Trabalho de Conclusão de Curso

Classificação de textos de decisões judiciais

Brunno Moreira Gêda
brunno@ic.ufal.br

Orientadores:

Prof. Dr. André Lage Freitas
Prof. Dr. Leonardo Viana Pereira

Maceió, 9 de abril de 2021

Brunno Moreira Gêda

Classificação de textos de decisões judiciais

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação do Instituto de Computação da Universidade Federal de Alagoas.

Orientadores:

Prof. Dr. André Lage Freitas

Prof. Dr. Leonardo Viana Pereira

Maceió, 9 de abril de 2021

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico

Bibliotecário: Marcelino de Carvalho Freitas Neto – CRB-4 - 1767

G295c Gêda, Brunno Moreira.
Classificação de textos de decisões judiciais / Brunno Moreira Gêda. – 2021.
18 f.

Orientador: André Lage Freitas.
Co-orientador: Leonardo Viana Pereira.
Monografia (Trabalho de conclusão de curso em Ciência da Computação) - Universidade Federal de Alagoas, Instituto de Computação. Maceió, 2021.

Bibliografia: f. 18.

1. Inteligência artificial. 2. Aprendizado do computador. 3. Direito. 4. Decisão judicial. 5. Processamento de linguagem natural (Computação). I. Título.

CDU: 004.8

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação do Instituto de Computação da Universidade Federal de Alagoas, aprovada pela comissão examinadora que abaixo assina.

Prof. Dr. André Lage Freitas - Orientador
Instituto de Computação
Universidade Federal de Alagoas

Prof. Dr. Leonardo Viana Pereira - Coorientador
Instituto de Computação
Universidade Federal de Alagoas

Prof. Dr. Fábio José Coutinho da Silva - Examinador
Instituto de Computação
Universidade Federal de Alagoas

Prof. Dr. José de Jesus Filho - Examinador
Consudata

Maceió, 9 de abril de 2021

Resumo

Neste trabalho, fazemos uso de técnicas de aprendizado de máquina e processamento de linguagem natural (NLP) para automatizar a classificação textual no contexto de decisões judiciais e, assim, tornar esse processo mais rápido. A importância e relevância deste trabalho se devem à grande quantidade de textos jurídicos produzidos em tribunais, o que torna ineficiente e lenta a avaliação humana. Assim, este trabalho consiste em treinar e avaliar uma base de dados de textos oriundos de Acórdãos (decisões judiciais) obtidos do Tribunal de Justiça de Alagoas para classificá-los de acordo com três possíveis resultados: recurso provido, recurso não provido, ou recurso parcialmente provido. Para isso, avaliamos o desempenho de cinco modelos de aprendizado de máquina supervisionado: Gaussian Naive Bayes, Árvore de Decisão, Máquina de Vetores de Suporte, Random Forest e XGBoost. Destacaram-se os modelos Árvore de Decisão, Máquina de Vetores de Suporte e XGBoost, que atingiram aproximadamente precisão de 99% (f1-score).

Palavras-chave: inteligência artificial, aprendizado de máquina, direito, decisão judicial

Abstract

The great amount of legal documents produced in law courts makes it very inefficient and slow to be reviewed by humans. For example, courts produce millions of legal decisions per year in Brazil. In this paper, we propose an approach to automate the classification of court decisions. Specifically, we address the classification of sentences that contains judge decisions. We use machine learning and natural language processing (NLP) techniques to classify appeal decisions as *granted, not granted*, and *partially granted*. As a result, legal professionals do not need to read the case decisions – or even their summary (*Ementa*) – to know its decision. The data set used in this paper has $1,596$ legal decisions from the State Supreme Court of Alagoas (*Tribunal de Justiça de Alagoas*). Moreover, we assessed the performance of five supervised machine learning models: Gaussian Naive Bayes, Decision Tree, Support Vector Machine, Random Forest, and XGBoost. The Decision Tree, Support Vector Machine, and XGBoost models achieved standout results with approximately an F1-score of 99%.

Key-words: artificial intelligence, machine learning, law, legal, case outcome, legal decisions

Conteúdo

1

Introdução

Uma das áreas do campo da computação que mais cresce atualmente é a inteligência artificial, ou IA, que visa simular a inteligência natural de seres humanos ou animais por meio de programas de computador. Para isso, foi desenvolvida uma variedade de técnicas diferentes.

As técnicas exploradas neste trabalho são parte da área de aprendizado de máquina, um subcampo de IA que baseia-se no treinamento de um modelo para executar alguma função através de vários casos de exemplo, refinando seu desempenho na atividade ao longo do tempo. Por exemplo, essas técnicas podem ser utilizadas quando se tem como objetivo fazer a representação de linguagem naturais humanas por meio de programas de computador. A área de estudo que investiga técnicas de representação e processamento de linguagens naturais é conhecida como Processamento de Linguagem Natural) Bird et al. (2009), ou NLP, do inglês, *Natural Language Processing*. Atualmente, os algoritmos de aprendizado de máquina estão muito presentes nas técnicas de NLP.

A automatização do deciframento e da interpretação de textos é especialmente útil na área do Direito, pois é produzida uma grande quantidade de documentos para várias situações diferentes. Essa abundância de textos pode tornar ineficiente a sua análise manual, podendo uma única análise de documentos durar até meses. O resultado de uma decisão judicial pode ser inferido a partir da interpretação da conclusão da decisão, então, ao invés de analisar os resultados manualmente, pode ser feita a digitalização dos documentos e a utilização de algoritmos de classificação para rotular as decisões por resultado automaticamente e mais rapidamente.

Na literatura, alguns trabalhos procuram fazer a classificação de textos judiciais usando NLP. No artigo Brüninghaus and Ashley (2006), os autores propõem um sistema de previsão de resultados legais que faz uso de técnicas de aprendizado de máquina em sumários de casos e categoriza trechos como “fatores", que são usados para fazer a previsão final. O artigo Sulea et al. (2017) usa casos do Tribunal de Cassação francês para fazer testes com classificação textual por meio de máquinas de vetores de suporte (SVM), sendo que detalhes do texto original

que revelam o resultado desejado das previsões foram ofuscados. Esses testes incluem a previsão do resultado de um caso e da área do Direito a qual ele pertence, e estimar a época em que a descrição do caso foi emitida. Gonçalves and Quaresma (2005) também emprega o uso de SVMs para fazer classificações de documentos legais, e faz uma avaliação da relevância que o uso de informações linguísticas, como lematização e marcação de classes gramaticais, podem ter nos resultados.

Este trabalho tem como objetivo responder à pergunta científica *"É possível utilizar técnicas de aprendizado de máquina para classificar decisões judiciais de tribunais do Brasil?"*. Para isso, este trabalho propõe utilizar técnicas de aprendizado de máquina para classificar decisões judiciais a partir das frases que descrevem a decisão. As decisões são classificadas como "provida" (procedente), "parcialmente provida" (parcialmente procedente),"não provida"(improcedente). Para validação, este trabalho utiliza cinco modelos diferentes de aprendizado de máquina, que tiveram seus desempenhos avaliados e comparados. Assim, como objetivo final, visamos, com este trabalho, construir um sistema de classificação de decisões judiciais que tenha alta acurácia, de modo que possa servir como alternativa mais rápida para análises manuais.

Este trabalho é dividido em alguns capítulos. No capítulo 2, é detalhado conceitos sobre o processamento de linguagem natural e são introduzidos os modelos de aprendizado de máquina utilizados. No capítulo 3, é descrita a metodologia do trabalho. No capítulo 4, são dispostos os resultados final da avaliação. Finalmente, o capítulo 5 conclui este trabalho e apresenta alguns trabalhos futuros.

2

Processamento de Linguagem Natural e Aprendizado de Máquina

2.1 Processamento de Linguagem Natural

No campo da computação, a automatização de sistemas que exigem a interpretação de línguas naturais humanas não é um trabalho trivial, pois essas línguas geralmente são complexas demais para poderem ser computadas por algoritmos matemáticos. Porém, a partir do uso de técnicas de representação numérica das palavras, um campo de pesquisa que está em processo de rápido desenvolvimento atualmente. Por exemplo, utilizando técnicas avançadas de aprendizado de máquina, um programa pode ser treinado para ter compreensão do funcionamento da linguagem humana. Essas técnicas fazem parte de uma sub-área da computação, conhecida como Processamento de Linguagem Natural (do inglês, *Natural Language Processing*, geralmente abreviada para NLP) Bird et al. (2009).

Para trabalhar com NLP, primeiro deve-se separar cada lexema em elementos distintos chamados tokens, em um processo chamado "tokenização". No final desse processo, cada elemento semântico é representado por um token, e geralmente são removidas construções inúteis para o objetivo final do programa, como pontuação, espaços, *strings* vazias e palavras vazias.

Além disso, pode ser feita a normalização de cada token. Uma forma de simplificação do texto para o processo de aprendizado de máquina que geralmente consiste em transformar palavras que compartilham a mesma raiz nessa própria raiz comum. Também faz parte do processo a transformação de letras maiúsculas em minúsculas, pois muitos dos softwares que efetuam algoritmos NLP são sensíveis à caixa (*case-sensitive*).

Algumas das técnicas de normalização de textos são descritas a seguir Bird et al. (2009):

- **Stemming**: redução de cada lexema para seu radical, de forma que palavras com o mesmo valor semântico são reduzidas para o mesmo radical. Por exemplo, as diferentes conju-

gações do verbo “amar”, como “amando”, “amou” ou “amará”, todas se reduzem para a raiz “am”.

- **Lematização**: redução de palavras para seus lemas, ou seja, as formas das palavras encontradas em verbetes de dicionários. Por exemplo, palavras como “amando” ou “amou” são reduzidas para o infinitivo “amar”.

Outra etapa do pré-processamento do texto é a remoção das palavras vazias (do inglês, *stop words*), que são as palavras que, por serem muito comuns no texto e não carregarem significado relevante ao projeto, são escolhidas para serem ignoradas pelo algoritmo de NLP. A remoção de palavras vazias assegura que só as palavras que trarão informação útil ao algoritmo são processadas, o que geralmente garante maior acurácia nos resultados.

O campo do NLP visa fazer a representação de linguagens naturais humanas de forma que possam ser interpretadas por computadores, e, para isso, existe uma variedade de modelos de representação. A seguir, serão detalhados alguns modelos relevantes para este trabalho:

- **Saco-de-palavras**: também conhecido como *bag-of-words*, representa textos de forma puramente quantitativa. Os lexemas e suas frequências de aparições no texto são armazenados em um vetor, e essas frequências são usadas para determinar a relevância de cada lexema no texto.

- **tf-idf** (*term frequency-inverse document frequency*): Também relaciona cada lexema à sua frequência, mas leva em conta também a probabilidade de um lexema estar presente no documento utilizado, visando dar menos importância a palavras que se repetem com frequência em textos sobre um determinado tema. O valor de cada lexema é calculado, então, a partir da divisão entre a sua frequência em um documento e a quantidade de documentos no corpo que o incluem.

2.2 Aprendizado de Máquina para classificar textos

Uma das muitas aplicações de NLP é o uso de aprendizado de máquina para classificar textos em diferentes categorias. Por exemplo, NLP poderia ser usado para classificar uma obra literária de acordo com gênero (drama, fantasia, terror, etc.) ou um artigo jornalístico de acordo com o tema (política, economia, ciência, etc.).

Classificadores supervisionados são um subtipo de classificador que se baseiam em aprendizado supervisionado, ou seja, faz uso de uma base de dados em que os rótulos esperados são previamente conhecidos para o treinamento do modelo de classificação Bird et al. (2009). Essa base de dados é conhecida como “base de treinamento”. O classificador usa essa base para fazer a extração de características, ou seja determinar as características de cada texto da base que são indicativas dos rótulos correspondentes — por exemplo, pode ser determinado que substantivos

que terminam com a letra “a” frequente são do gênero feminino, e os que terminam com “o” geralmente são do gênero masculino. É, então, usada outra base de dados chamada de “base de teste” para que o modelo classificador faça previsões baseadas nas características extraídas anteriormente. Os resultados desses testes podem ser usados para avaliar o desempenho do modelo e aplicar mudanças nele, para que possa fazer classificações cada vez mais precisas.

A seguir, serão expostos alguns modelos clássicos de aprendizado de máquina utilizados na classificação de textos, escolhidos por geralmente terem bom desempenho em aplicações diversas.

2.2.1 Gaussian Naive Bayes

São calculados os valores das probabilidades a priori de que cada rótulo será escolhido com base na frequência de cada rótulo na base de treinamento. Todos os textos na base de teste são inicializados com esses valores. Em seguida, as probabilidades de cada rótulo para cada texto na base são incrementalmente alteradas com base na extração de características dos textos, que são classificados com os rótulos que têm o maior valor de probabilidade Bird et al. (2009).

2.2.2 Árvores de Decisão (*Decision Trees*)

É construída uma árvore classificadora, que funciona como um fluxograma: cada nó representa uma característica que pode estar presente no texto a ser classificados, e as folhas representam os rótulos. A classificação é feita, portanto, a partir de decisões feitas em cada nó, iniciando pelo nó raiz, e determinada pelo valor da folha alcançada Bird et al. (2009).

2.2.3 Máquina de vetores de suporte (*Support-vector machine* ou SVM)

Este modelo de aprendizado supervisionado é usado para fazer a classificação de dados em apenas duas categorias, e por isso é denominado de classificador binário. Os dados a serem classificados são dispostos em um espaço multidimensional e divididos em duas seções por um hiperplano, de forma que a distância entre os elementos de ambas a categorias seja a maior possível, criando uma distinção clara Cortes and Vapnik (1995).

Máquinas de vetores de suporte têm várias aplicações, como classificação de textos e imagens, e reconhecimento de escrita à mão.

2.2.4 *Random Forest*

São construídas múltiplas árvores de decisão a partir da base de dados, cada uma usando seleção de um subconjunto aleatório de características, e é usada a moda ou a média das previsões para fazer a classificação. Esse modelo foi criado para resolver o problema de sobreajuste comum em árvores de decisão Ho (1995).

2.2.5 XGBoost

XGBoost é uma biblioteca de aprendizado de máquina que pode ser usada para classificar textos. Faz uso de um algoritmo conhecido como *gradient boosting*, que faz uso de um conjunto de modelos preditivos mais fracos para fazer predições fortes, por meio da otimização de uma função de custo. O XGBoost é baseado em um conjunto de árvores de decisão, além de incluir uma variedade de técnicas de otimização que o diferenciam de outros algoritmos de *gradient boosting* Chen and Guestrin (2016).

2.3 Métodos de avaliação

Para garantir que um modelo de aprendizado supervisionado está funcionando como esperado, esse modelo, depois de passar pela base de treinamento, deve ser utilizado em uma outra base de dados chamada "base de validação". Nesta parte do processo, é avaliado o desempenho do modelo e seus parâmetros são reajustados de acordo com os resultados Bird et al. (2009). Algumas das métricas mais usadas no processo de validação são acurácia (proporção de itens na base de dados que foram identificados corretamente), precisão (proporção dos itens encontrados em uma busca que são relevantes), e revocação (proporção dos itens relevantes que foram encontrados em uma busca).

Alguns dos principais métodos de avaliação de modelos classificadores são:

- ***F-score***: Medida numérica de desempenho do modelo que é resultado do cálculo da média harmônica da precisão e a revocação.

- **Matriz de confusão**: Matriz que representa dados da previsão do modelo classificador, em que as fileiras da matriz representam a proporções dos valores previstos pelo modelo e as colunas representam os valores reais. Cada elemento da matriz, portanto, representa a proporção em que cada rótulo foi dado a um item pelo modelo com respeito a seu valor verdadeiro.

- **Validação cruzada**: A base de dados é dividida em subconjuntos, que são tomados de forma iterativa como bases de teste para o modelo classificador, enquanto seus subconjuntos complementares servem como bases de treinamento. É calculada, então, a média de desempenho do modelo em cada iteração, que é usada para medir o desempenho geral. A validação cruzada é um conjunto de uma variedade de técnicas que têm o mesmo embasamento, e existem diversos métodos de validação cruzada, como o método *holdout*, que é feito em apenas uma iteração com dois subconjuntos, ou o método *k-fold*, que divide a base em k subconjuntos de tamanho igual que são usados como base de teste em k iterações.

3

Metodologia

Este capítulo apresenta a metodologia empregada para responder à pergunta científica *"É possível utilizar técnicas de aprendizado de máquina para classificar decisões judiciais de tribunais do Brasil?"*, apresentada no Capítulo 1. As próximas seções expõem mais detalhes sobre a metodologia deste trabalho.

3.1 Conjunto de dados

Primeiramente, é fundamental entender o conjunto de dados (*data set*) para melhor compreender o problema e as principais características dos dados. Este trabalho utilizou uma base de dados de decisões de segunda instância (Acórdãos) do Tribunal de Justiça de Alagoas Lage-Freitas et al. (2019). Da base de dados, foram extraídos 1.605 textos (saída da linha 10 do Código-fonte 1 3.2) que continham as frases com as decisões finais dos Acórdãos e seus respectivos rótulos em relação ao resultado da decisão. Posteriormente, foram removidas da base de dados 29 decisões que não haviam sido marcadas com rótulo, restando um total de **1.596** amostras.

Neste trabalho, foram usados três rótulos diferentes para classificar os textos que descrevem a decisão dos Acórdãos: **yes** (decisão completamente favorável), **no** (decisão não favorável), **partial** (decisão parcialmente favorável), sendo consideradas apenas decisões de mérito. A lista a seguir contém exemplos de textos da base de dados, e seus rótulos correspondentes:

- *"Recurso conhecido e provido" - yes*
- *"Recurso conhecido e rejeitado" - no*
- *"Recurso conhecido e provido em parte" - partial*
- *"Provimento do recurso do município de Arapiraca" - yes*

- *"Embargos de declaração conhecidos e não acolhidos" - no*
- *"Apelo conhecido e provido em parte" - partial*

3.2 Classificação de decisões através de aprendizado de máquina supervisionado

A própria pergunta de pesquisa deste trabalho já propõe a utilização de parte da metodologia, ou seja, algoritmos de aprendizado de máquina. Essa escolha deu-se porque a literatura da área aponta para soluções que utilizam o aprendizado de máquina como método para extrair informações de textos jurídicos Brüninghaus and Ashley (2006); Sulea et al. (2017); Gonçalves and Quaresma (2005). Assim, resta decidir qual classe de algoritmo deve ser utilizada. Optou-se pela utilização do aprendizado de máquina supervisionado porque é a opção mais eficaz quando se tem uma base de dados que foi previamente rotulada. Dentre os algoritmos existentes na literatura, foram utilizados os algoritmos Gaussian Naive Bayes, Árvore de Decisão, Máquina de Vetores de Suporte, Random Forest e XGBoost, descritos na Seção 2.2. Esses algoritmos foram escolhidos por se tratarem de abordagens clássicas, comumente empregadas para tarefas de classificação e que geralmente apresentam bom desempenho.

A metodologia completa utilizada neste trabalho foi implementada pelo programa descrito abaixo, que faz uso de cada uma das técnicas descritas para classificar o conjunto de dados usado. O código-fonte e os demais dados para reproduzir os resultados obtidos nesse trabalho estão disponíveis no endereço a seguir: `https://github.com/Brunno990/tcc-nlp`

Primeiro, são selecionados os dados rotulados que interessam o estudo, sendo eles apenas os rótulos *yes, no, partial*, totalizando 1.523 amostras, de acordo com a saída na linha 18 do Código-fonte 1. É, então, criada uma outra versão da base de dados em que 255 decisões rotuladas com "*no*"foram removidas, que será avaliada separadamente. Em seguida, é feita uma limpeza geral do texto das bases de dados. Primeiro, a identificação de palavras vazias (saída da linhas 30-37 do Código-fonte 1), mas não considerando como palavras vazias as palavras que semanticamente são importantes para o problema de pesquisa: *'conhecida', 'conhecido', 'não', 'sem', 'parte'*. Essas palavras são comumente consideradas como palavras vazias para outros problemas de processamento de linguagem natural.

Já nas linhas 44-63 do Código-fonte 1, são removidas as palavras vazias, o texto é padronizado colocando todas as letras em caixa baixa, as pontuações são removidas e é feita a lematização, como descrito na Seção 2.1. Depois, as base de dados são reestruturadas, representando os rótulos das decisões (*yes, no, partial*) por valores numéricos e incluindo uma coluna com o texto limpo (linhas 131-141 do Código-fonte 1). Por fim, os modelos são treinados e avaliados usando validação cruzada nas linhas 143-188 do Código-fonte 1.

Código-fonte 1. Programa que implementa a Metodologia utilizada neste trabalho.

```
import numpy as np
import pandas as pd
import spacy

with open('2019-07-03-brunno.csv') as csvfile:
    df = pd.read_csv(csvfile, ';')
    df.columns = ['text', 'label']
csvfile.close()
df.shape
(1605, 2)

print(df['label'].value_counts())

# selecionamos apenas os dados rotulados como yes, no, partial
df_op = (df['label'] == 'yes') | (df['label'] == 'no') | (df['label'] == 'part
#| (df['label'] == 'prejudicada') | (df['label'] == 'not-cognized') | (df['lab
df = df[df_op]
df.shape
(1523, 2)

df2 = df.drop(df[df['label'].eq('no')].sample(255).index)

import time

start = time.time()

import numpy as np
import pandas as pd
import spacy

nlp = spacy.load('pt')
stopwords = spacy.lang.pt.stop_words.STOP_WORDS
# exclui algumas palavras da lista de stop-words
stopwords_to_remove = ['conhecida', 'conhecido', 'n o', 'sem', 'parte']
for word in stopwords_to_remove:
    stopwords.remove(word)
# adiciona algumas palavras da lista de stop-words
stopwords.add('e')

# verificando a remo  o e adi  o de stop-words
if 'n o' in stopwords: print('n o     stopword')
```

```
41 if 'conhecida' in stopwords: print('conhecida     stopword')
42 if 'e' in stopwords: print('e     stopword')
43
44 def apply_cleaning_function_to_list(X):
45     cleaned_X = []
46     for element in X:
47         cleaned_X.append(clean_text(element))
48     return cleaned_X
49
50 def clean_text(raw_text):
51     """Lower case, remove stop-words, remove punctuation, and extracts lemmas.
52     text = raw_text.lower()
53     doc = nlp(text)
54     words = [w for w in doc if not w.is_stop and not w.is_punct]
55     return [w.lemma_ for w in words]
56
57 text_to_clean = list(df['text'])
58 df['cleaned_text'] = apply_cleaning_function_to_list(text_to_clean)
59 df
60
61 text_to_clean = list(df2['text'])
62 df2['cleaned_text'] = apply_cleaning_function_to_list(text_to_clean)
63 df2
64
65 end = time.time()
66 total_time = end - start
67 print('Execution time in seconds: ' + str(total_time) )
68
69
70 df.to_csv('dados_limpos.csv', ';', index=False)
71 df2.to_csv('dados_limpos2.csv', ';', index=False)
72
73 import numpy as np
74 import pandas as pd
75 from sklearn.feature_extraction.text import CountVectorizer
76 from sklearn.feature_extraction.text import TfidfTransformer
77 from sklearn.naive_bayes import MultinomialNB
78
79 labels = ['yes', 'no', 'partial']#, 'prejudicada', 'not-cognized', 'conflito-c
80
81 def label_to_number(x):
```

```
82      return {
83          'yes': 0,
84          'no': 1,
85          'partial': 2#,
86  #          'prejudicada': 3,
87  #          'not-cognized': 4,
88  #          'conflito-competencia': 5
89      }[x]
90
91  from sklearn.feature_extraction.text import TfidfVectorizer
92  from sklearn import preprocessing
93  from sklearn.model_selection import GridSearchCV
94  from sklearn.svm import SVC
95  from xgboost.sklearn import XGBClassifier
96  from sklearn.model_selection import cross_val_score
97  from sklearn.naive_bayes import GaussianNB
98  from sklearn.tree import DecisionTreeClassifier
99  from sklearn.model_selection import ShuffleSplit
100 from sklearn.ensemble import RandomForestClassifier
101 from sklearn.model_selection import train_test_split
102 import time
103
104 def svc_param_selection(X, y, nfolds):
105     Cs = [0.001, 0.01, 0.1, 1, 10]
106     gammas = [0.001, 0.01, 0.1, 1]
107     degree=[1,2,3,4,5,6,7,8,9]
108     kernel=[ 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed']
109     param_grid = {'C': Cs, 'gamma' : gammas}
110     grid_search = GridSearchCV(SVC(kernel='rbf'), param_grid, cv=nfolds)
111     grid_search.fit(X, y)
112     grid_search.best_estimator_
113     return grid_search.best_estimator_
114
115 def xgboost_param_selection(X, y, nfolds):
116     param_grid = {'objective':['multi:softmax'],
117                 'learning_rate': [0.3], #so called `eta` value
118                 'max_depth': [6],
119                 'min_child_weight': [1,2,3],
120                 'silent': [1],
121                 'subsample': [1],
122                 'colsample_bytree': [1],
```

```
123                 'n_estimators': [1000], #number of trees, change it to 1000 for
124                 'missing':[-999]}
125     grid_search = GridSearchCV(XGBClassifier(), param_grid, cv=nfolds)
126     grid_search.fit(X, y)
127     grid_search.best_estimator_
128     return grid_search.best_estimator_
129
130 def classification(base, cenario):
131     with open(base) as csvfile:
132         df = pd.read_csv(csvfile, ';')
133         df.columns = ['text', 'label', 'cleaned_text']
134     csvfile.close()
135     df
136
137     # codificando o label
138     for i in range(len(df['label'])):
139         label = label_to_number(df['label'][i])
140         df['label'][i] = label
141     df
142
143     X_preprocessed = df['cleaned_text']
144     vectorizer = TfidfVectorizer()
145     vectorizer.fit(X_preprocessed)
146     X_tfidf = vectorizer.transform(X_preprocessed)
147
148     le = preprocessing.LabelEncoder()
149     y_coded=le.fit_transform(df['label'])
150     y_coded
151
152     start = time.time()
153     num_folds=5
154     cv = ShuffleSplit(n_splits=num_folds, test_size=0.2, random_state=1)
155
156     best_SVM=svc_param_selection(X_tfidf, y_coded, num_folds)
157     print('Best SVM parameters are:\n' + str(best_SVM))
158     best_XG=xgboost_param_selection(X_tfidf, y_coded, num_folds)
159     print('Best XGBoost parameters are:\n' + str(best_XG))
160
161     clf = GaussianNB()
162     clf2=DecisionTreeClassifier()
163     clf3=best_SVM
```

```
164     clf4=RandomForestClassifier(n_estimators=100, max_depth=4,random_state=0)
165     clf5=best_XG
166     scores = cross_val_score(clf, X_tfidf.toarray(), y_coded, cv=cv)
167     scores2 = cross_val_score(clf2, X_tfidf.toarray(), y_coded, cv=cv)
168     scores3 = cross_val_score(clf3, X_tfidf.toarray(), y_coded, cv=cv)
169     scores4 = cross_val_score(clf4, X_tfidf.toarray(), y_coded, cv=cv)
170     scores5 = cross_val_score(clf5, X_tfidf.toarray(), y_coded, cv=cv)
171
172     end = time.time()
173     total_time = end - start
174     print('Execution time in seconds: ' + str(total_time) )
175
176     print('CEN RIO '+cenario)
177
178     print('== SCORES ==      \n   Gaussian NB  :'+str(scores) +'\n   Decision T
179     '\n SVM          :'+ str(scores3) +'\n   Random Forest:'+str(scores4)+'\n
    XGBoost      :'+str(scores5))
180
181     print('\n== SCORES MEAN AND STD==')
182     print("Accuracy NB: %0.4f (+/- %0.4f)" % (scores.mean(), scores.std() ))
183     print("Accuracy DT: %0.4f (+/- %0.4f)" % (scores2.mean(), scores2.std() ))
184     print("Accuracy SVM: %0.4f (+/- %0.4f)" % (scores3.mean(), scores3.std() )
185     print("Accuracy RF: %0.4f (+/- %0.4f)" % (scores4.mean(), scores4.std() ))
186     print("Accuracy XGBOOST: %0.4f (+/- %0.4f)" % (scores5.mean(), scores5.std
187 classification('dados_limpos.csv', '1')
188 classification('dados_limpos2.csv', '2')
```

4

Resultados

Foram criados dois cenários para validação da metodologia proposta no Capítulo 3, apresentados na Tabela 4.1. No Cenário 1, fazemos uso de todos os dados disponíveis na base. No Cenário 2, foram removidos aleatoriamente da base de dados 255 decisões com o rótulo "*no*", com o objetivo de equalizar a distribuição de rótulos diferentes e, assim, eliminar um possível viés no modelo.

Tabela 4.1: Cenários. O Cenário 1 inclui todo o conjunto de dados enquanto o Cenário 2 remove foi balanceado, igualando a quantidade de exemplos `no` e `yes`.

Cenário	**no**	**yes**	**partial**	**Total**
Cenário 1	697	442	384	1.523
Cenário 2	442	442	384	1.268

Foi realizada uma validação cruzada (Seção 2.3) dos modelos Gaussian Naive Bayes, Árvore de Decisão, Máquina de Vetores de Suporte, Random Forest e XGBoost (Seção 2.2), tal qual implementada pelo Código-fonte 1 no Capítulo 3. O computador utilizado para executar o experimento tem as seguintes características: máquina virtual Ubuntu 64-bit com memória principal de 5.277 MB executando na máquina física Windows 10 64-bits com processador Intel Core i5-4400 3.10 GHz e 8 GB de memória principal.

As Tabelas 4.2 e 4.3 apresentam os resultados do Cenários 1 e 2 respectivamente. Ambos cenários mostraram resultados semelhantes entre si, com F1-score muito alta e excedendo 0.9 para todos os modelos exceto o Gaussian NB, que teve desempenho consideravelmente pior nos dois casos (F1-score de 0.5751 no cenário 1 e 0.4071 no cenário 2). Destacam-se, principalmente, os desempenhos dos modelos Árvore de Decisão e SVM, que têm F1-score acima de 0.99 e os desvios padrão mais baixos em ambos cenários.

Tabela 4.2: **Resultados do Cenário 1**: todo o conjunto de dados. Comparação de modelos utilizados para predição de classificação de textos de decisões judiciais utilizando a métrica F1-score e validação cruzada 5-fold.

Modelos	**F1-score ± Desvio padrão**
Gaussian NB	0.5751 ± 0.0856
Decision Tree	**0.9987 ± 0.0026**
SVM	**0.9974 ± 0.0032**
Random Forest	0.9003 ± 0.0138
XGBoost	**0.9987 ± 0.0026**

Tabela 4.3: **Resultados do Cenário 2**: todo o conjunto de dados. Comparação de modelos utilizados para predição de classificação de textos de decisões judiciais utilizando a métrica F1-score e validação cruzada 5-fold.

Modelos	**F1-score ± Desvio padrão**
Gaussian NB	0.4071 ± 0.0270
Decision Tree	**0.9921 ± 0.0043**
SVM	**0.9945 ± 0.0040**
Random Forest	0.9354 ± 0.0365
XGBoost	0.9890 ± 0.0084

5

Conclusão

A principal conclusão deste trabalho é que é possível classificar decisões judiciais usando modelos de aprendizado de máquina supervisionados.

O modelo Gaussian NB teve F1-score de apenas 57.51%, mas os outros modelos conseguiram atingir uma precisão altíssima, com F1-score de $\sim 99\%$ para os modelos Decision Tree, SVM e XGBoost. Quando o conjunto de dados é balanceado, a precisão para os modelos Decision Tree, SVM e XGBoost, apesar de um pouco mais baixa, ainda é de $\sim 99\%$ para todos os três modelos. Além disso, a classificação é consistente pois todos os modelos mostram desvios padrão insignificantes para o problema.

A acurácia quase perfeita consistentemente obtida nos resultados levantou suspeitas de um possível viés no conjunto de dados usado. Após a avaliação dos resultados, percebeu-se uma baixa variedade nos dados utilizados, com uma maioria considerável dos textos utilizando dos mesmos padrões linguísticos, os quais criam distinção clara entre os resultados das decisões. Pode-se concluir, então, que se feito em uma base de dados mais diversa, é provável que alcançaríamos resultados muito diferentes. Se considerarmos que os textos de decisões do Tribunal de Justiça de Alagoas podem ser representados pela base de dados utilizado, podemos esperar alta acurácia desses modelos, se usados para classificar dados desse tribunal. Entretanto, isso não garante precisão em casos de textos mais ambíguos ou em textos de outros tribunais que podem não ser bem representados pela base de dados. Ou seja, para caso de texto cujos padrões linguísticos sejam diferentes dos empregados na base de dados utilizada neste trabalho.

O uso da metodologia proposta neste trabalho, na prática, poderia automatizar o processo de classificação quase por completo e, assim, torná-lo muito mais rápido em uma variedade de usos. Por exemplo, a classificação automática pode ser utilizada na predição de resultados das decisões judiciais, na uniformização da jurisprudência ou em tarefas mais simples de análise quantitativa de resultados para fins de relatórios institucionais.

Em relação à avaliação dos modelos, um trabalho futuro poderá incluir ampliar essa etapa separando a base de dados em treinamento, teste e validação com dados que não pertencem à

base de treinamento. Por fim, uma investigação relevante é comparar o desempenho de algoritmos de aprendizado de máquina com expressões regulares[1].

[1]Exemplo de classificador de decisões judiciais do TJSP que utiliza expressões regulares: https://github.com/jjesusfilho/tjsp/blob/master/R/classificar_recurso.R

Referências bibliográficas

Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, 2009.

Stefanie Brüninghaus and Kevin D. Ashley. Progress in textual case-based reasoning:predicting the outcome of legal cases from text. 2006.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. 2016. **DOI** 10.1145/2939672.2939785.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, pages 273–297, 1995. **DOI** 10.1007/BF00994018.

Teresa Gonçalves and Paulo Quaresma. Is linguistic information relevant for the classification of legal texts? pages 168–176, 06 2005. **DOI** 10.1145/1165485.1165512.

Tin Kam Ho. pages 278–282, 1995.

André Lage-Freitas, Héctor Allende-Cid, Orivaldo Santana, and Lívia de Oliveira-Lage. Predicting Brazilian court decisions. Technical report, Universidade Federal de Alagoas, apr 2019. URL http://arxiv.org/abs/1905.10348.

Octavia-Maria Sulea, Marcos Zampieri, Shervin Malmasi, Mihaela Vela, Liviu P. Dinu, and Josef van Genabith. Exploring the use of text classification in the legal domain, 2017.