_csv(filename, header=0)
    x = df.drop(['species'],axis=1).values
    y = df['species'].values
    return x, y


# =================
errorCount = 0
neighbors = [1,3,5,7,9]

iris_data, iris_target = file2matrix('KNN/shuffle_iris.csv')
normMat, ranges, minVals = autoNorm(iris_data)
numTestVecs = int(normMat.shape[0] * 0.25)


for k in range(len(neighbors)):
    for i in range(numTestVecs):
      classifierResult = classify0(normMat[i], normMat[numTestVecs:],
                                   iris_target[numTestVecs:],
                                   neighbors[k])
      if classifierResult != iris_target[i]:
        errorCount += 1
    print('K neighbors:'+str(neighbors[k]))
    print("error rate: %f" % (errorCount / numTestVecs))
```

A.2 Código - Naive Bayes

```
# -*- coding: utf-8 -*-
import pickle
import numpy as np
import re

def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
    p1 = sum(vec2Classify * p1Vec) + np.log(pClass1)
    p0 = sum(vec2Classify * p0Vec) + np.log(1.0 - pClass1)
    if p1 > p0:
        return 1
    else:
        return 0

def createVocabList(dataSet):
    vocabSet = set([])
```

```
    for document in dataSet:
        vocabSet = vocabSet | set(document)
    return list(vocabSet)

def trainNB0(trainMatrix, trainCategory):
    numTrainDocs = len(trainMatrix)
    numWords = len(trainMatrix[0])
    pAbusive = sum(trainCategory) / numTrainDocs
    p0Num, p1Num = np.ones(numWords), np.ones(numWords)
    p0Denom, p1Denom = 2.0, 2.0
    for i in range(numTrainDocs):
        if trainCategory[i] == 1:
            p1Num += trainMatrix[i]
            p1Denom += sum(trainMatrix[i])
        else:
            p0Num += trainMatrix[i]
            p0Denom += sum(trainMatrix[i])
    p1Vect = np.log(p1Num / p1Denom)
    p0Vect = np.log(p0Num / p0Denom)
    return p0Vect, p1Vect, pAbusive

def bagOfWords2VecMN(vocabList, inputSet):
    returnVec = [0] * len(vocabList)
    for word in inputSet:
        if word in vocabList:
            returnVec[vocabList.index(word)] += 1
    return returnVec

def remove_emoji(string):
    emoji_pattern = re.compile("["
                       u"\U0001F600-\U0001F64F"
                       u"\U0001F300-\U0001F5FF"
                       u"\U0001F680-\U0001F6FF"
                       u"\U0001F1E0-\U0001F1FF"
                       u"\U00002702-\U000027B0"
                       u"\U000024C2-\U0001F251"
                       "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', string)

def textParse(bigString):
    import re
    listOfTokens = remove_emoji(bigString)
    listOfTokens = re.sub(r'\bVirginAmerica\b\s+',"",listOfTokens)
    listOfTokens = re.sub(r'^https?:\/\/.*[\r\n]*', '', listOfTokens,
                                        flags=re.MULTILINE)
    listOfTokens = re.split(r'\W+', listOfTokens)
    return [tok.lower() for tok in listOfTokens if len(tok) > 1]
```

```
# ====================
def testingNB():
    docList, classList = [], []
    for i in range(1, 101):
        wordList = textParse(open('./Naive_Bayes/tweets/tweets_negative/
                                                %d.txt' % i,
                                  errors='ignore').read())
        docList.append(wordList)
        classList.append(1)
        wordList = textParse(open('./Naive_Bayes/tweets/tweets_positive/
                                                %d.txt' % i,
                                  errors='ignore').read())
        docList.append(wordList)
        classList.append(0)
    vocabList = createVocabList(docList)
    trainingSet, testSet = list(range(200)), []
    for i in range(75):
        randIndex = int(np.random.uniform(0, len(trainingSet)))
        testSet.append(trainingSet[randIndex])
        del(trainingSet[randIndex])
    trainMat, trainClasses = [], []
    for docIndex in trainingSet:
        trainMat.append(bagOfWords2VecMN(vocabList, docList[docIndex]))
        trainClasses.append(classList[docIndex])
    p0V, p1V, pSpam = trainNB0(np.array(trainMat), np.array(trainClasses
                                            ))
    errorCount = 0
    for docIndex in testSet:
        wordVector = bagOfWords2VecMN(vocabList, docList[docIndex])
        if classifyNB(np.array(wordVector),
                      p0V, p1V, pSpam) != classList[docIndex]:
            errorCount += 1
            print("classification error", docList[docIndex])
    print('the error rate is: ', errorCount / len(testSet))

testingNB()
```

A.3 Código - Decision Trees

A.3.1 Tree

```
# -*- coding: utf-8 -*-
import operator

def calcShannonEnt(dataSet):
```

```
    from math import log
    numEntries = len(dataSet)
    labelCounts = {}
    for featVec in dataSet:
        currentLabel = featVec[-1]
        if currentLabel not in labelCounts:
            labelCounts[currentLabel] = 1
        else:
            labelCounts[currentLabel] += 1
    shannonEnt = 0.0
    for key in labelCounts:
        prob = labelCounts[key] / numEntries
        shannonEnt -= prob * log(prob, 2)
    return shannonEnt

def splitDataSet(dataSet, axis, value):
    retDataSet = []
    for featVec in dataSet:
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]
            reducedFeatVec.extend(featVec[axis + 1:])
            retDataSet.append(reducedFeatVec)
    return retDataSet

def chooseBestFeatureToSplit(dataSet):
    numEntries = len(dataSet)
    numFeatures = len(dataSet[0]) - 1
    baseEntropy = calcShannonEnt(dataSet)
    bestInfoGain, bestFeature = 0, -1
    for i in range(numFeatures):
        uniqueVals = set([example[i] for example in dataSet])
        newEntropy = 0.0
        for value in uniqueVals:
            subDataSet = splitDataSet(dataSet, i, value)
            prob = len(subDataSet) / numEntries
            newEntropy += prob * calcShannonEnt(subDataSet)
        infoGain = baseEntropy - newEntropy
        if infoGain > bestInfoGain:
            bestInfoGain = infoGain
            bestFeature = i
    return bestFeature

def majorityCnt(classList):
    classCount = {}
    for vote in classList:
        if vote not in classCount:
            classCount[vote] = 1
```

```
        else:
            classCount[vote] += 1
    sortedClassCount = sorted(classCount.items(), key=lambda x: x[1],
                              reverse=True)
    return sortedClassCount[0][0]


def createTree(dataSet, labels):
    classList = [example[-1] for example in dataSet]
    if classList.count(classList[0]) == len(classList):
        return classList[0]
    if len(dataSet[0]) == 1:
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)
    bestFeatLabel = labels[bestFeat]
    myTree = {bestFeatLabel: {}}
    del(labels[bestFeat])
    featValues = [example[bestFeat] for example in dataSet]
    for value in set(featValues):
        subLabels = labels[:]
        myTree[bestFeatLabel][value] = createTree(
            splitDataSet(dataSet, bestFeat, value), subLabels)
    return myTree


def classify(inputTree, featLabels, testVec):
    firstStr = list(inputTree.keys())[0]
    secondDict = inputTree[firstStr]
    featIndex = featLabels.index(firstStr)
    key = testVec[featIndex]
    valueOfFeat = secondDict[key]
    if isinstance(valueOfFeat, dict):
        classLabel = classify(valueOfFeat, featLabels, testVec)
    else:
        classLabel = valueOfFeat
    return classLabel


def storeTree(inputTree, filename):
    import pickle
    fw = open(filename, "wb")
    pickle.dump(inputTree, fw)
    fw.close()


def grabTree(filename):
    import pickle
```

```
    fr = open(filename, "rb")
    return pickle.load(fr)
```

A.3.2 Tree Plotter

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt


decisionNode = dict(boxstyle="sawtooth", fc="0.8")
leafNode = dict(boxstyle="round4", fc="0.8")
arrow_args = dict(arrowstyle="<-")



def plotNode(nodeTxt, centerPt, parentPt, nodeType, ax):
    ax.annotate(nodeTxt, xy=parentPt, xycoords='axes fraction',
                xytext=centerPt, textcoords='axes fraction',
                va="center", ha="center", bbox=nodeType,
                arrowprops=arrow_args)



def plotMidText(cntrPt, parentPt, txtString, ax):
    xMid = (parentPt[0] - cntrPt[0]) / 2.0 + cntrPt[0]
    yMid = (parentPt[1] - cntrPt[1]) / 2.0 + cntrPt[1]
    ax.text(xMid, yMid, txtString, va="center", ha="center",
            rotation=30)


def plotTree(myTree, parentPt, nodeTxt, totalW, totalD, xyOff, ax):
    numLeafs = getNumLeafs(myTree)
    firstStr = list(myTree.keys())[0]
    cntrPt = (xyOff[0] + (1 + numLeafs) / (2 * totalW), xyOff[1])
    plotMidText(cntrPt, parentPt, nodeTxt, ax)
    plotNode(firstStr, cntrPt, parentPt, decisionNode, ax)
    secondDict = myTree[firstStr]
    xyOff[1] -= 1 / totalD
    for key in sorted(secondDict.keys()):
        if isinstance(secondDict[key], dict):
            plotTree(secondDict[key], cntrPt, str(key),
                     totalW, totalD, xyOff, ax)
        else:
            xyOff[0] += 1.0 / totalW
            plotNode(secondDict[key], (xyOff[0], xyOff[1]), cntrPt,
                     leafNode, ax)
            plotMidText((xyOff[0], xyOff[1]), cntrPt, str(key), ax)
```

```
    xyOff[1] += 1 / totalD


def createPlot(inTree, figsize):
    fig, ax = plt.subplots(subplot_kw={'xticks':[], 'yticks':[],
                                       'frameon':False},
                           figsize=figsize)
    totalW = getNumLeafs(inTree)
    totalD = getTreeDepth(inTree)
    xyOff = [-0.5 / totalW, 1]
    plotTree(inTree, (0.5, 1.0), '', totalW, totalD, xyOff, ax)
    plt.show()


def getNumLeafs(myTree):
    numLeafs = 0
    firstStr = list(myTree.keys())[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        # Test if node is dictionary
        if isinstance(secondDict[key], dict):
            numLeafs += getNumLeafs(secondDict[key])
        else:
            numLeafs += 1
    return numLeafs


def getTreeDepth(myTree):
    maxDepth = 0
    firstStr = list(myTree.keys())[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        if isinstance(secondDict[key], dict):
            thisDepth = 1 + getTreeDepth(secondDict[key])
        else:
            thisDepth = 1
        if thisDepth > maxDepth:
            maxDepth = thisDepth
    return maxDepth
```

A.3.3 Call Tree

```
# -*- coding: utf-8 -*-
import tree
import tree_plotter
import numpy
```

```
# ====
figsize = (10,7)
# ===

fr = open('Decision_Tree/lenses.txt')
lenses_data = [inst.strip().split('\t') for inst in fr.readlines()]
lenses_label = ['age', 'prescript', 'astigmatic', 'tearRate']
lenses_tree = tree.createTree(lenses_data, lenses_label)

tree.storeTree(lenses_tree,'Decision_Tree/lenses_classifier.txt')


tree_plotter.createPlot(lenses_tree,figsize)

#print(tree.classify(dict_mytree,labels,[1,1]))
```

A.4 Código - Regressão Logística

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(inX):
    return 1 / (1 + np.exp(-inX))

def stocGradAscent1(dataMatrix, classLabels, numIter=150):
    m, n = dataMatrix.shape
    weights = np.ones(n)
    for j in range(numIter):
        dataIndex = list(range(m))
        for i in range(m):
            alpha = 4 / (1.0 + j + i) + 0.01
            randIndex = int(np.random.uniform(0, len(dataIndex)))
            h = sigmoid(np.dot(dataMatrix[dataIndex[randIndex]], weights
                                                    ))
            error = classLabels[dataIndex[randIndex]] - h
            weights = (weights +
                       alpha * error * dataMatrix[dataIndex[randIndex]])
            del(dataIndex[randIndex])
    return weights

def classifyVector(inX, weights):
    prob = sigmoid(np.dot(inX, weights))
    if prob > 0.5:
        return 1
```

```
    else:
        return 0


def colicTest():
    frTrain = open('./logistic_regression/horseColicTraining.txt')
    frTest = open('./logistic_regression/horseColicTest.txt')
    trainingSet, trainingLabels = [], []
    for line in frTrain.readlines():
        currLine = line.strip().split('\t')
        lineArr = []
        for i in range(21):
            lineArr.append(float(currLine[i]))
        trainingSet.append(lineArr)
        trainingLabels.append(float(currLine[21]))
    trainWeights = stocGradAscent1(np.array(trainingSet),
                                   np.array(trainingLabels), 1000)
    errorCount, numTestVec = 0, 0
    for line in frTest.readlines():
        numTestVec += 1
        currLine = line.strip().split('\t')
        lineArr = []
        for i in range(21):
            lineArr.append(float(currLine[i]))
        if int(classifyVector(np.array(lineArr),
                              trainWeights)) != int(currLine[21]):
            errorCount += 1
    errorRate = errorCount / numTestVec
    print("the error rate of this test is: %f" % errorRate)
    return errorRate


def multiTest():
    numTests = 10
    errorSum = 0
    for k in range(numTests):
        errorSum += colicTest()
    print("after %d iterations the average error rate is: %f"
          % (numTests, errorSum / numTests))

multiTest()
```

A.5 Código - Regressão Linear

```
# -*- coding: utf-8 -*-
import re
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd


def standRegres(xArr, yArr):
    xTx = np.dot(xArr.T, xArr)
    if np.linalg.det(xTx) == 0.0:
        print("This matrix is singular, cannot do inverse")
        return
    ws = np.dot(np.linalg.inv(xTx),
                np.dot(xArr.T, yArr[:, np.newaxis]))
    return ws.ravel()


def lwlr(testPoint, xArr, yArr, k=1.0):
    m = xArr.shape[0]
    weights = np.eye(m)
    for j in range(m):
        diffMat = testPoint - xArr[j]
        weights[j, j] = np.exp(np.dot(diffMat, diffMat) / (-2 * k ** 2))
    xTx = np.dot(xArr.T, np.dot(weights, xArr))
    if np.linalg.det(xTx) == 0.0:
        print("This matrix is singular, cannot do inverse")
        return
    ws = np.dot(np.linalg.inv(xTx),
                np.dot(xArr.T, np.dot(weights, yArr[:, np.newaxis])))
    return np.dot(testPoint, ws.ravel())


def lwlrTest(testArr, xArr, yArr, k=1.0):
    m = testArr.shape[0]
    yHat = np.zeros(m)
    for i in range(m):
        yHat[i] = lwlr(testArr[i], xArr, yArr, k)
    return yHat

def rssError(yArr, yHatArr):
    return ((yArr - yHatArr) ** 2).sum()



def ridgeRegres(xArr, yArr, lam=0.2):
    xTx = np.dot(xArr.T, xArr)
    denom = xTx + np.eye(xArr.shape[1]) * lam
    if np.linalg.det(denom) == 0.0:
        print("This matrix is singular, cannot do inverse")
```

```
        return
    ws = np.dot(np.linalg.inv(denom),
                np.dot(xArr.T, yArr[:, np.newaxis]))
    return ws.ravel()


def ridgeTest(xArr, yArr):
    yMean = np.mean(yArr)
    yArr = yArr - yMean
    xMeans = np.mean(xArr, axis=0)
    xStd = np.std(xArr, axis=0)
    xArr = (xArr - xMeans) / xStd
    numTestPts = 30
    wMat = np.zeros((numTestPts, xArr.shape[1]))
    for i in range(numTestPts):
        ws = ridgeRegres(xArr, yArr, np.exp(i - 10))
        wMat[i] = ws
    return wMat


def crossValidation(xArr, yArr, numVal=10):
    m = len(yArr)
    indexList = list(range(m))
    errorMat = np.zeros((numVal, 30))
    for i in range(numVal):
        trainX, trainY = [], []
        testX, testY = [], []
        np.random.shuffle(indexList)
        for j in range(m):
            if j < m * 0.9:
                trainX.append(xArr[indexList[j]])
                trainY.append(yArr[indexList[j]])
            else:
                testX.append(xArr[indexList[j]])
                testY.append(yArr[indexList[j]])
        trainX, trainY = np.array(trainX), np.array(trainY)
        testX, testY = np.array(testX), np.array(testY)
        wMat = ridgeTest(trainX, trainY)
        meanTrain = np.mean(trainX, axis=0)
        stdTrain = np.std(trainX, axis=0)
        testX = (testX - meanTrain) / stdTrain
        for k in range(30):
            yEst = np.dot(testX, wMat[k]) + np.mean(trainY)
            errorMat[i, k] = rssError(yEst, testY)
    meanErrors = np.mean(errorMat, axis=0)
    bestWeights = wMat[np.argmin(meanErrors)]
    meanX = np.mean(xArr, axis=0)
```

```
    stdX = np.std(xArr, axis=0)
    unReg = bestWeights / stdX
    print("the best model from Ridge Regression is:\n", unReg)
    print("with constant term: ",
          -1 * np.dot(meanX, unReg) + np.mean(yArr))

lgX, lgY = [], []
setDataCollect(lgX, lgY)
lgX, lgY = np.array(lgX), np.array(lgY)
lgX1 = np.ones((82, 4))
lgX1[:, 1: 4] = lgX
ws = standRegres(lgX1, lgY)
print(ws)

print(crossValidation(lgX, lgY, 10))
```

A.6 Código - K-Means

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.datasets import load_wine
import pandas as pd

def distSLC(vecA, vecB):
    a = np.sin(vecA[1] * np.pi / 180) * np.sin(vecB[1] * np.pi / 180)
    b = (np.cos(vecA[1] * np.pi / 180) * np.cos(vecB[1] * np.pi / 180) *
         np.cos(np.pi * (vecB[0] - vecA[0]) / 180))
    return np.arccos(a + b) * 6371.0

def distEclud(vecA, vecB):
    return np.sqrt(np.sum(np.power(vecA - vecB, 2)))

def randCent(dataSet, k):
    n = dataSet.shape[1]
    centroids = np.zeros((k, n))
    for j in range(n):
        minJ = min(dataSet[:, j])
        rangeJ = max(dataSet[:, j]) - minJ
        centroids[:, j] = minJ + rangeJ * np.random.rand(k)
    return centroids

def kMeans(dataSet, k, distMeas=distEclud, createCent=randCent):
    m = dataSet.shape[0]
    clusterAssment = np.zeros((m, 2))
```

```
    centroids = createCent(dataSet, k)
    clusterChanged = True
    while clusterChanged:
        clusterChanged = False
        for i in range(m):
            minDist = np.inf
            minIndex = -1
            for j in range(k):
                distJI = distMeas(centroids[j], dataSet[i])
                if distJI < minDist:
                    minDist = distJI
                    minIndex = j
            if clusterAssment[i, 0] != minIndex:
                clusterChanged = True
            clusterAssment[i] = [minIndex, minDist ** 2]
        for cent in range(k):
            ptsInClust = dataSet[np.nonzero(clusterAssment[:, 0] == cent
                                            )[0]]
            centroids[cent] = np.mean(ptsInClust, axis=0)
    return centroids, clusterAssment

def biKmeans(dataSet, k, distMeas=distEclud):
    m = dataSet.shape[0]
    clusterAssment = np.zeros((m, 2))
    centList = [np.mean(dataSet, axis=0)]
    for j in range(m):
        clusterAssment[j, 1] = distMeas(centList[0], dataSet[j]) ** 2
    while len(centList) < k:
        lowestSSE = np.inf
        for i in range(len(centList)):
            ptsInCurrCluster = dataSet[np.nonzero(clusterAssment[:, 0] =
                                                  = i)[0]]
            centroidMat, splitClustAss = kMeans(ptsInCurrCluster, 2,
                                                distMeas)
            sseSplit = sum(splitClustAss[:, 1])
            sseNotSplit = sum(
                clusterAssment[np.nonzero(clusterAssment[:, 0] != i)[0],
                                          1])
            if sseSplit + sseNotSplit < lowestSSE:
                bestCentToSplit = i
                bestNewCents = centroidMat
                bestClustAss = splitClustAss.copy()
                lowestSSE = sseSplit + sseNotSplit
        bestClustAss[
            np.nonzero(bestClustAss[:, 0] == 1)[0], 0] = len(centList)
        bestClustAss[
            np.nonzero(bestClustAss[:, 0] == 0)[0], 0] = bestCentToSplit
```

```
        centList[bestCentToSplit] = bestNewCents[0]
        centList.append(bestNewCents[1])
        clusterAssment[np.nonzero(
            clusterAssment[:, 0] == bestCentToSplit)[0]] = bestClustAss
    return np.array(centList), clusterAssment


def cluster(numClust=3):
    df = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-
                                        databases/wine-quality/
                                        winequality-red.csv',sep=';') #
                                        header=0
    print(df)
    df = df[:100]

    fixed_acidity__citric_acid = df[['fixed acidity','citric acid']].
                                        values
    fixed_acidity__density = df[['fixed acidity','density']].values
    total_sulfur_dioxide__ph= df[['total sulfur dioxide','pH']].values
    fixed_acidity__ph= df[['fixed acidity','pH']].values

    datMat = total_sulfur_dioxide__ph

    myCentroids, clustAssing = biKmeans(datMat, numClust, distMeas=
                                        distSLC) #distEclud e distSLC
    scatterMarkers = ['s', 'o', '^', '8', 'p', 'd', 'v', 'h', '>', '<']
    plt.figure(figsize=(5, 4))
    plt.xlabel("Total Sulfur Dioxide")
    plt.ylabel("pH")
    plt.title(" Total Sulfur Dioxide e pH - 3 Centroides")
    for i in range(numClust):
        ptsInCurrCluster = datMat[np.nonzero(clustAssing[:, 0] == i)[0]]
        markerStyle = scatterMarkers[i % len(scatterMarkers)]
        plt.scatter(ptsInCurrCluster[:, 0], ptsInCurrCluster[:, 1],
                    marker=markerStyle, s=90)
    plt.scatter(myCentroids[:, 0], myCentroids[:, 1],
                marker='+', s=300)


    plt.show()

cluster(3)
```

A.7 Projeto Final - Extração dos Atributos

```
# -*- coding: utf-8 -*-
```

```
from skimage import io, color, img_as_ubyte
from sklearn.metrics.cluster import entropy
import os, sys
import numpy as np
from PIL import Image
import statistics
import pandas as pd
import datetime as dt
import image_slicer

# Argumento que passa o nome da imagem
argv_img = sys.argv[1]
# argv_img = "./ml_app_final/test2.jpg"


# Variaveis de features
dir_64_imgs = './64_slices'
bw, bh = 8, 8 # tamanho bloco (altura e largura)
stego = "yes"

def identificar_os():
    # Variaveis
    windows_unidade_padrao = 'c'
    diretorio_base = ''

    # Check se    windows (nt) ou linux/osx
    if os.name == 'nt':
        diretorio_base = windows_unidade_padrao + ':'
    else:
        diretorio_base = os.getenv("HOME")
    return diretorio_base

def cria_diretorio():
    if os.path.isdir(dir_64_imgs):
        pass
    else:
       os.mkdir(dir_64_imgs)
    return dir_64_imgs

def img_full_to_gray(path_img):
    rgbImg = io.imread(path_img)
    r_gray_img = img_as_ubyte(color.rgb2gray(rgbImg))
    return r_gray_img

def slice_img(path_img):
    # Variaveis
```

```
    qtd_slices = 64 #8x8
    dir = cria_diretorio()

    # Dividir a imagem em 64 partes
    tiles = image_slicer.slice(path_img, qtd_slices, save=False)
    image_slicer.save_tiles(tiles, directory=dir,format='JPEG')

def open_64_imgs():
    from os import listdir
    from os.path import isfile, join

    # Gerar uma lista com todas as imagens do dir 64_slices
    all_imgs = [f for f in listdir(dir_64_imgs) if isfile(join(
                                        dir_64_imgs, f))]
    all_imgs2 = all_imgs
    # Tive q criar uma copia pq a funcao extrai_36_slices estava
                                        sobreescrevendo all_imgs para 36

    # Ordem Alfabetica
    all_imgs = sorted(all_imgs)
    all_imgs2 = sorted(all_imgs2)

    # Remover as "bordas" da Imagem para a ultima feature
    list_36_slices = extrai_36_slices(all_imgs)


    #transformar as imagens em grayscale
    r_list_36img_gray = img_to_gray(list_36_slices)
    r_list_64img_gray = img_to_gray(all_imgs2)

    return r_list_36img_gray, r_list_64img_gray

def img_to_gray(path_img):
    from skimage.io import imread
    # Variaveis
    list_gray_img = []

    for i in range(len(path_img)):
        gray_img = imread(dir_64_imgs + os.sep + path_img[i], as_gray=
                                            True)
        # rgbImg = io.imread(dir_64_imgs + os.sep + path_img[i])
        # gray_img = img_as_ubyte(color.rgb2gray(rgbImg))
        list_gray_img.append(gray_img)

    return list_gray_img

def extrai_36_slices(list_all_64):
```

```
    # A imagem composta de 8x8, retiraremos as seguintes posicoes.
    #
    #     0   1 2 3 4 5 6 7
    #     8   * * * * * * 15
    #    16   * * * * * * 23
    #    24   * * * * * * 31
    #    32   * * * * * * 39
    #    40   * * * * * * 47
    #    48   * * * * * * 55
    #    56   57   58   59   60   61   62   63

    # Variaveis
    # Possui as posi  es em index que devem ser removidas
    imgs_index = [0,1,2,3,4,5,6,7,8,15,16,23,24,31,32,39,40,47,48,55,56,
                                            57,58,59,60,61,62,63]
    for i in reversed(imgs_index):
        del list_all_64[i]

    return list_all_64

def deleta_dir_64_slices(dir):
    import shutil
    # Remove diretorios com arquivos
    shutil.rmtree(dir)

def extract_entropy_full(path_img):
    ### FEATURE 1 ###
    # Extraindo a entropia da imagem inteira
    rgbImg = io.imread(path_img)
    grayImg = img_as_ubyte(color.rgb2gray(rgbImg))
    return_entropia = entropy(grayImg)
    return return_entropia

def entropia_por_bloco(list_blocos_img):
    # String de Retorno
    return_lista_entropia_img = []

    for i in range(len(list_blocos_img)):
            return_lista_entropia_img.append(entropy(list_blocos_img[i])
                                                    )
    return return_lista_entropia_img

def media_entropia_blocos(entropia_blocos):
    ### FEATURE 2 ###
    return_media_entropia = statistics.mean(entropia_blocos)
    return return_media_entropia
```

```
def stdev_entropia_blocos(entropia_blocos):
    ### FEATURE 3 ###
    return_stdev_entropia = statistics.stdev(entropia_blocos)
    return return_stdev_entropia

def regex(list_64):
    import re

    # Variaveis
    r_lista_regex = []

    # Remover todos os caracteres que nao forem 0 e 1
    for i in range(len(list_64)):
        # Regex sub substitui em "" oq nao for 0 ou 1 em casas unicas
        regex = re.sub('0*([1-9]\d+|[2-9])','',str(list_64[i]))
        # Regex findall extrai apenas os numeros
        regex2 = re.findall('\d+', str(regex))
        r_lista_regex.append(regex2)

    return r_lista_regex

def markov_chains(lista_img):
    # Variaveis
    chain_list = []
    matrix_2x2_minima = ['0','0','1','1']
    return_markov_chain = []

    # Transformar em lista unidimesional
    for i in range(len(lista_img)):
        chain_list.append(np.concatenate(lista_img[i]).ravel())

    # Regex
    lista_regex = regex(chain_list)

    # Adicionar 2 vezes 0 e 1 para formar a tabela 2x2
    for i in range(len(lista_regex)):
        for j in range(len(matrix_2x2_minima)):
            lista_regex[i].append(matrix_2x2_minima[j])

        # Aplicar a markov chains para cada posi  o
        mk = lista_regex[i]
        # A probabilidade (forward) do Estado Atual (0) para se Tornar
                                            um proximo
        #   Estado (1) pode ser encontrado na coluna '1' e linha '0' (0.
                                            5).
        #
```

```
        # Caso queira as probabilidades anteriores (backward), apenas
                                            sete o
        #       normalize = 1.
        #
        # === Exemplo do DATAFRAME para referencia da explicao acima ===
        #         Proximo    0    1
        #         Atual
        #         0         0.7  0.5
        #         1         0.0  1.0
        x = pd.crosstab(pd.Series(mk[:-1], name='Atual'),pd.Series(mk[1:
                                            ],name='Proximo'),normalize=
                                            0)
        return_markov_chain.append(x)

    # Retornar markov_chains em dataframe (em array pode-se utilizar .
                                        values ao final)
    return return_markov_chain

def isolar_variaveis_matrix_markov_chains(lista_dataframe_mk):
    # Variaveis para append
    return_mk_00 = []
    return_mk_01 = []
    return_mk_10 = []
    return_mk_11 = []

    for i in range(len(lista_dataframe_mk)):
        x = 0
        # A cada linha eu fa o o append para a lista de variaveis.
        for index, row in lista_dataframe_mk[i].iterrows():
            # Como    uma matrix fixa de 2x2, preferi usar um contador
                                                simples
            #       para a escolha de qual lista ser  usada para o
                                                append.
            if x == 0:
                    return_mk_00.append(row['0'])
                    return_mk_01.append(row['1'])
            else:
                    return_mk_10.append(row['0'])
                    return_mk_11.append(row['1'])
            x = x+1

    return return_mk_00, return_mk_01, return_mk_10, return_mk_11

def media_matrix_markov(input_mk_00,input_mk_01,input_mk_10,input_mk_11)
                                        :
    ### FEATURE 4-7 ###
    return_media_00 = statistics.mean(input_mk_00)
```

```
    return_media_01 = statistics.mean(input_mk_01)
    return_media_10 = statistics.mean(input_mk_10)
    return_media_11 = statistics.mean(input_mk_11)

    return return_media_00, return_media_01, return_media_10,
                                        return_media_11

def stdev_matrix_markov(input_mk_00,input_mk_01,input_mk_10,input_mk_11)
                                        :
    ### FEATURE 8-11 ###
    return_stdev_00 = statistics.stdev(input_mk_00)
    return_stdev_01 = statistics.stdev(input_mk_01)
    return_stdev_10 = statistics.stdev(input_mk_10)
    return_stdev_11 = statistics.stdev(input_mk_11)

    return return_stdev_00, return_stdev_01, return_stdev_10,
                                        return_stdev_11

def media_full_img(path_img):
    # Variaveis
    markov_chain_full = []
    matrix_2x2_minima = ['0','0','1','1']

    # Transformar a img em gray e aplicar o regex
    gray_img = img_full_to_gray(path_img)
    lista_regex = regex(gray_img)

    # Adicionar 2 vezes 0 e 1 para formar a tabela 2x2
    for i in range(len(lista_regex)):
        for j in range(len(matrix_2x2_minima)):
            lista_regex[i].append(matrix_2x2_minima[j])

        # Aplicar a markov chains para cada posi   o
        mk = lista_regex[i]
        # A probabilidade (forward) do Estado Atual (0) para se Tornar
                                            um proximo
        #   Estado (1) pode ser encontrado na coluna '1' e linha '0' (0.
                                            5).
        #
        # Caso queira as probabilidades anteriores (backward), apenas
                                            sete o
        #       normalize = 1.
        #
        # === Exemplo do DATAFRAME para referencia da explicao acima ===
        #         Proximo    0    1
        #         Atual
        #         0        0.7  0.5
```

```
        #         1         0.0   1.0
        x = pd.crosstab(pd.Series(mk[:-1], name='Atual'),pd.Series(mk[1:
                                                ],name='Proximo'),normalize=
                                                0)
        markov_chain_full.append(x)

    # Reutilizar as funcoes j   definidas
    list_00, list_01, list_10, list_11 =
                                            isolar_variaveis_matrix_markov_chains
                                            (markov_chain_full)
    r_full_00, r_full_01, r_full_10, r_full_11 = media_matrix_markov(
                                            list_00, list_01, list_10,
                                            list_11)

    return r_full_00, r_full_01, r_full_10, r_full_11

def conditional_entropy_in_dct(list_36_gray_img, list_64_gray_img):
    from scipy.fftpack import fft, dct
    from pyitlib import discrete_random_variable as drv

    # Variaveis
    list_36_dct = []
    list_36_conditional_entropy = []
    chain_list_36_dct = []
    chain_list_36_origin = []

    # Converter cada bloco em DCT-II
    for i in range(len(list_36_gray_img)):
            list_36_dct.append(dct(list_36_gray_img[i]))

    for i in range(len(list_36_dct)):
        chain_list_36_dct.append(np.concatenate(list_36_dct[i]).ravel())

    for i in range(len(list_36_gray_img)):
        chain_list_36_origin.append(np.concatenate(list_36_gray_img[i]).
                                                ravel())

    # Aplicar a Conditional Entropy para cada bloco/slice
    for i in range(len(chain_list_36_dct)):
        list_36_conditional_entropy.append(drv.entropy_conditional(
                                                chain_list_36_dct[i],
                                                chain_list_36_origin[i],
                                                base=np.exp(2)))

    return list_36_conditional_entropy

def criar_csv(path):
```

```
 # Inicia com o caminho absoluto
caminho_absoluto = path + os.sep + 'csv' + os.sep

# Confere se j  existe a pasta para receber o CSV
if not os.path.exists(caminho_absoluto):
    os.makedirs(caminho_absoluto)
    print("Criado Diret rio:", caminho_absoluto)

# Cria o arquivo com o padr o db_stego.csv ou db_origin.csv
caminho_ab_csv = caminho_absoluto + 'db_stego.csv'
print("Caminho do Arquivo CSV: ", caminho_ab_csv)

arquivo = open(caminho_ab_csv, 'a+', encoding="utf-8")
arquivo.writelines("""Stego?;Entropy_img_full;Mean_entropy_slices;
                                    Stdev_entropy_slices;
                                    Mean_MarkovChains_00;
                                    Mean_MarkovChains_01;
                                    Mean_MarkovChains_10;
                                    Mean_MarkovChains_11;
                                    Stdev_MarkovChains_00;
                                    Stdev_MarkovChains_01;
                                    Stdev_MarkovChains_10;
                                    Stdev_MarkovChains_11;
                                    Mean_imgfull_MarkovChains_00;
                                    Mean_imgfull_MarkovChains_01;
                                    Mean_imgfull_MarkovChains_10;
                                    Mean_imgfull_MarkovChains_11;
                                    Conditional_Entropy_Slice_01;
                                    Conditional_Entropy_Slice_02;
                                    Conditional_Entropy_Slice_03;
                                    Conditional_Entropy_Slice_04;
                                    Conditional_Entropy_Slice_05;
                                    Conditional_Entropy_Slice_06;
                                    Conditional_Entropy_Slice_07;
                                    Conditional_Entropy_Slice_08;
                                    Conditional_Entropy_Slice_09;
                                    Conditional_Entropy_Slice_10;
                                    Conditional_Entropy_Slice_11;
                                    Conditional_Entropy_Slice_12;
                                    Conditional_Entropy_Slice_13;
                                    Conditional_Entropy_Slice_14;
                                    Conditional_Entropy_Slice_15;
                                    Conditional_Entropy_Slice_16;
                                    Conditional_Entropy_Slice_17;
                                    Conditional_Entropy_Slice_18;
                                    Conditional_Entropy_Slice_19;
                                    Conditional_Entropy_Slice_20;
```

```
                                        Conditional_Entropy_Slice_21;
                                        Conditional_Entropy_Slice_22;
                                        Conditional_Entropy_Slice_23;
                                        Conditional_Entropy_Slice_24;
                                        Conditional_Entropy_Slice_25;
                                        Conditional_Entropy_Slice_26;
                                        Conditional_Entropy_Slice_27;
                                        Conditional_Entropy_Slice_28;
                                        Conditional_Entropy_Slice_29;
                                        Conditional_Entropy_Slice_30;
                                        Conditional_Entropy_Slice_31;
                                        Conditional_Entropy_Slice_32;
                                        Conditional_Entropy_Slice_33;
                                        Conditional_Entropy_Slice_34;
                                        Conditional_Entropy_Slice_35;
                                        Conditional_Entropy_Slice_36;"""
                                        )
    arquivo.write("\n")
    arquivo.close()
    return caminho_ab_csv

def feature_to_list(var_00, var_01, var_10, var_11):
    # Variaveis
    lista_append = []

    # Apend as variaveis
    lista_append.append(var_00)
    lista_append.append(var_01)
    lista_append.append(var_10)
    lista_append.append(var_11)

    return lista_append

def salvar_csv(input_csv,id_stego,ent_full,mean_ent,stdev_ent,l_mean_mk,
                                    l_stdev_mk,l_mean_full,l_cond_ent):
    # As variaveis com l_ no inicio s o listas.

    arquivo = open(input_csv, 'a+', encoding="utf-8")

    # Escrevendo as features simples
    arquivo.write(str(id_stego))
    arquivo.write(";")
    arquivo.write(str(ent_full))
    arquivo.write(";")
    arquivo.write(str(mean_ent))
    arquivo.write(";")
    arquivo.write(str(stdev_ent))
```

```
    arquivo.write(";")

    # Escrevendo as features em lista com loop
    for i in l_mean_mk:
        arquivo.write(str(i))
        arquivo.write(";")

    for j in l_stdev_mk:
        arquivo.write(str(j))
        arquivo.write(";")

    for k in l_mean_full:
        arquivo.write(str(k))
        arquivo.write(";")

    for x in l_cond_ent:
        arquivo.write(str(x))
        arquivo.write(";")

    arquivo.write("\n")
    arquivo.close()
    print('\nSalvo')

def main():
    # === PRE FEATURES ===
    # ====================

    # Calcular tempo inicial do script
    tempo_inicial = dt.datetime.now().strftime("%H:%M:%S")

    # Dividir a img em 64 partes
    slice_img(argv_img)

    # Abrir cada img do dir 64_imgs
    list_gray_36, list_gray_64 = open_64_imgs()

    # Extrair a entropia para cada bloco(slices).
    lista_entropia_blocos = entropia_por_bloco(list_gray_64)

    # Transformar cada bloco em Lista de Markov Chains
    matrix_markov_chains = markov_chains(list_gray_64)

    # # Isolar variaveis do Dataframe para calculo de M dia e Desvio
                                        Padrao
    mk_00, mk_01, mk_10, mk_11 = isolar_variaveis_matrix_markov_chains(
                                        matrix_markov_chains)
```

```
    # === FEATURES ===
    # ================

    # FEATURE 1 - Entropia da imagem inteira
    entropia_img_full = extract_entropy_full(argv_img)

    # FEATURE 2 - Media da entropia de cada bloco da imagem
    media_entropia = media_entropia_blocos(lista_entropia_blocos)

    # FEATURE 3 - Desvio Padrao da entropia de cada bloco da imagem
    stdev_entropia = stdev_entropia_blocos(lista_entropia_blocos)

    # FEATURE 4-7 - M dia de markov chains por cada bloco (0/0, 0/1, 1/
                                        0, 1/1).
    media_mk_00, media_mk_01, media_mk_10, media_mk_11 =
                                        media_matrix_markov(mk_00, mk_01
                                        , mk_10, mk_11)

    # FEATURE 8-11 - Desvio Padrao de markov chains por cada bloco (0/0,
                                         0/1, 1/0, 1/1).
    stdev_mk_00, stdev_mk_01, stdev_mk_10, stdev_mk_11 =
                                        stdev_matrix_markov(mk_00, mk_01
                                        , mk_10, mk_11)

    # FEATURE 12-15 - "The average probability" por toda a imagem para
                                        cada uma das
    #                   transi  es (0/0, 0/1, 1/0, 1/1).
    media_full_00, media_full_01, media_full_10, media_full_11 =
                                        media_full_img(argv_img)

    # FEATURE 16-51 - The conditional entropy for each non-boundary
    #                 position in the 8 x 8 DCT coefficient grid,
                                        calculated for
    #                 the entire image.
    list_36_features_conditional_entropy = conditional_entropy_in_dct(
                                        list_gray_36, list_gray_64)

    # === FINALIZACAO ===
    # ====================
    # Deleta o dir 64_slices
    deleta_dir_64_slices(dir_64_imgs)

    # Criar o CSV
    dir_base = identificar_os()
    caminho_csv = criar_csv(dir_base)

    # Transformar as features em listas para facil insercao
```

```
    list_media_mk = feature_to_list(media_mk_00, media_mk_01,
                                    media_mk_10, media_mk_11)
    list_stdev_mk = feature_to_list(stdev_mk_00, stdev_mk_01,
                                    stdev_mk_10, stdev_mk_11)
    list_media_mk_full = feature_to_list(media_full_00, media_full_01,
                                    media_full_10, media_full_11)

    # Salva em um CSV os dados recolhidos
    salvar_csv(caminho_csv,stego,entropia_img_full,media_entropia,
                                    stdev_entropia,list_media_mk,
                                    list_stdev_mk,list_media_mk_full
                                    ,
                                    list_36_features_conditional_entropy
                                    )

    # Encerra o calculo do tempo do script
    print("--- Tempo Inicial de Execu  o:   "+ tempo_inicial +" ---")
    tempo_final = dt.datetime.now().strftime("%H:%M:%S")
    print("--- Tempo Final de Execu  o:     "+ tempo_final  + " ---")


if __name__ == "__main__":
        main()
```

A.8 Projeto Final - Treinamento

```
# -*- coding: utf-8 -*-
from IPython.display import Image
import pydotplus
import pandas as pd
import numpy as np
from io import StringIO
from google.colab import files
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.preprocessing import StandardScaler
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams["figure.figsize"] = 10, 5
%matplotlib inline

# Upload do db full
files.upload()
df_db_full = pd.read_csv("db_full.csv")
```

```
df_db_full

# Separar as features (x) e o target(y)
x_stego = df_db_full.drop('Stego?', axis=1)
y_stego = df_db_full['Stego?']


# Separar treino e teste de x,y.
xtrain, xtest, ytrain, ytest = train_test_split(x_stego, y_stego,
                                    test_size=0.25,

# Naive Bayes
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(xtrain, ytrain)
y_model_naive_bayes = model.predict(xtest)
print("Naive Bayes accuracy : ",accuracy_score(ytest,
                                    y_model_naive_bayes))

# Decision Tree
from sklearn.tree import DecisionTreeClassifier, export_graphviz
tree = DecisionTreeClassifier()
tree = tree.fit(xtrain, ytrain)
y_model_decision_tree = tree.predict(xtest)

print("Decision Tree accuracy: ",accuracy_score(ytest,
                                    y_model_decision_tree))

# Exibindo a arvore de decis es em imagem

dot_data = StringIO()
export_graphviz(tree, out_file=dot_data)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
                                                random_state=1)

# Perceptron (MLP)
from sklearn.neural_network import MLPClassifier

# Aplicar random_state ao separar train e test
xtrain_mlp, xtest_mlp, ytrain_mlp, ytest_mlp = train_test_split(x_stego,
                                     y_stego,test_size=0.25,
                                                random_state=0)

mlp = MLPClassifier(hidden_layer_sizes=(8,8,8), activation='relu',
                                    solver='adam', max_iter=2000)
mlp.fit(xtrain, ytrain)
```

```
predict_train = mlp.predict(xtrain_mlp)
predict_test = mlp.predict(xtest_mlp)

# Score
print(confusion_matrix(ytrain_mlp,predict_train))
print(classification_report(ytrain_mlp,predict_train))
```

APÊNDICE B – Reprodução da Aplicação Final

Neste apêndice consta as instruções necessárias para a reprodução do projeto final abordado no Capítulo 5.

B.1 Preparação do Dataset

B.1.1 Download das Imagens

Conforme abordado na Seção 4.1, é necessário 150 imagens em *JPG* para compor o dataset, então foi utilizado o script *google-images-download* criado pelo Hardik Vasa para coletar as 150 imagens de forma automatizada. Este script utiliza o motor de buscas da *Google* para pesquisar e baixar as imagens para o disco rígido local. Para mais detalhes, é necessário acessar o repositório do Github conforme o link <https://github.com/hardikvasa/google-images-download>. Os parâmetros utilizados no script foram: Keywords, Format e Limit.

B.1.2 Elaboração da Lista de Caracteres

Para elaborar o conjunto de caracteres para esconder nas imagens, foi utilizado em ambiente linux o *urandom* para gerar de forma entropica qualquer caractere, o *tr* para filtrar apenas caracteres minúsculos de A à Z, *fold* para quebra a linha de texto em 32 caracteres e *head* para exibir apenas as primeiras 75 entradas. A sintaxe completa é: **$ cat /dev/urandom | tr -dc 'a-z' | fold -w 32 | head -n 75**.

B.1.3 Esteganografar as Imagens

Foi utilizado o algoritmo *F5* para realizar a esteganografia nas imagens, a versão utilizada foi do link <https://code.google.com/archive/p/f5-steganography/> e foi elaborada na linguagem java. A sintaxe utilizada foi **$ java -jar f5.jar e -e texto_de_entrada.txt imagem_de_entrada.jpg saida_imagem_esteganografada.jpg**. Como forma de automatização, foi utilizado um loop em *Python* para esteganografar todas as imagens da aplicação.

B.2 Reprodução da Extração dos Atributos

O algoritmo para extração dos atributos foi elaborado em *Python3* e o único argumento necessário é o caminho absoluto para o arquivo alvo. Como exemplo, pode-se utilizar **$ python3 extract_features.py /$HOME/pesquisa/imagem.jpg** e sua saída será um diretório nomeado como *CSV* com um arquivo.csv que contém os atributos extraídos. Pode-se utilizar o script tanto em distribuições Linux, MacOS ou Windows, porém o caminho do diretório *CSV*

nas duas primeiras opções será */$home/csv* e no Windows será *C:\CSV*. Para a extração dos atributos, além do *Python3* instalado, é necessário possuir as seguintes bibliotecas:

- cycler==0.10.0
- decorator==4.4.2
- future==0.18.2
- image-slicer==0.3.0
- imageio==2.9.0
- joblib==0.16.0
- kiwisolver==1.2.0
- matplotlib==3.3.0
- networkx==2.4
- numpy==1.19.1
- pandas==1.0.5
- Pillow==7.2.0
- pydotplus==2.0.2
- pyitlib==0.2.2
- pyparsing==2.4.7
- python-dateutil==2.8.1
- pytz==2020.1
- PyWavelets==1.1.1
- scikit-image==0.17.2
- scikit-learn==0.23.1
- scipy==1.5.1
- six==1.15.0
- threadpoolctl==2.1.0
- tifffile==2020.7.22

B.3 Treinamento do Modelo

O *Google Colabs* foi utilizado como ambiente para o treinamento e coleta de resultados. É necessário possuir uma conta Google e todo o processamento é feito pelos servidores em nuvem. Como primeiro acesso, é recomendado utilizar o guia de introdução dado pelo link <https://colab.research.google.com/notebooks/intro.ipynb> e então criar um novo *Jupyter Notebook* para realizar o treinamento. É necessário apenas instalar duas bibliotecas: python-pydot e pydotplus, que podem ser instalados dentro do proprio *Notebook* com os comandos **!apt-get install python-pydot** e **!pip install pydotplus**. Utilizando a biblioteca *google.colab files* deve-se enviar o arquivo de extensão *CSV* e dar continuidade ao treinamento do modelo. Toda a aplicação fica salva em nuvem e seu processamento de GPU são possui custo.