UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DIEGO JESUS DA COSTA MOREIRA

CLASSIFICAÇÃO DE TEXTO USANDO CODIFICAÇÃO DE PALAVRAS

RIO DE JANEIRO
2021

DIEGO JESUS DA COSTA MOREIRA

CLASSIFICAÇÃO DE TEXTO USANDO CODIFICAÇÃO DE PALAVRAS

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Profa. Priscila Machado Vieira Lima
Co-orientador: Prof. Mário Roberto Folhadela Benevides

RIO DE JANEIRO
2021

CIP - Catalogação na Publicação

M838c

Moreira, Diego Jesus da Costa
Classificação de texto usando codificação de palavras / Diego Jesus da Costa Moreira. -- Rio de Janeiro, 2021.
48 f.

Orientadora: Priscila Machado Vieira Lima.
Coorientador: Mário Roberto Folhadela Benevides.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Matemática, Bacharel em Ciência da Computação, 2021.

1. NLP. 2. PLN. 3. Clasificação de textos. 4. Processamento de linguagem natural. I. Lima, Priscila Machado Vieira, orient. II. Benevides, Mário Roberto Folhadela, coorient. III. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a), sob a responsabilidade de Miguel Romeu Amorim Neto - CRB-7/6283.

DIEGO JESUS DA COSTA MOREIRA

CLASSIFICAÇÃO DE TEXTO USANDO CODIFICAÇÃO DE PALAVRAS

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 03 de Fevereiro de 2021

BANCA EXAMINADORA:

Priscila Machado Vieira Lima, Ph.D.
(UFRJ)

Mário Roberto Folhadela Benevides, Ph.D.
(UFF)

Carlos Eduardo Pedreira, Ph.D. (UFRJ)

João Carlos Pereira da Silva, D.Sc. (UFRJ)

Dedico esse trabalho à minha família que me apoiou em todos os momentos e me auxiliou nos periodos mais difíceis da minha vida.

AGRADECIMENTOS

Agradeço à minha familia, amigos e colegas que me auxiliaram durante a minha jornada de graduação, sem vocês eu não teria conseguido.

“Almost dead yesterday, maybe dead tomorrow, but alive, gloriously alive, today."

Robert Jordan

RESUMO

Neste trabalho, serão explicadas 3 formas de codificação de textos. Essas codificações são comparadas em busca de um melhor resultado na classificação do nosso conjunto de dados. Para realizar a classificação do texto codificado, são comparados 3 classificadores. Os nossos principais objetivos são avaliar a acurácia e tempo de execução desses classificadores usando os codificadores testados e testar a acurácia da codificação nos classificadores testados. Para realizar os testes serão usados dois conjuntos de dados, um com muitos exemplos e outros com poucos exemplos, o motivo para essa escolha é analisar como os codificadores e classificadores atuam nesses cenários distintos e tentar traçar uma curva de progressão para as métricas apuradas. Por fim, esse trabalho irá abordar uma possível combinação entre codificadores e classificadores usados. Essa escolha será baseada nos gráficos gerados e resultados obtidos. A analise principal, que será feita tanto para os codificadores quanto para os classificadores, será verificar se os modelos mais simples conseguem competir com os modelos mais robustos em acurácia e se os modelos mais complexos conseguem competir em tempo de execução.

Palavras-chave: classificação de texto. nlp. pln. wisard. svm. random forest. word2vec. w2v. tf. tfidf. bow.

ABSTRACT

In this work, 3 forms of embedding are explained, these embeddings are compared in the search of a better result in the task of classification of our data set. To classify the embedded text, 3 classifiers are compared. Our main goal is to evaluate the accuracy and time of execution of these classifiers using the embedded text and to test the embedding accuracy of our models. To perform the tests, two data sets will be used, one with many examples and others with few examples, the reason for this choice is to analyze how the embedding and classifiers perform in these different scenarios and try to plot a progression curve for the metrics calculated. Finally, this work will address a possible choice between classifiers and encoders. This choice will be based on the charts generated and results obtained, the main analysis that will be done for both embedders and classifiers will be to verify if the simplest models can compete with the more robust models in term of accuracy and the complex model can compete in term of time of execution.

Keywords: text classification. nlp. wisard. svm. random forest. word2vec. w2v. tf. tfidf. bow.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

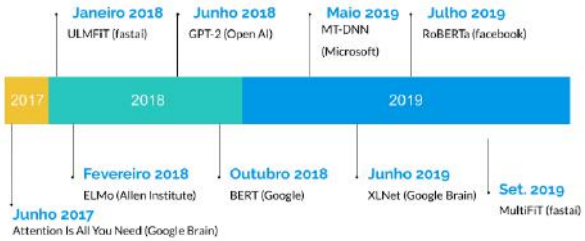
RAM	Random-Access Memory
w2v	Word2Vec
TF-IDF	Tendency frequency-Inverse document frequency
BOW	Bag-of-Words

SUMÁRIO

1 INTRODUÇÃO

Atualmente vivemos na era da informação e uma das características dessa era é o fato de existir, virtualmente, uma quantidade infinita de dados para trabalharmos e, para aprendizado de máquina, dados é muito importante. Possuir muito dado é bom, porém, o formato desse dado é de suma importância, uma vez que devemos adaptar nosso modelo de aprendizado a esse detalhe. O aumento na quantidade de dados se deve principalmente à majoração do uso das redes sociais, porém, como é possível imaginar, esses dados são, em sua maioria, expressos em linguagem natural. Com isso, uma alternativa para trabalhar com esses dados se torna imprescindível. Além disso, podemos perceber um grande movimento da comunidade cientifica para ampliar nosso conhecimento em processamento de linguagem natural(PLN). Isso deve muito ao fato de grandes industrias como Facebook, Google e Open AI estarem cada vez mais demandando dessa área.

Figura 1 – Artigos em PLN



Fonte: Guillou, Pierre (2020). Processamento de Linguagem Natural (PLN) com Deep Learning: panorama das tendências recentes[Blog Post]. Retirado de https://medium.com/@pierre$_guillou/processamento-de-linguagem-natural-pln-com-deep-learning-panorama-das-tendncias-recentes-40375c2303c33.Acessadoem16/11/2020$

Neste trabalho abordaremos algumas soluções para trabalhar com dados compostos por textos em linguagem natural. Primeiro, iremos apresentar os conceitos usados: Codificação do texto e a classificação do texto. Serão apresentados 3 modelos para transformar dados textuais em dados numéricos: *Bag-of-words* (BOW) (HARRIS, 1954), *Tendency frequency-Inverse document frequency* (TF-IDF) (JONES, 1972) e *Word2Vec* (W2V)(MIKOLOV et al., 2013).

Como nossos casos de testes neste trabalho serão classificar textos, será necessário um classificador para trabalhar com os dados após eles serem codificados. Desta forma, iremos apresentar o *Random Forest*(BREIMAN, 2001), *Support vector machine* (SVM)(BOSER; GUYON; VAPNIK, 1992) e *WiSARD* (WSD)(ALEKSANDER; THOMAS; BOWDEN, 1984).

Buscando uma comparação entre performances, iremos apresentar os resultados e fa-

zer um comparativo entre as ferramentas que foram usadas. Este comparativo será o centro do nosso trabalho, pois com ele poderemos verificar, por exemplo, de que forma a *WiSARD*(um modelo mais simples) funciona em uma tarefa complexa como classificação de textos, como o SVM(um modelo mais complexo) se comporta em relação ao tempo de execução ou até mesmo analisar como um modelo que não é baseado em redes neurais(*Random Forest*) compete com modelos que usam a forma de aprendizado mais difundida na industria atualmente.

A abordagem deste trabalho é buscar uma relação entre os classificadores e os codificadores, por exemplo, com qual classificador o BOW funciona melhor ou se o SVM alcança o melhor resultado usando a codificação do TF-IDF. Apesar de o trabalho não ter utilizado todas as formas de aprendizado ou de codificação, podemos ter uma ideia inicial de como os classificadores se comportam a essa nova abordagem de aprendizado que é tão necessária nos dias de hoje.

2 APRESENTAÇÃO DE CONCEITOS

Neste trabalho foi usada uma variedade de conceitos com o objetivo de obter a melhor combinação entre acurácia e tempo de execução, pois, apesar de um modelo alcançar uma boa acurácia, se ele consumir muito recurso para esse resultado o uso dele pode ficar bastante limitado, dado que não existem aplicações com recursos ilimitados. Os conceitos utilizados foram pesquisados em vários artigos científicos, sempre buscando a melhor abordagem para o problema apresentado.

Para classificar textos de forma mais eficiente foi desenvolvido um modelo consistindo em 3 etapas: pré-processamento do texto, codificação do texto para matrizes numéricas(*embedding*) e classificação. As etapas são independentes umas das outras, sendo assim, é possível alterá-las individualmente. Os classificadores empregados podem ser usados com outras formas de codificação e pré-processamento, e vice-versa.

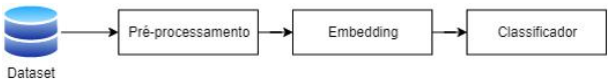


Figura 2 – Arquitetura do Modelo

2.1 CODIFICAÇÃO

Montar um modelo computacional para processar textos requer, primeiro, que seja feita uma conversão dos dados textuais para dados numéricos. Essa conversão é chamada de codificação.

Uma boa codificação faz com que textos parecidos possuam representações numéricas parecidas. Essas representações muitas vezes são vetores numéricos, onde quanto maior a dimensão, maior o poder computacional necessário para processá-lo e maior é a quantidade de informação armazenada.

Existem várias formas de codificar um texto, nesse trabalho iremos focar em *Bag-Of-Words* (BOW) (HARRIS, 1954), *Term frequency–Inverse Document Frequency* (TF-IDF) (JONES, 1972) e *Word2vec* (W2V)(MIKOLOV et al., 2013), por serem as codificações mais usadas na literatura e e por serem bases para outros modelos mais complexos.

2.1.1 BOW

BOW é uma representação de texto bastante usado no processamento de linguagem natural. Essa abordagem parte do pressuposto que as palavras, sozinhas, são responsáveis por definir a classe do texto. Isto é, o texto é definido pelas palavras que ele contém, independente da ordem que ela aparece, da sua companhia, gramática ou contexto.

BOW funciona mantendo a frequência das palavras em um texto. Como dito acima, apenas o indicativo da palavra pertencer ou não ao texto é importante, todo resto é ignorado. Assim, após a codificação, o BOW retorna uma matriz NxN, sendo N o tamanho do seu vocabulário.

Um exemplo de uso do BOW seria a codificação de dois textos:

- Texto 1: "O lobo esta na mesa"
- Texto 2: "o meu gato esta na mesa agora?"

O vocabulário dos textos é ("meu, gato, lobo, esta, agora, o, mesa, na") e a representação to texto 1 fica como:

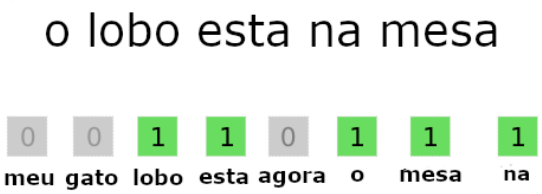


Figura 3 – Representação BOW

O modelo do BOW é muito simples de entender e implementar, e oferece muita flexibilidade para customização para o seu conjunto de dados.

Sempre que falamos em trabalhar com textos, o modelo do BOW surge como uma opção de representação do texto.

Quando falamos em predição de problemas e classificação de documentos o BOW é usado com grande sucesso, sendo assim, essas duas tarefas sãos as mais indicadas para o uso do BOW. Apesar desse sucesso, o modelo possui também algumas limitações, tais como:

Vocabulário: O vocabulário requer um design cuidadoso, principalmente para controlar o tamanho do vocabulário, pois, como foi explicado anteriormente, quando o vocabulário for muito grande a representação do texto em forma de matriz ficará excessivamente esparsa. Com isso, esse controle do vocabulário impacta diretamente a representação do documento.

Esparsidade: Representações esparsas são mais difíceis de modelar, tanto por razões computacionais (complexidade de espaço e tempo) quanto por informações, onde o desafio é fazer com que os modelos aproveitem tão pouca informação em um espaço representacional tão grande.

Significado: Ignorar a ordem das palavras, o que acarreta na não observância do contexto, e, por sua vez, descarta o significado das palavras no documento (semântica).

Contexto e significado podem oferecer muito ao modelo; a falta deles pode dar diversas interpretações a sinônimos, (“bonito” vs “lindo”), ou interpretações iguais a frases cujo contexto é diferente, (“isso é interessante” vs “isso é interessante?”), entre outros.

2.1.2 TF-IDF

TF-IDF *(Tendency frequency-Inverse document frequency)* é uma medida estatística para indicar a importância de uma palavra em um documento. Um dos usos do TF-IDF é para seleção de parâmetros. O uso do TF-IDF nesse trabalho será para buscar as palavras (parâmetros) mais importantes para cada texto e montar uma matriz NxN, sendo N o tamanho do vocabulário onde as palavras de maior importância para cada texto terá um peso maior na hora de classificar esse documento.

Um exemplo de uso do TF-IDF nos dois textos usado anteriormente:

- Texto 1: "O lobo esta na mesa"
- Texto 2: "o meu gato esta na mesa agora?"

Após a execução do TF-IDF a codificação identificará que as palavras ("gato", "lobo", "agora", "meu") são importantes para as classes de seus textos, pois são únicas em seus respectivos textos.

O algoritmo do TF-IDF é útil quando você tem um conjunto de documentos, geralmente grande, que precisa ser classificado. Esse modelo é especialmente fácil de implementar, pois não é necessário o seu treino com antecedência.

O TF-IDF é, de certa forma, bem parecido com o BOW, e possui as mesmas características, como sendo eficiente em problemas de classificação de documentos e predição de problemas, bem como por sofrer com os mesmos problemas de vocabulário, sua característica em se manter esparso e o significado das palavras.

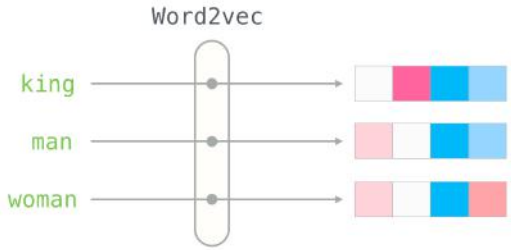
Além de seu uso para classificação de documentos, que será o abordado nesse trabalho, o TF-IDF também é bastante usado para seleção de características da base de dados para o aprendizado.

2.1.3 Word2Vec

O *Word2Vec* é um modelo de linguagem que recebe um grande conjunto de textos e retorna vetores numéricos de N dimensões representando as palavras. O objetivo dessa transformação é utilizar palavras próximas para garantir que as representações vetoriais possuam informações semânticas e sintáticas.

Para visualizar, vamos considerar uma palavra como um vetor de N dimensões entre -2 e 2. Quando mais próximo do -2, iremos pintar de vermelho, quando mais próximo do 2, iremos pintar de azul e iremos pintar de branco quando for 0. [Figura 3]

Figura 4 – Representação Word2Vec



Fonte: Alammar, Jay (2020). The Illustrated Word2vec [Blog post]. Retirado de http://jalammar.github.io/illustrated-word2vec/. Acessado em 16/11/2020

Essa representação permite que apareça alguns resultados interessantes quando fazemos aritméticas com vetores que representam as palavras. Como exemplo, podemos citar a aritmética feita em: "*king*-"*man*"+"*woman*", o resultado é um vetor parecido com a representação da palavra "*queen*". [Figura 4]

BOW e TF-IDF são formas de representação de texto, o *Word2Vec* é uma forma de representação de palavras. Para realizar a codificação do texto usando o *Word2Vec* é necessário encontrar uma forma de combinar as representações vetoriais das palavras que compõem o texto. Neste trabalho, essa combinação foi feita usando a média das representações numéricas dos vetores que compõem o texto.

Figura 5 – Operações com palavras

Fonte: Alammar, Jay (2020). The Illustrated Word2vec [Blog post]. Retirado de http://jalammar.github.io/illustrated-word2vec/. Acessado em 16/11/2020

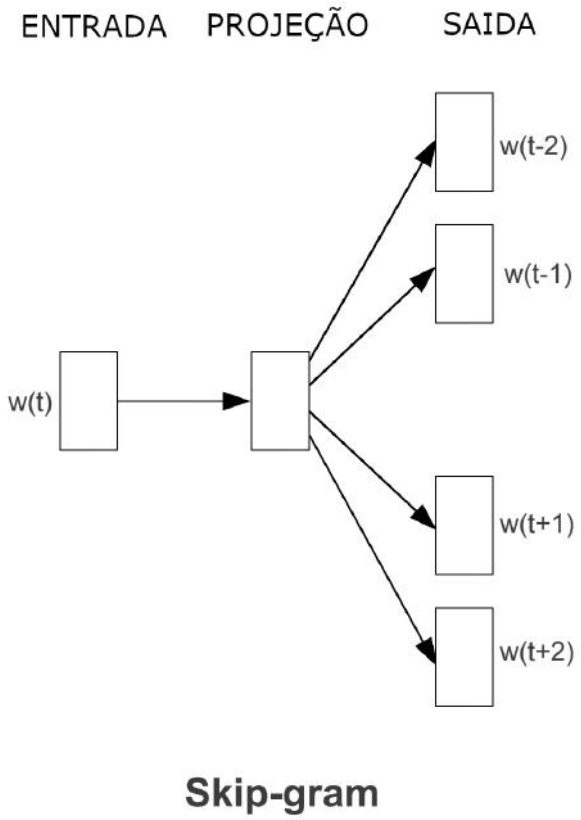
O *Word2Vec* realiza o seu aprendizado de forma não supervisionada, isto é, o conjunto de dados para treinamento não precisa ser rotulado. Para isso o treinamento será feito criando uma "tarefa falsa"para a rede treinar. As entradas e saídas não são importantes, o objetivo é simplesmente aprender os pesos da camada oculta, e esses pesos são as representações numéricas das palavras.

Além do funcionamento descrito acima, o *Word2Vec* possui duas formas de treinamento: *Skip-Gram* e *Continuous Bag of Words (CBOW)*, cada um com suas vantagens e desvantagens.

Skip-gram

A "tarefa falsa"do modelo *Skip-gram* será que, dada uma palavra, tentaremos prever as palavras vizinhas. Definiremos uma palavra vizinha pelo tamanho da janela - um hiperparâmetro.

Figura 6 – Skip-Gram



Fonte: MIKOLOV, T. et al. 2013. Acessado em 16/11/2020

Dada a frase: *"Eu gostaria de um suco de laranja"* e uma janela de tamanho 2, se a palavra-fonte for "suco", as palavras vizinhas serão (de, um, de, laranja). Logo, as tuplas de palavra-fonte e palavra-alvo serão: (suco, de), (suco, um), (suco, de) e (suco, laranja). Dentro da janela, as distâncias entre a palavra-fonte e as palavras-alvo não possuem nenhuma influência, portanto, as tuplas acima serão tratadas da mesma forma durante o treinamento.

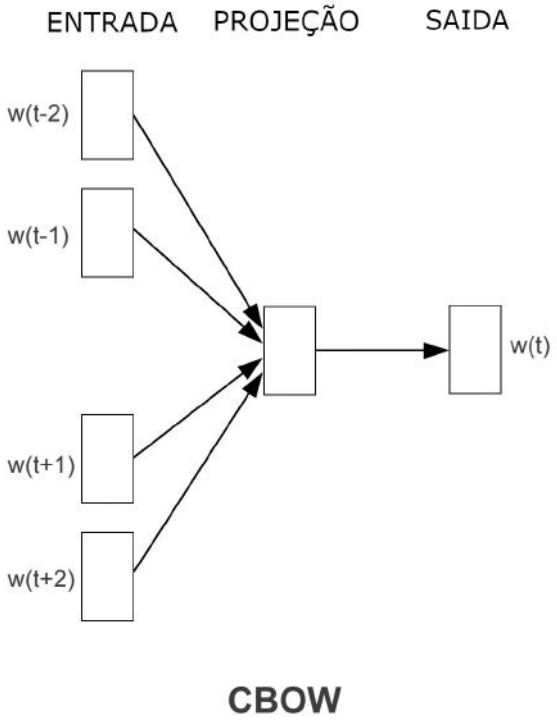
Os vetores de entrada serão representações *one-hot* da palavra-fonte com dimensão 1xV, onde V é o número de palavras no vocabulário. A única camada oculta terá a dimensão VxE, onde E é o tamanho da representação numérica da palavra e é um hiperparâmetro. A saída da camada oculta será da dimensão 1xE, que alimentaremos em uma camada *softmax*. As dimensões da camada de saída serão de 1xV, onde cada valor no vetor será a pontuação de probabilidade da palavra-alvo nessa posição. Em seguida, calcularemos os vetores de erros correspondentes a cada palavra-alvo (no exemplo, 4 vetores) e executaremos uma soma por elemento, reduzindo para 1 vetor 1xV. Em seguida, os pesos da camada oculta serão atualizados com base nesse vetor de erro acumulado.

Continuous Bag of Words (CBOW)

A "tarefa falsa"do CBOW é um pouco semelhante ao *Skip-gram*, no sentido de que ainda pegamos um par de palavras e ensinamos o modelo que elas aparecem juntas, mas em vez de adicionar os erros, adicionamos as palavras de entrada para a mesma palavra de destino.

A dimensão da camada oculta e da camada de saída permanecerá a mesma. Somente a dimensão da camada de entrada e o cálculo das ativações da camada oculta serão alterados. Se tivermos 4 palavras de contexto para uma única palavra-alvo, teremos 4 vetores de entrada 1xV. Cada um será multiplicado com a camada oculta VxE, retornando vetores 1xE. Todos os 4 vetores 1xE serão calculados como elementos para obter a ativação final que será, então, alimentada na camada *softmax*.

Figura 7 – CBOW



Fonte: MIKOLOV, T. et al. 2013. Acessado em 16/11/2020

O *Skip-Gram* funciona melhor com uma pequena quantidade de dados de treinamento; representando as palavras ou as frases que são consideradas raras de modo eficiente. Enquanto o *CBOW* leva vantagem por ser muito mais rápido do que o *skip-gram*, além de ter uma acurácia melhor quando trabalhamos com textos que contém muitas palavras repetidas.

Além dessas diferenças entre os dois modelos dentro da proposta do *Word2Vec*, o modelo como um todo requer menos memória que outras representações, além do fato de

levar em consideração as palavras da vizinhança, faz com que a representação das palavras possua mais informação.

Apesar das vantagens de usar o *Word2Vec*, encontrar os hiper-parâmetro é difícil e, além disso, a função de *softmax* é custosa do ponto de vista computacional e, consequentemente, o treino dos modelos leva muito tempo.

2.2 CLASSIFICADORES

Em ciência da computação, classificação é o processo de prever a classe de determinados pontos de dados. Por exemplo, a detecção de *spam* nos provedores de serviços de *e-mail* pode ser identificada como um problema de classificação. Esta é uma classificação binária, uma vez que existem apenas 2 classes como *spam* e não *spam*. Um classificador utiliza alguns dados de treinamento para entender como determinadas características de entrada se relacionam com a classe. Nesse caso, os *e-mails* conhecidos de *spam* e não-*spam* devem ser usados como dados de treinamento. Quando o classificador é treinado com precisão, ele pode ser usado para detectar um *e-mail* desconhecido.

Em uma definição pouco formal, podemos dizer que existem dois tipos de classificadores: "preguiçosos"e "estudiosos".

Classificadores "preguiçosos"

Os classificadores "preguiçosos"simplesmente armazenam os dados do treinamento e esperam até que os dados do teste apareçam. Quando isso ocorre, a classificação é realizada com base nos dados mais relacionados nos dados de treinamento armazenados. Comparado aos classificadores "estudiosos", os classificadores "preguiçosos"têm menos tempo de treinamento, porém mais tempo na previsão. Exemplos: *KNN*, *Case-based reasoning*.

Classificadores "estudiosos"

Os classificadores "estudiosos"constroem um modelo de classificação com base nos dados de treinamento fornecidos antes de receber os dados para classificação. Ele deve poder se comprometer com uma única hipótese que cubra todo o espaço da instância. Devido à construção do modelo, os alunos ansiosos levam muito tempo para treinar e menos tempo para prever. Exemplos: *SVM*, *Random Forest*, *WiSARD*.

Nesse trabalho, iremos focar apenas nos classificadores "estudiosos".

2.2.1 Random Forest

Para entender o classificador *Random Forest* (BREIMAN, 2001), devemos primeiro entender o conceito de árvores de decisão.

Árvores de Decisão

Uma árvore de decisão é uma ferramenta de suporte à decisão que usa um gráfico ou modelo de decisão semelhante a uma árvore e suas possíveis consequências, incluindo resultados de eventos aleatórios, custos de recursos e utilidade. Com essa ferramenta é possível exibir um algoritmo que contém apenas instruções de controle condicionais.

Dado à sua simplicidade, a forma mais fácil de explicar árvores de decisão é por meio de um exemplo.

Vamos supor que temos o objetivo de classificar se o dia é propício para jogar tênis. Como características do dia, temos: Clima, umidade e vento. Dividindo nossos dados, é possível montar a árvore de decisão abaixo.

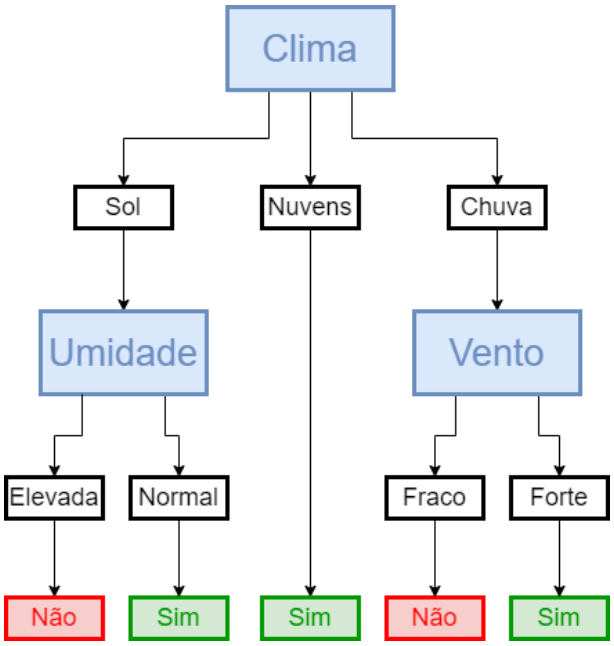


Figura 8 – Arvore de decisão

Com a árvore montada com base nos dados de treinamento, para classificar se um dia é propício para jogar tênis é preciso passear pela árvore seguindo as características do dia.

Se as características do dia forem (Clima: Sol, Umidade: Normal), o passeio na árvore ficará o seguinte:

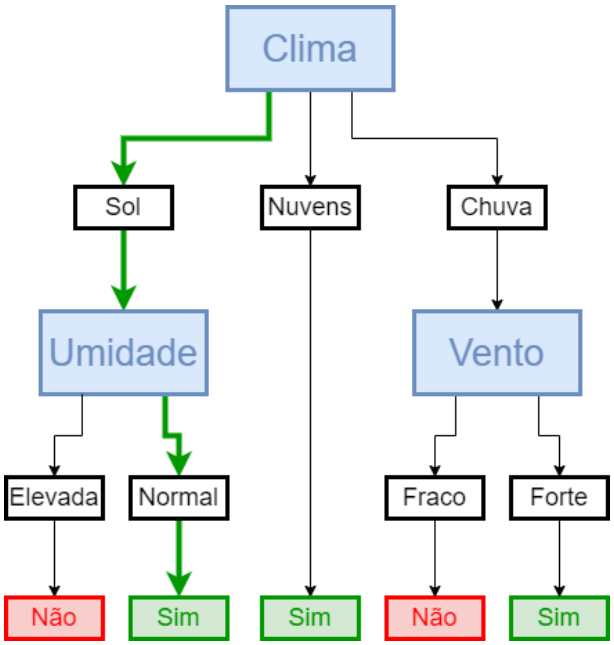


Figura 9 – Arvore de decisão

Seguindo pelo caminho destacado, é possível ver que o dia descrito seria um dia propício para jogar tênis.

Obviamente, na vida real, os dados não serão tão limpos, mas a lógica que uma árvore de decisão emprega para ser construída permanece a mesma. Em cada nó, deverá ser perguntado: "Qual característica me permitirá dividir as observações de maneira em que os grupos resultantes sejam os mais diferentes possíveis?". Essa pergunta nem sempre é fácil de responder, fazendo com que montar uma árvore ótima seja um desafio, sendo esse um dos pontos negativos desse tipo de modelo. Porém, após alcançar uma árvore sub-ótima, o modelo consegue classificar novos exemplos com bastante rapidez.

Classificador Random Forest

Random Forest consiste em um grande número de árvores de decisões individuais que funcionam como um comitê. Cada árvore classifica individualmente e vota em uma classe, a classe com mais votos se torna a previsão do nosso modelo e essa é a razão pelo qual esse modelo funciona tão bem: um grande número de modelos (árvores), relativamente não correlacionados, operando como um comitê, superará qualquer um dos modelos constituintes individuais(é o mesmo argumento utilizado para o uso de comitês) e essa chave para o modelo a baixa correlação entre suas arvores.

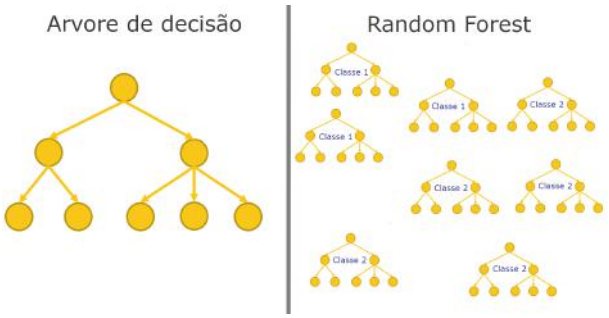


Figura 10 – Random Forest

A razão desse efeito é que as árvores se protegem de seus erros individuais (desde que nem sempre errem na mesma direção). Enquanto algumas árvores podem estar erradas, muitas outras estarão certas, portanto, como um grupo, as árvores podem se mover na direção correta.

Sendo assim, possuem 2 pré-requisitos para que o modelo possua um bom desempenho:

- O conjunto de dados deve possuir algum indicativo que garanta que os modelos criados a partir desse conjunto sejam melhores do que estimativas aleatórias.
- As previsões (e, portanto, os erros) feitas pelas árvores individuais precisam ter baixas correlações entre si.

Para garantir que as árvores da arquitetura não sejam muito correlacionadas, o modelo segue dois métodos: *Bagging* e Aleatoriedade das características.

Bagging

As árvores de decisões são muito sensíveis aos dados em que são treinadas. Pequenas alterações no conjunto de treinamento podem resultar em estruturas de árvores significativamente diferentes. A *Random Forest* tira vantagem disso, permitindo que cada árvore, individualmente, faça uma amostragem aleatória do conjunto de dados com substituição, resultando em árvores diferentes.

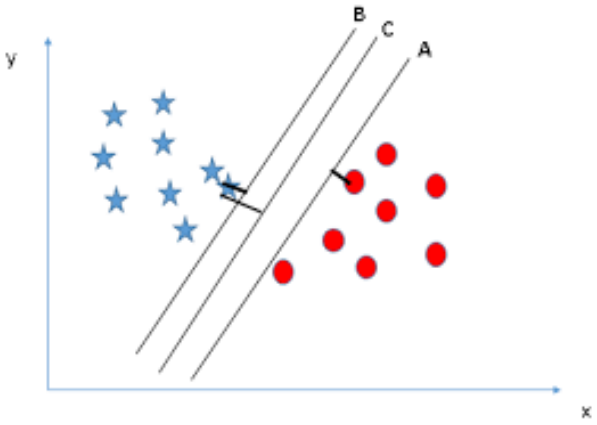
Aleatoriedade das características

Em uma árvore de decisão, quando é hora de dividir um nó, consideramos todos os recursos possíveis e escolhemos aquele que produz a maior separação entre as observações no nó esquerdo e as do nó direito. Por outro lado, cada árvore na *Random Forest* pode selecionar apenas um subconjunto aleatório de características. Isso força ainda mais variação entre as árvores no modelo e, finalmente, resulta em menor correlação entre as árvores e mais diversificação.

2.2.2 Support vector machine(SVM)

O objetivo do SVM (BOSER; GUYON; VAPNIK, 1992) é encontrar um hiperplano em um espaço N-dimensional (N - o número de características) que classifica distintamente os pontos de dados.

Figura 11 – Support vector machine(SVM)



Fonte: Kmr, Gautam (2020). Support Vector Machine — Insights [Blog post]. Retirado de https://towardsai.net/p/machine-learning/support-vector-machine-an-insight-cdae000758dc. Acessado em 16/11/2020

Figura 12 – Support vector machine(SVM)

Existem muitos hiperplanos possíveis que podem ser escolhidos para separar os dados das duas classes acima. Nosso objetivo é encontrar um plano com a margem máxima, ou seja, a distância máxima entre os pontos de dados das duas classes. A maximização da distância da margem fornece algum reforço para que os pontos de dados futuros possam ser classificados com mais confiança. No exemplo acima, o plano escolhido seria o "C".

Hiperplanos são limites de decisão que ajudam a classificar os pontos de dados. Os pontos de dados que caem nos dois lados do hiperplano podem ser atribuídos a diferentes classes. Além disso, a dimensão do hiperplano depende do número de características. Se o número de características de entrada for 2, o hiperplano será apenas uma linha. Se o número de características de entrada for 3, o hiperplano se tornará um plano bidimensional. Torna-se difícil imaginar quando o número de características excede 3.

Como visto anteriormente, quanto maior for a margem entre os pontos de dados e o hiperplano, melhor será a classificação, portanto, o objetivo do SVM é maximizar essa margem. A função de perda que ajuda a maximizar a margem é a função *hinge loss*.

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

Figura 13 – Função de perda

O custo é 0 se o valor previsto e o valor real tiverem o mesmo sinal. Caso contrário, calculamos o valor da perda. É adicionado um parâmetro de regularização à função de custo. O objetivo do parâmetro de regularização é equilibrar a maximização e a perda de margem.

O SVM funciona melhor quando há uma margem clara entre suas classes, isso pode ser observado claramente por conta da sua função de perda. Além disso, o modelo funciona melhor quando o número de dimensões é maior que o número de amostras. Apesar desses pontos positivos, existem algumas desvantagens para o uso de VMS, como o fato do algoritmo não funcionar muito bem para grandes volumes de dados e não performar bem quando o conjunto de dados tem muito ruído (as classes alvos estão sobrepostas).

2.2.3 WiSARD

A *WiSARD* (ALEKSANDER; THOMAS; BOWDEN, 1984) é um tipo de rede neural sem peso e é baseada em um modelo neural que usa tabelas de pesquisa para armazenar a função calculada por cada neurônio, em vez de armazená-la em pesos de conexões neuronais. Este modelo consiste em empregar um decodificador 1-para-n para que a partir de um código **binário**, que no nosso caso representa o nosso texto codificado, seja possível escolher dentre n caminhos diferentes (escolhas possíveis de classes) aquele que melhor represente a classe do texto.

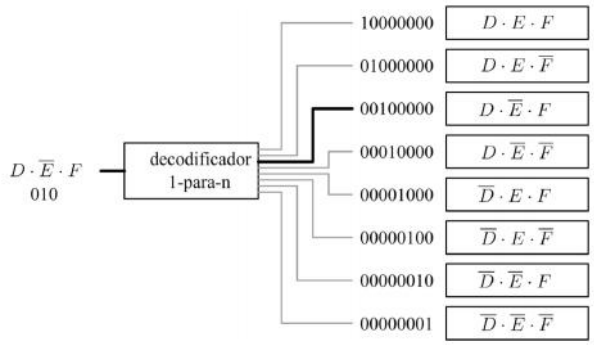


Figura 14 – Decodificador WiSARD.

Fonte: ARAÚJO, L, A. et al. 2011. Acessado em 21/02/2021

O aprendizado da *WiSARD* é feito usando os discriminadores *RAM*, de forma simples, podemos pensar no aprendizado como uma sobreposição das matrizes de treino na

memória. Gerando o que é chamado de imagem mental. Basicamente, é a imagem que o modelo tem da classe treinada.

A WiSARD é formada por vários discriminadores, onde cada um deles é treinado em uma classe de padrões específica. A classificação dos vários discriminadores é feita com todos os discriminadores, gerando uma resposta para o padrão de entrada. As várias respostas são avaliadas por um algoritmo que as comparam e geram uma confiança "c"para cada uma das respostas. A resposta com a maior confiança é escolhida como a classe da entrada.

Esse modelo é bem simples comparado ao SVM e, por ser simples, ele se torna bem rápido em seu treino e classificação, porém, por este motivo, também temos algumas limitações que não tornam esse modelo indicado para todas as aplicações.

Uma das maiores limitações em relação a WiSARD é a sua adaptabilidade em relação a formatação dos dados. Como foi dito, a WiSARD precisa trabalhar com dados binários e dados que não sejam facilmente binarizados, posto que esses podem atrapalhar o desempenho da WiSARD.

Para este trabalho, iremos utilizar uma binarização por limiar, ou seja, dado um vetor numérico não binário, iremos converter os valores deste vetor que sejam maiores que um limiar "l"para 1 e caso contrario iremos converter para 0.

3 CLASSIFICAÇÃO DE TEXTO

3.1 BASES DE DADOS UTILIZADAS

3.1.1 BBC News

Mais de 2000 documentos em inglês retirados do site do BBC News entre 2004 e 2005. (GREENE; CUNNINGHAM, 2006)

O conjunto de dados foi pré-processado utilizando *stemming* (algoritmo de Porter) e teve as *stopwords* removidas. Além disso, esse conjunto foi dividido em 5 classes: negócios, entretenimento, politica, esporte e tecnologia.

Exemplo de alguns textos desse conjunto de dados:

Classe	Texto
Business	"Japan narrowly escapes recession ..."
Entertainment	"Poppins musical gets flying start ..."
Politics	"Labour plans maternity pay rise ..."
Sports	"Dibaba breaks 5,000m world record ..."
Tech	"Microsoft seeking spyware trojan ..."

3.1.2 IMDB Sentiment

Conjunto de dados contendo 5.0000 textos de análises de filmes do IMDB, os textos são divididos em duas classes: positiva e negativa. (MAAS et al., 2011)

Exemplo das classes:

Classe	Texto
Positive	Bromwell High is a cartoon comedy...
Negative	Story of a man who has unnatural feelings for a pig...

3.2 CODIFICAÇÃO

3.2.1 BOW

Para realizar a codificação do texto usando BOW foi utilizado a biblioteca do sklearn em python. O objetivo desse método é realizar a contagem das palavras em um determinado texto e, assim, retornar um vetor do tamanho do vocabulário com a frequência de suas respectivas palavras. Esse vetor resultado que é usado com a codificação do texto.

3.2.2 TF-IDF

Na codificação usando o TF-IDF foi usado o método chamado *TfidfVectorizer* da biblioteca do sklearn onde, conforme explicado no capítulo anterior, calcula-se a frequência inversa das palavras no documento e retorna um vetor com o peso de cada palavra. Esse vetor de peso é o que iremos utilizar no momento da classificação.

3.2.3 Word2Vec

Para utilizar o modelo do *Word2Vec* em inglês, foi utilizado o modelo pré-treinado pela biblioteca do *gensim* (ŘEHŮŘEK; SOJKA, 2010) em *python*. Esse modelo foi pré-treinado em parte do conjunto de dados do *Google News*, em torno de 100 bilhões de palavras. O modelo contém vetores de 300 dimensões para 3 milhões de palavras e frases.

Foi utilizado um modelo pré-treinado pois, além dessa ser uma das vantagem em se trabalhar com codificação de palavras, não seria possível realizar um aprendizado em um conjunto de dados maior.

Após carregar o modelo pré-treinado, a codificação usada nesse trabalho converte todos as palavras do texto para sua representação vetorial obtida pelo *word2vec* e calcula a média dessas palavras, retornando um vetor de 300 dimensões que representa o texto que queremos codificar.

3.3 CLASSIFICADORES

Os classificadores recebem as representações numéricas dos textos e treinam usando uma validação cruzada de *10-folds*. Neste trabalho, utilizamos os classificadores: Random Forest, Suported Vector Machine(SVM) e a WiSARD. Para encontrar os melhores resultados possíveis, foi usado um *Grid-search* para buscar a melhor combinação de parâmetros para os classificadores. Além disso, serão executadas 10 rodadas para coleta das métricas que serão apresentadas na seção de resultados.

3.3.1 Random Forest

Para trabalhar com o *Random Forest* foi utilizada a biblioteca do sklearn. Conforme supracitado, a acurácia do classificador depende de seus hiper-parâmetros. Como o *Random Forest* possui muitos hiper-parâmetros (20), para esse trabalho foi montada várias combinações aleatórias de hiper-parâmetros e escolhemos os melhores hiper-parâmetros para cada codificação.

Os parâmetros combinados foram: **n_estimators**(Número de arvores na floresta), **min_samples_split**(O número mínimo de amostras necessárias para uma divisão interna), **min_samples_leaf**(O Número mínimo de amostrar necessário para ser uma folha), **max_features**(Número de features a serem consideradas antes de realizar uma

divisão), **max_depth**(Tamanho máximo da profundidade das arvores). Esses parâmetros são os que apresentam maior mudança na acurácia quando alterados.

3.3.2 Support vector machine(SVM)

Nesse trabalho, para utilizar o classificador SVM foi usado a biblioteca do sklearn. O classificador possui mais de 15 hiper-parâmetros que podem ser melhorados para aumentar a performance, porém nesse trabalho iremos alterar apenas os parâmetros C (Parâmetro de regularização), *gamma* (Coeficiente de kernel para *'rbf', 'poly' e 'sigmoid'*) e *kernel*(Especifica o tipo de *kernel* a ser usado pelo algoritmo) usando uma busca em grade.

3.3.3 WiSARD

Para utilizar a *WiSARD* em *python* nesse trabalho, foi utilizado a biblioteca *wisardpkg* desenvolvida pelo *lab IA Zero*. A biblioteca foi desenvolvida quase que inteiramente em C++ com interface em *python*. O modelo possui 11 parâmetros, porém nos nossos testes vimos que os parâmetros com valores padrão produziram os melhores resultados. Mais detalhes sobre a biblioteca pode ser encontrado em: https://github.com/IAZero/wisardpkg

4 RESULTADOS EXPERIMENTAIS

Todos os testes foram realizados em uma máquina com as seguintes configurações:

- Windows 10 Pro 64 bits
- Processador I7-9750H
- 16 Gb de memória ram
- GPU NVIDIA GeForce GTX 1650

Para propósito de comparação, colocamos o desempenho dos classificadores com os melhores valores para os parâmetros selecionados (definidos abaixo) e com os valores considerados padrão pela documentação de cada um dos classificadores. Nos gráfico e tabelas a seguir, o desempenho dos classificadores com parâmetros padrões são descritos como XXX_Padrão, sendo XXX o nome do classificador, enquanto os classificadores com os melhores parâmetros são definidos por XXX, sendo XXX o nome do classificador.

4.1 BBC-NEWS

Conjunto de dados com 5 labels e 2225 exemplos

4.1.1 Bag-Of-Words(BOW)

Como foi dito anteriormente, o BOW foi usado para a codificação do texto. Com o texto codificado, foram usados os classificadores. Os melhores parâmetros dos classificadores foram, naturalmente, separados por conjuntos de dados e por método de codificação, sendo assim, os resultados abaixo foram obtidos usando conjunto de dados *BBC-NEWS* e a codificação BOW. Para esta configuração, os parâmetros dos classificadores foram:

4.1.1.1 Melhores parâmetros

4.1.1.1.1 Random Forest

Para este classificador foram usados os seguintes parâmetros.

- n_estimators: 1400
- min_samples_split: 5
- min_samples_leaf: 1

- max_features: 'sqrt'
- max_depth: 80

4.1.1.1.2 SVM

Para este classificador foram usados os seguintes parâmetros.

- C: 0.01
- gamma: 0.001
- kernel: 'linear'

4.1.1.1.3 WiSARD

Conforme foi dito acima, a WiSARD funcionou melhor com os parâmetros padrões da biblioteca, sendo assim, os parâmetros usados nas execuções foram os descritos na documentação do método:

- addressSize: 3
- ignoreZero: False

4.1.1.2 Acurácia

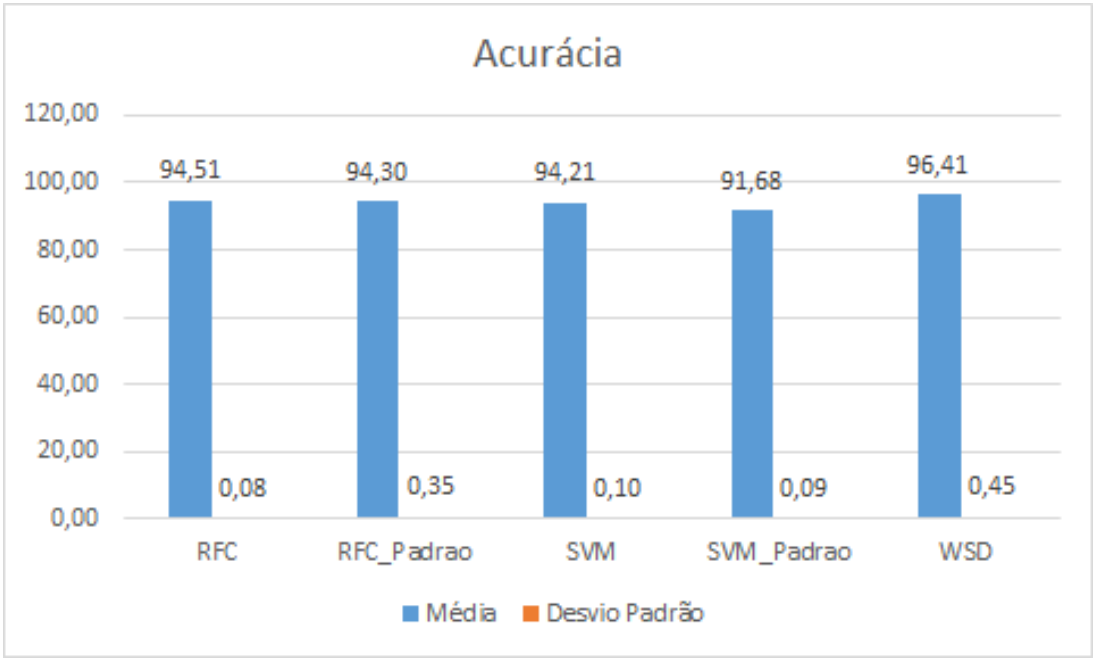


Figura 15 – Acurácia BOW

4.1.1.3 Tempo de execução

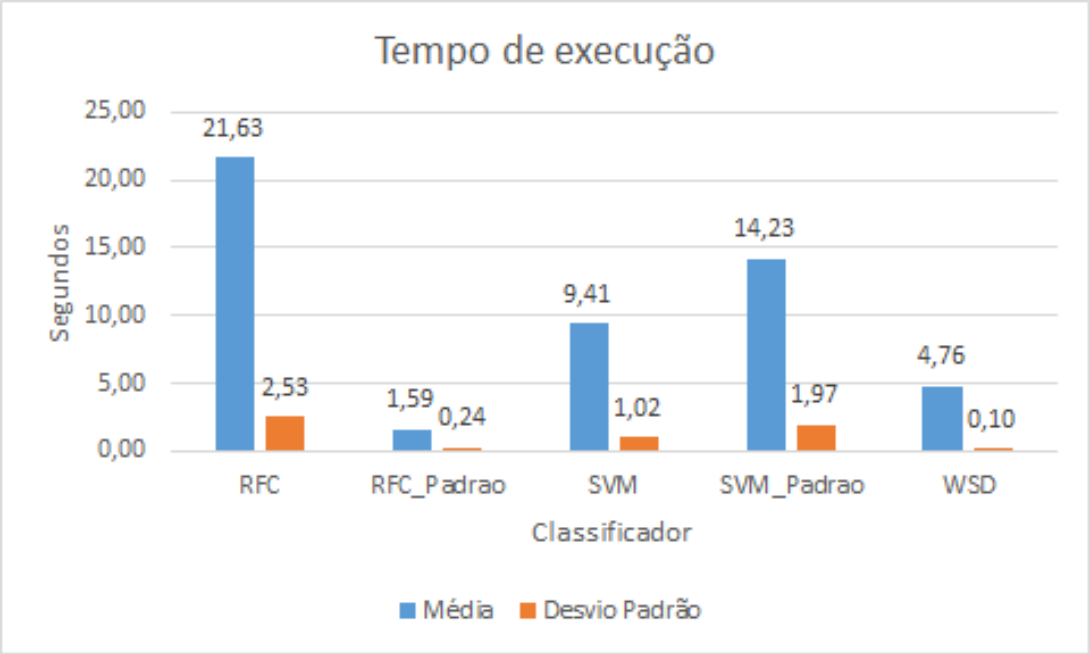


Figura 16 – Tempo de execução BOW

4.1.1.4 Análise

As execuções mostram a média e desvio padrão da acurácia após 10 rodadas. Sendo assim, com exceção da WiSARD, todos os classificadores tiveram uma melhora em performance com a mudança dos parâmetros.

Em geral, como o nosso desvio padrão ficou bem baixo (<0,5), podemos tranquilamente dizer que a WSD obteve o melhor desempenho trabalhando com o BOW.

Quando levamos em consideração o tempo de execução, a WiSARD continua tendo um desempenho melhor, apesar de o *Random Forest* com os parâmetros padrões ter tido um tempo menor que a WiSARD, a diferença na acurácia coloca a WiSARD em vantagem.

4.1.2 TF-IDF

Os resultados abaixo foram obtidos usando conjunto de dados *BBC-NEWS* e a codificação TF-IDF. Para esta configuração, os parâmetros dos classificadores foram:

4.1.2.1 Melhores parâmetros

4.1.2.1.1 Random Forest

Para este classificador foram usados os seguintes parâmetros.

- n_estimators: 1000

- min_samples_split: 10
- min_samples_leaf: 1
- max_features: 'sqrt'
- max_depth: 80

4.1.2.1.2 SVM

Para este classificador foram usados os seguintes parâmetros.

- C: 10
- gamma: 0.001
- kernel: 'linear'

4.1.2.1.3 WiSARD

Conforme foi dito acima, a WiSARD funcionou melhor com os parâmetros padrões da biblioteca, sendo assim, os parâmetros usados nas execuções foram os descritos na documentação do método:

- addressSize: 3
- ignoreZero: False

4.1.2.2 Acurácia

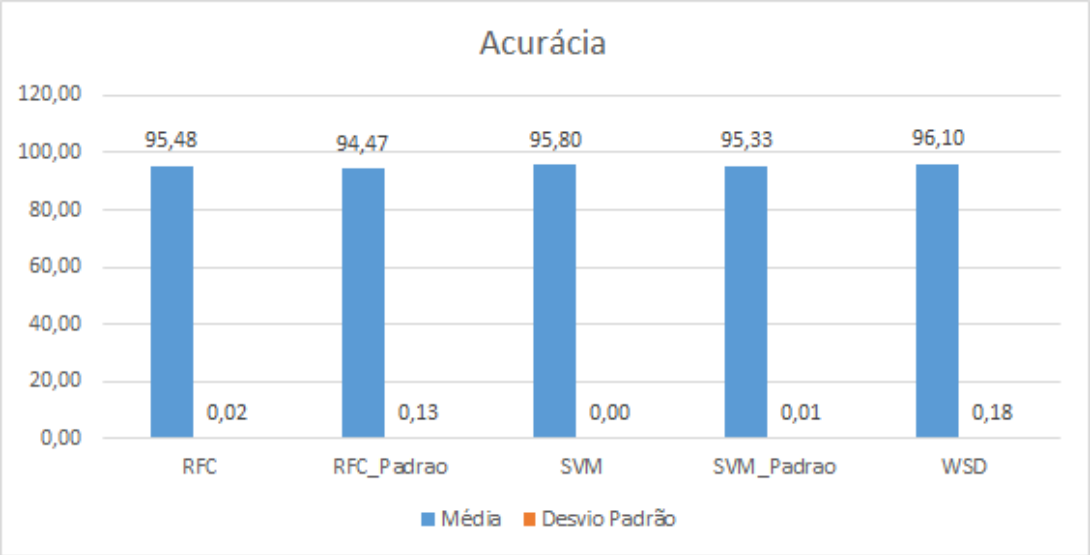


Figura 17 – Acurácia TF-IDF

4.1.2.3 Tempo de execução

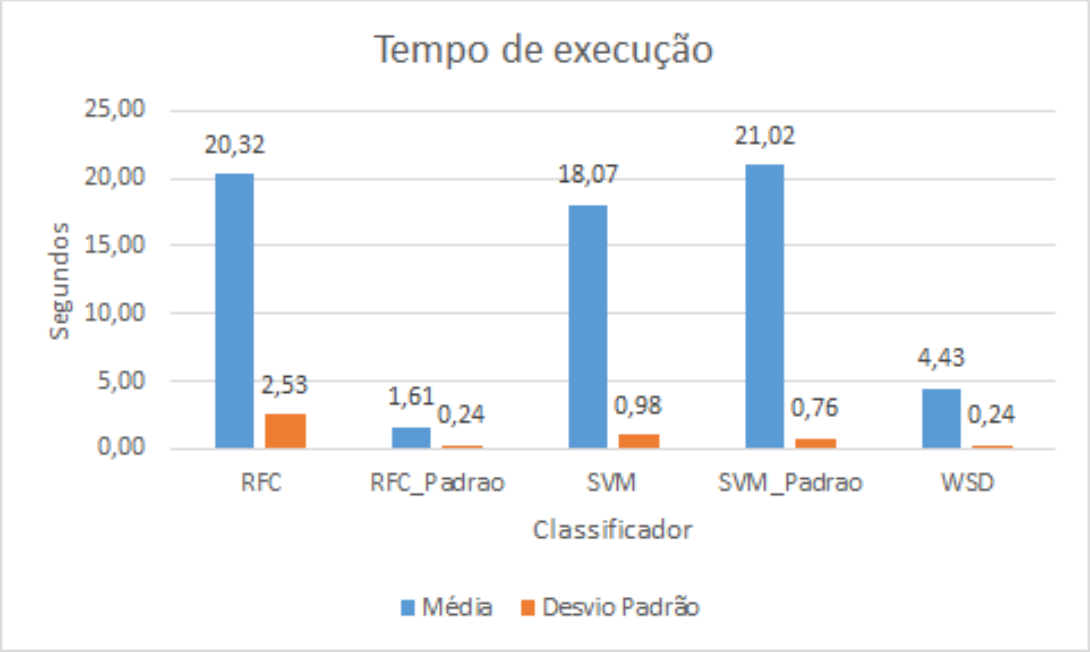


Figura 18 – Tempo de execução TF-IDF

4.1.2.4 Analise

Os resultados foram consistentes com os encontrados na seção 4.1.1; o desvio padrão continua baixo e a acurácia da WiSARD se mantém com a melhor acurácia e com a melhor combinação acurácia/tempo. É esperado que estes resultados sejam parecidos, pois os dois métodos de codificação são parecidos entre si.

4.1.3 Word2Vec

Os resultados abaixo foram obtidos usando conjunto de dados *BBC-NEWS* e a codificação TF-IDF. Para esta configuração, os parâmetros dos classificadores foram:

4.1.3.1 Melhores parâmetros

4.1.3.1.1 Random Forest

Para este classificador foram usados os seguintes parâmetros.

- n_estimators: 600
- min_samples_split: 5
- min_samples_leaf: 2
- max_features: 'auto'

- max_depth: 70

4.1.3.1.2 SVM

Para este classificador foram usados os seguintes parâmetros.

- C: 10
- gamma: 1
- kernel: 'rbf'

4.1.3.1.3 WiSARD

Conforme foi dito acima, a WiSARD funcionou melhor com os parâmetros padrões da biblioteca, sendo assim, os parâmetros usados nas execuções foram os descritos na documentação do método:

- addressSize: 3
- ignoreZero: False

4.1.3.2 Acurácia

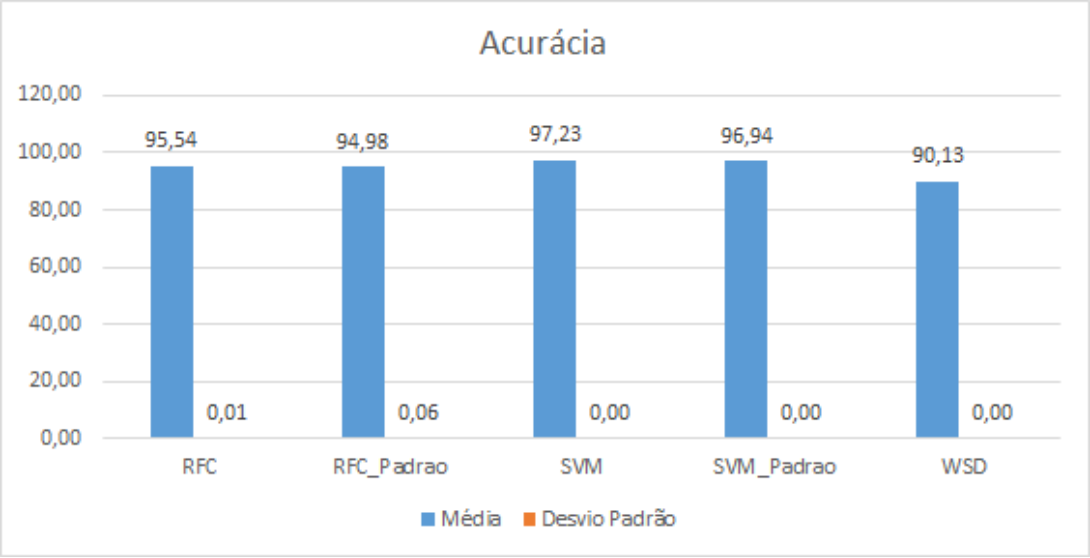


Figura 19 – Acurácia Word2Vec

4.1.3.3 Tempo de execução

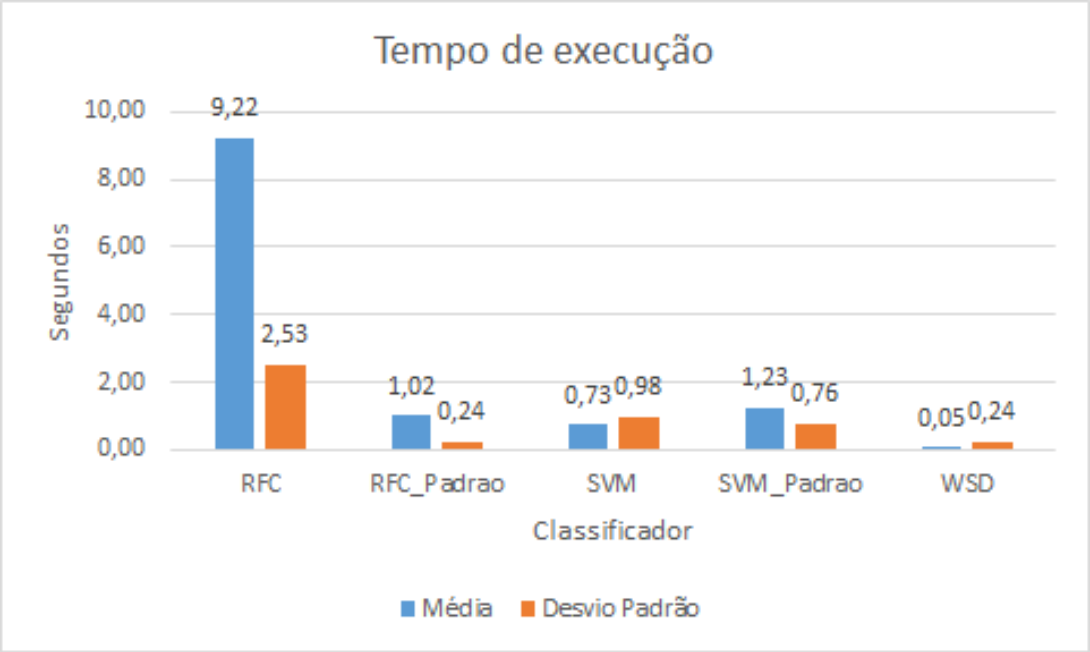


Figura 20 – Tempo Word2Vec

4.1.3.4 Analise

O resultado final da codificação feita pelo W2V é bastante diferente do resultado final do BOW e TF-IDF, com isso, era esperado uma mudança maior nos resultados. Dito isso, podemos ver que o desempenho da WiSARD sofre com esse tipo de codificação, porém o desempenho geral do nosso modelo de classificação foi melhor, sendo possível alcançar uma acurácia maior do que os outros tipos de codificação.

4.2 IMDB

Conjunto de dados para análise de sentimento com 2 *labels* e 5.0000 exemplos. Como estamos trabalhando com um volume de dados muito extenso, foi preciso alterar a forma com que trabalhamos com esses dados. Para usar o SVM e o *Random Forest* foi preciso trabalhar somente usando a biblioteca do Pandas para *python*, porém a *WiSARD* não trabalha com dados nesse formato, portanto, nesse caso usamos uma particularidade da *WiSARD* para treinar o modelo em blocos pequenos de dados, isto é, dividido o conjunto de dados em pequenos blocos de no máximo 2000 exemplos cada um e treinamos a *WiSARD* assim.

4.2.1 Bag-Of-Words(BOW)

Os parâmetros que apresentaram melhores resultados para o conjunto de dados do IMDB foram muito parecidos com os parâmetros usados no *BBC-NEWS*, isso se deve principalmente ao fato de que os parâmetros dos classificadores determinam como as ferramentas de classificação trabalham com os formatos dos dados e não com a quantidade de dados, isto é, os parâmetros determinam, pro exemplo, como o SVM vai trabalhar com o texto codificado pelo BOW. O resultado final das codificações pelo BOW e pelo TF-IDF são parecidos, por isso os seus melhores parâmetros encontrados são semelhantes. Em comparação, a codificação pelo W2V é completamente diferente das outras duas codificações, e isso faz com que os melhores parâmetros encontrados para se trabalhar com ele seja completamente diferente dos parâmetros encontrados para se trabalhar com o BOW e TF-IDF.

4.2.1.1 Melhores parâmetros

4.2.1.1.1 Random Forest

Para este classificador foram usados os seguintes parâmetros.

- n_estimators: 1000
- min_samples_split: 5
- min_samples_leaf: 1
- max_features: 'sqrt'
- max_depth: 80

4.2.1.1.2 SVM

Para este classificador foram usados os seguintes parâmetros.

- C: 0.01
- gamma: 0.01
- kernel: 'linear'

4.2.1.1.3 WiSARD

Conforme foi dito acima, a WiSARD funcionou melhor com os parâmetros padrões da biblioteca, sendo assim, os parâmetros usados nas execuções foram os descritos na documentação do método:

- addressSize: 3
- ignoreZero: False

4.2.1.2 Acurácia

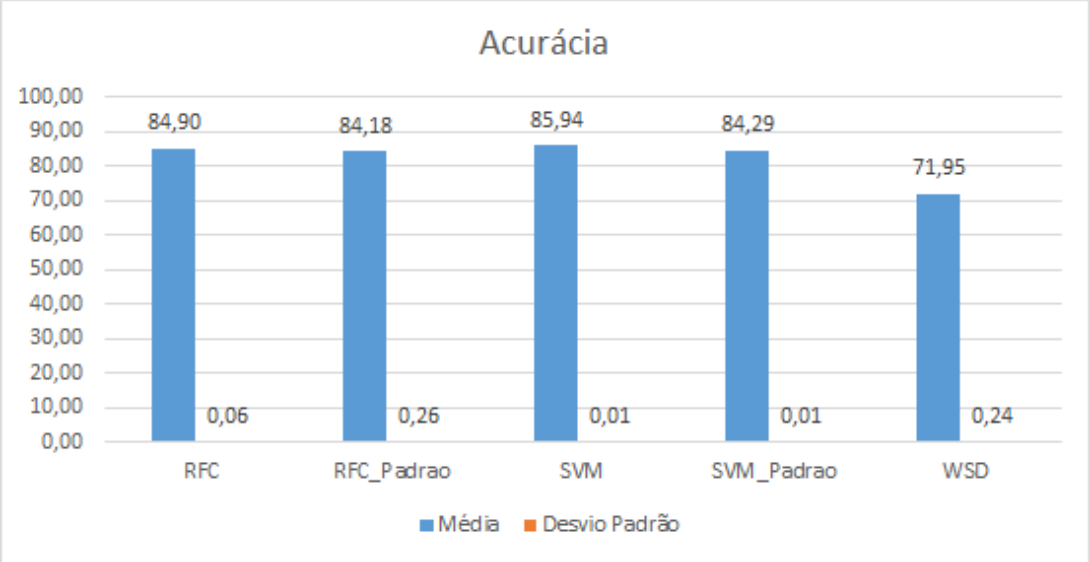


Figura 21 – Acurácia BOW

4.2.1.3 Tempo de execução

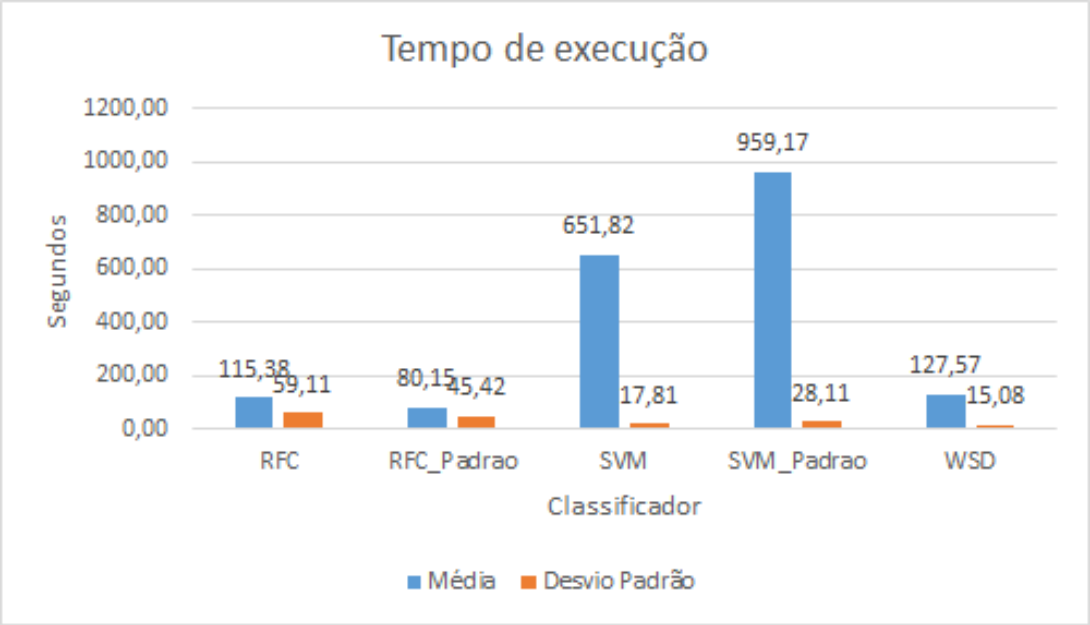


Figura 22 – Tempo de execução BOW

4.2.1.4 Analise

Com um conjunto de dados consideravelmente maior que o primeiro apresentado, os tempos de execução dispararam, resultado bastante esperado. Dito isso, a acurácia do *Random Forest* e do SVM foram bem parecidos, mas como os tempos do *Random Forest* foram menores, o conjunto acurácia/tempo do *Random Forest* se mostra a melhor escolha para a configuração apresentada.

4.2.2 TF-IDF

4.2.2.1 Melhores parâmetros

4.2.2.1.1 SVM

- n_estimators: 1000
- min_samples_split: 5
- min_samples_leaf: 1
- max_features: 'sqrt'
- max_depth: 80

4.2.2.1.2 SVM

Para este classificador foram usados os seguintes parâmetros.

- C: 10
- gamma: 0.001
- kernel: 'linear'

4.2.2.1.3 WiSARD

Conforme foi dito acima, a WiSARD funcionou melhor com os parâmetros padrões da biblioteca, sendo assim, os parâmetros usados nas execuções foram os descritos na documentação do método:

- addressSize: 3
- ignoreZero: False

4.2.2.2 Acurácia

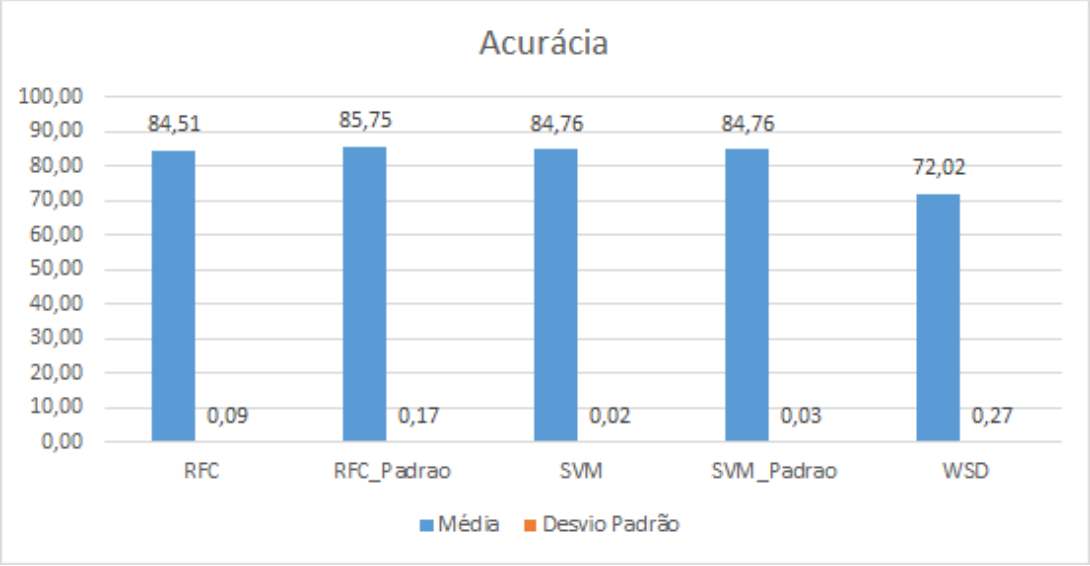


Figura 23 – Acurácia TF-IDF

4.2.2.3 Tempo de execução

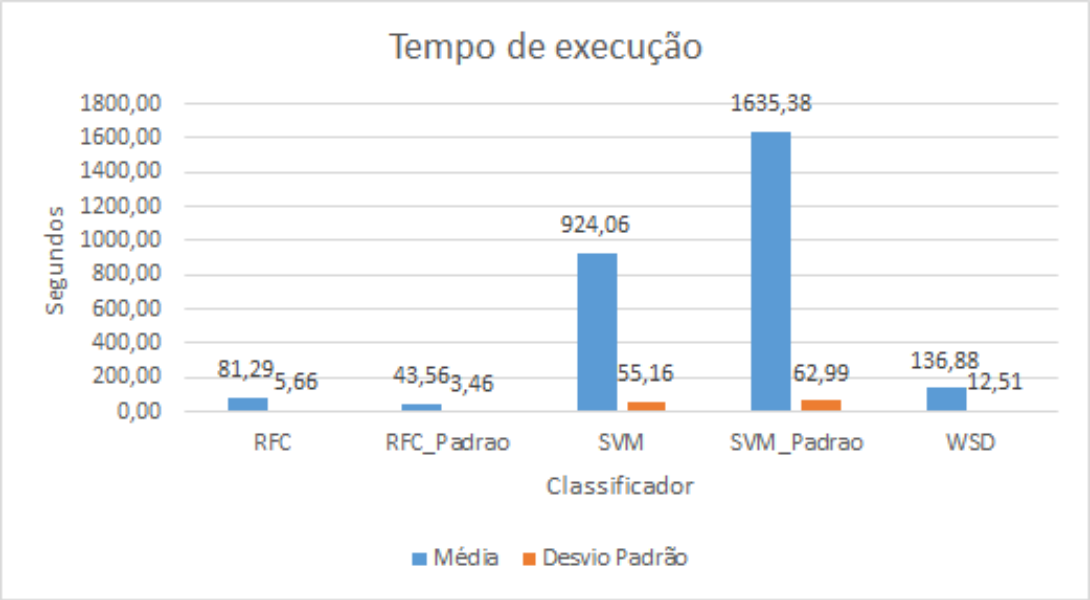


Figura 24 – Tempo de execução TF-IDF

4.2.2.4 Analise

Os resultados foram similares com os encontrados na seção 4.1 no sentido de que os resultados entre o BOW e TF-IDF são bem parecidos, pois os resultados finais dos modelos são bem semelhantes entre si. Sendo assim, vemos uma leve vantagem em números absolutos para o SVM, porém quando consideramos o conjunto acurácia/tempo o *Random Forest* se destaca.

4.2.3 Word2Vec

Como a biblioteca do *gensim* que foi utilizada para trabalhar com o *Word2Vec* não trabalha bem com a estrutura do Pandas que montamos, não foi possível executar o *Word2Vec* para o *Random Forest* e SVM, pois esses modelos não permitem que o aprendizado seja feito de forma dividida, como a *WiSARD*.

Como não foi possível executar o W2V com o *Random Forest* e SVM, não foi possível encontrar os melhores parâmetros para a execução, porém observando os resultados obtidos é possível argumentar que os parâmetros sejam parecidos com os encontrados na seção 4.1.3 deste trabalho.

4.2.3.1 Melhores parâmetros

4.2.3.1.1 WiSARD

Por conta da estrutura da WiSARD foi possível executar o conjunto de dados e os melhores parâmetros foram novamente os parâmetros padrões da biblioteca:

- addressSize: 3
- ignoreZero: False

4.2.3.2 Acurácia

Figura 25 – Acurácia Word2Vec

4.2.3.3 Tempo de execução

Figura 26 – Tempo Word2Vec

4.2.3.4 Analise

Como o processo de codificação do texto via W2V é bastante complexo e o conjunto de dados bastante extenso, a máquina usada não executou o modelo para todos os classificadores. Entretanto, por conta da estrutura da WiSARD, foi possível dividir o treinamento em etapas e treinar todo o conjunto de dados com facilidade. Por conta de não conseguir executar o conjunto de dados para todos os classificadores, não é possível realizar uma comparação entre os classificadores. Dito isso, acredita-se que o fato de a WiSARD ter conseguido executar o conjunto de dados é uma grande vantagem deste classificador, pois, um classificador que tem um desempenho ruim em acurácia, ainda é melhor do que um classificador que não consegue executar.

4.2.4 Compilado final

4.2.4.1 BBC News

Acurácia (%)						
	BOW - Default	BOW	TF-IDF - Default	TF-IDF	W2V - Default	W2V
Random Forest	94,30	94,51	94,46	95,48	94,98	95,54
SVM	91,68	94,20	95,33	95,79	96,94	97,23
WiSARD	96,41	96,41	96,09	96,09	90,13	90,13

Figura 27 – Acurácia BBC News

Tempo (s)			
	BOW	TF-IDF	W2V
Random Forest	21,63	20,32	9,22
SVM	9,41	18,07	0,73
WiSARD	4,76	4,43	0,05

Figura 28 – Tempo BBC News

4.2.4.2 IMDB

Acurácia (%)						
	BOW - Default	BOW	TF-IDF - Default	TF-IDF	W2V - Default	W2V
Random Forest	84,18	84,90	85,75	84,51	--	--
SVM	85,29	85,94	84,76	84,76	--	--
WiSARD	71,95	71,95	72,02	72,02	56,94	56,94

Figura 29 – Acurácia IMDB

Tempo (s)			
	BOW	TF-IDF	W2V
Random Forest	115,38	43,56	--
SVM	80,15	924,06	--
WiSARD	127,57	136,88	2

Figura 30 – Tempo IMDB

5 CONCLUSÃO

Neste trabalho, foram comparadas 3 formas de binarização e 3 classificadores, portanto a conclusão deve ser separada em duas partes: primeiro vamos comparar os classificadores, em seguida abordaremos os codificadores. Essa conclusão deve ser feita dessa forma pois os classificadores vistos podem ser usados em conjunto com outros codificadores e os codificadores vistos podem ser usados com outro classificadores vice-versa.

Primeiro vamos falar dos classificadores. Como vimos, a *WiSARD* trabalha com dados numéricos, portanto, quando estamos trabalhando com um conjunto de dados estritamente neste formato, esse modelo se sai melhor do que os outros comparados (SVM e *Random Forest*), como é o caso da codificação do texto feita usando BOW e TF-IDF. Podemos, então, traçar uma linha e mostrar que quando a binarização dos dados não possui influência, a *WiSARD* se sai melhor que os outros modelos de classificação mostrados.

Quando trabalhamos com dados que precisam de uma binarização, que é o caso quando usamos a codificação do *Word2Vec*, a *WiSARD* possui a pior acurácia. Dado que, em dados que não requerem binarização, a *WiSARD* tem uma performance muito boa, podemos assumir que a binarização que utilizamos no *Word2Vec* foi o que atrapalhou o modelo e isso não é estranho, pois a binarização por limiar, que foi a usada, perde muita informação, sendo assim podemos dizer que os modelos de SVM e *Random Forest* classificaram os textos com muito mais informação do que a *WiSARD* nesse caso.

Classificadores não podem ser definidos apenas por acurácia, pois não temos recurso ilimitado para treinar/classificar e talvez seja necessário usar esses modelos embarcados, portanto, devemos levar em consideração o tempo de execução de cada modelo. O tempo medido e apresentado nos gráficos são os tempos necessários para os modelos realizarem o treinamento e teste.

Como era de se esperar, a *WiSARD* é extremamente rápida, seguida pelo *Random Forest*, Enquanto o modelo mais lento foi o SVM. É a grande vantagem da *WiSARD* ser tão rápida, a sua estrutura permite que esse modelo possa ser embarcado. O SVM é um modelo extremamente poderoso, por isso é o mais lento dos modelos testados, portanto, apesar de uma boa acurácia, não é um bom modelo para usarmos com classificação de texto, pois a acurácia não foi muito superior aos outros modelos, tendo-se mostrado relevantemente lento em sua execução.

Agora, falando sobre codificadores, vimos que o BOW e TF-IDF são parecidos em suas ideias e, por isso, as acurácias e tempos de execução são bem parecidos e suas aplicações também são bem parecidas. Em geral, podemos usar esses modelos em problemas de classificação simples, onde não precisamos de tanta informação acerca do texto. Nos nossos exemplos, as classes são facilmente separadas e, portanto, não precisamos de tantas informações sobre as palavras do texto. No caso, essas informações podem atrapalhar a

classificação, semelhante ao *over-fitting*. O *Word2Vec* em si, é um codificador muito mais poderoso, pois temos informações sobre as palavras, como o contexto, por exemplo.

O maior problema sobre o *Word2Vec* no nosso modelo é que apesar de possuirmos as informações sobre as palavras, quando tentamos consolidar essas informações temos perdas. Essas perdas afetam bastante a eficácia do *Word2vec* e acredita-se que isso seja o motivo de o *Word2Vec* ter um desempenho ruim nos testes.

Para trabalhos futuros, devemos focar em dois pontos: melhorar a binarização para a *WiSARD* e melhorar a consolidação das informações das palavras para textos do *Word2Vec*. O motivo de focar nessas melhorias em trabalhos futuros é pelo fato da *WiSARD* ter se mostrado como o modelo mais eficiente e melhorar a acurácia dele, principalmente para codificadores que necessitem de binarização, sendo mais fácil do que melhorar a eficiência dos outros modelos. Além disso, como vimos que o modelo do *Word2Vec* é mais poderoso, é melhor atacar os problemas identificados e sair com um modelo mais robusto e poderoso.

REFERÊNCIAS

ALEKSANDER, I.; THOMAS, W.; BOWDEN, P. Wisard· a radical step forward in image recognition. **Sensor review**, MCB UP Ltd, 1984.

BOSER, B. E.; GUYON, I. M.; VAPNIK, V. N. A training algorithm for optimal margin classifiers. In: **Proceedings of the fifth annual workshop on Computational learning theory**. [S.l.: s.n.], 1992. p. 144–152.

BREIMAN, L. Random forests. **Machine learning**, Springer, v. 45, n. 1, p. 5–32, 2001.

GREENE, D.; CUNNINGHAM, P. Practical solutions to the problem of diagonal dominance in kernel document clustering. In: **Proc. 23rd International Conference on Machine learning (ICML'06)**. [S.l.]: ACM Press, 2006. p. 377–384.

HARRIS, Z. S. Distributional structure. **Word**, Taylor & Francis, v. 10, n. 2-3, p. 146–162, 1954.

JONES, K. S. A statistical interpretation of term specificity and its application in retrieval. **Journal of documentation**, MCB UP Ltd, 1972.

MAAS, A. L. et al. Learning word vectors for sentiment analysis. In: **Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies**. Portland, Oregon, USA: Association for Computational Linguistics, 2011. p. 142–150. Disponível em: <http://www.aclweb.org/anthology/P11-1015>.

MIKOLOV, T. et al. **Efficient Estimation of Word Representations in Vector Space**. 2013. Disponível em: <http://arxiv.org/abs/1301.3781>.

ŘEHŮŘEK, R.; SOJKA, P. Software Framework for Topic Modelling with Large Corpora. In: **Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks**. Valletta, Malta: ELRA, 2010. p. 45–50. <http://is.muni.cz/publication/884893/en>.