Modelo de Análise Opiniões Híbrido para um Sistema de Ensino Inteligente

Rodolfo Silva[A81716] Bruno Veloso[A78352]
Rafael Lourenço[A86266]

Universidade do Minho, Departamento de Informática

Orientadores: Orlando Belo & Jefferson Magalhães

Resumo

O presente relatório tem como objetivo o desenvolvimento de um modelo híbrido para um sistema de ensino inteligente, capaz de classificar opiniões em 3 classes distintas, sendo estas negativo, neutro e positivo. Um modelo híbrido consiste na junção de dois classificadores, um com base num dicionário léxico (o *VADER*), e o outro com base num classificador de *machine learning* ou *deep learning*. Para isto foi necessário a realização de um bom balanceamento do *dataset*, um pré-processamento eficiente e por último um método eficaz para a união de resultados. Por fim, foram realizados vários testes aos possíveis modelos híbridos, chegando à conclusão que o melhor modelo híbrido obtido foi a junção do *VADER* com uma rede neuronal, que tirava partido de *word embeding*, alcançando uma precisão de 90%.

Keywords: *Machine Learning*, *Deep Learning*, *Modelo Híbrido*, *Word Embeding*

Conteúdo

Modelo de Análise Opiniões Híbrido para um Sistema de Ensino Inteligente 1
Rodolfo Silva Bruno Veloso Rafael Lourenço
1 Introdução 3
1.1 Contextualização 3
1.2 Estrutura do relatório 3
2 *Dataset* 4
2.1 Constituição do *dataset* 4
2.2 Transformação e Análise do *dataset* 4
3 Pré-processamento 5
3.1 Remoção de HTML 5
3.2 Correção de erros gramaticais 6
3.3 Separação de palavras e frases 6
3.4 Tokenização da frase 6
3.5 Remoção de *stop words* 6
3.6 Reestruturação dos dados 7
3.7 Balanceamento do *dataset* 7
4 Machine Learning 9
4.1 Logistic Regression 9
4.2 Naive Bayes 10
4.3 Random Forest 11
4.4 Support Vector Machines 12
4.5 XGBoost 13
4.6 One Class Classification 14
4.7 Análise de resultados 15
5 Deep Learning 16
5.1 CNN 17
5.2 RNN 18
5.3 CRNN 21
6 Modelo Híbrido 23
6.1 Resultados obtidos com Modelo Híbrido 24
7 Conclusão e Trabalho futuro 26

1 Introdução

Este relatório é referente ao trabalho de grupo proposto na Unidade Curricular de Laboratórios em Engenharia Informática. O projeto consiste em conceber um modelo híbrido capaz de classificar opiniões de estudantes, em relação ao ensino, em três classes distintas, sendo estas positivo, negativo e neutro. Um modelo híbrido consiste em juntar os *outputs* de um modelo de *machine learning* ou *deep learning* com um modelo de dicionário léxico, a fim de obter um melhor acerto na classificação das opiniões dos estudantes.

Numa primeira fase do relatório, será abordado o *dataset* que se usou, bem como o pré-processamento realizado numa fase de preparação para os modelos de classificação.

De seguida, falar-se-á dos modelos de *machine learning* e *deep learning* desenvolvidos, assim como os seus resultados.

Por fim, será explicado todo o modelo híbrido implementado e quais os constituintes do mesmo, em que este apresentará melhores resultados nestas classificação.

1.1 Contextualização

Ao longo do tempo, tem se verificado um aumento do uso de *machine learning* e *deep learning* para classificação de determinadas coisas, para que este seja feita pela máquina e sem qualquer intervenção humana, tornando este processo o mais automático possível.

O ensino é uma das áreas que cada vez mais está a recorrer a estes processos. Por vezes, é impossível verificar comentário a comentário sobre o ensino, pelo que requer um sistema que o faça automaticamente, na tentativa de haver uma melhoria no ensino e saber quais os pontos a focar. Posto isto, neste trabalho pretende-se a construção de um modelo híbrido de classificação de opiniões de estudante em relação ao ensino, cujo objetivo é ter o maior acerto possível nesta classificação, dando assim uma maior perspetiva do que as opiniões representam (se negativo ou não, etc.) e, assim, podendo melhorar o ensino com base na informação presente nestas opiniões.

1.2 Estrutura do relatório

O relatório encontra-se estruturado da seguinte forma:

Inicialmente, na *secção 1* será feita uma breve introdução e contextualização do problema do caso em estudo. Na *secção 2*, será abordado qual o *dataset* usado e na *secção 3* todo o pré-processamento executado sobre as opiniões deste, de forma a preparar o mesmo para *machine learning* e *deep learning*. Seguindo de uma *secção 4* em que se falará sobre todos os modelos de *machine learning* aplicados sobre o dados na tentativa de classificação de opiniões, seguido de uma *secção 5* em que se falará de todo o *deep learning* implementado. Já perto do fim, uma *secção 6*, que abordará em que consiste o modelo híbrido implementado, seguindo de uma conclusão e análise crítica presente na *secção 7*.

2 *Dataset*

Um bom *dataset* é essencial para conseguir treinar modelos, tanto de *machine learning* como *deep learning*.

Uma vez que o *dataset* disponibilizado não continha entradas suficientes para o treino de modelos inteligentes, foi necessário procurar um *dataset* que melhor se adequasse a este projeto a desenvolver. Posto isto, o grupo encontrou um *dataset*[2] do *Kaggle*, criado a partir de um *scrapping* feito sobre o *website Coursera*, sendo que cada entrada deste representa um comentário a um determinado curso do *Coursera*. O grupo considerou este *dataset* o mais apropriado a usar, uma vez que é o mais próximo da área onde o modelo híbrido será usado.

2.1 Constituição do *dataset*

O *dataset* previamente referido é constituído por 107018 entradas e três colunas. São estas: *Id*, que representa o identificador do comentário; *Review*, que é o comentário em si, e *Label*, representando o número de estrelas dadas pelo utilizador, correspondendo a um valor num intervalo de 1 a 5.

2.2 Transformação e Análise do *dataset*

Uma vez que este *dataset* continha opiniões que não eram ingleses, o grupo considerou que a melhor abordagem estaria na remoção destes, reduzindo o número de entradas de 107018 para 105436. Para além disso, como o caso de estudo só engloba três classes (negativo, neutro e positivo), foram convertidos os valores presentes na coluna *Label* (agora denominada *polarityClass*). Assim sendo, todos os valores 1 e 2 da coluna *Label* foram convertidos para *0* (que representa negativo); todos os valores 3 foram substituídos por *1* (que representa neutro) e, por fim, todos os valores 4 e 5 foram passados a *2* (que representa positivo).

Após esta alteração da coluna *Label*, foi possível ver a grande disparidade entre as três classes distintas, sendo que a classe 0 (negativo) tinha 4642 entradas, a classe 1 (neutro) tinha 5011 entradas, e a classe 2 (positivo) tinha 95783 entradas, como se pode verificar no gráfico 2.2.

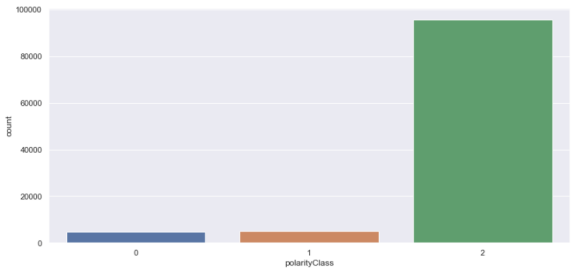

Figura 1. Distribuição das classes

3 Pré-processamento

O *dataset* utilizado para estudo é referente a todas as avaliações de cursos presentes no *website Coursera*. Por este motivo, o *dataset* possui alguns defeitos que inibiam a aprendizagem por parte do algoritmo. Assim sendo, foi feito um *script python* que visa limpar os dados, de forma a permitir o seu estudo. A estrutura do *script* é apresentada na Figura 2:

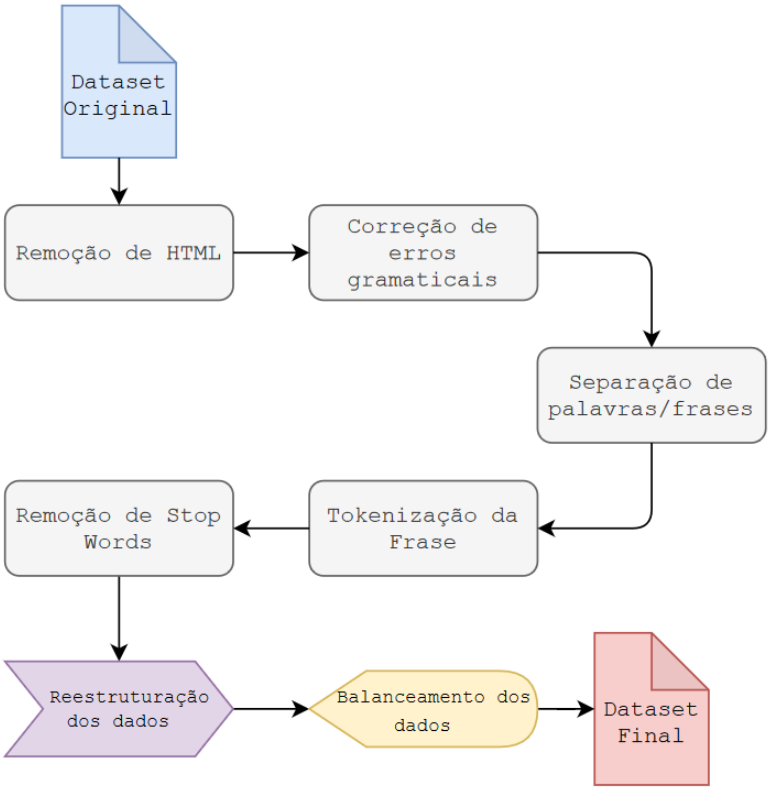

Figura 2. Distribuição das classes

3.1 Remoção de HTML

Foram removidas as *tags html* que podiam estar presentes no texto. Para tal, recorreu-se à biblioteca *Beautiful Soup* que remove automaticamente as *tags*.

3.2 Correção de erros gramaticais

Para a correção de erros ortográficos e procura de palavras idênticas, foi usada a biblioteca SymSpell, que utiliza um algoritmo para encontrar todas as *strings* numa certa distância de edição máxima (*Restricted Damerau-Levenshtein*) dentro de um dicionário indexado, que torna essa procura muito rápida, tornando o pré-processameno também mais rápido. Informalmente, a distância *Damerau-Levenshtein* entre duas palavras é o número mínimo de operações (que consistem em inserir, remover ou substituir de um único caractere, transposição de dois caracteres adjacentes) necessárias para alterar uma palavra. Outra particularidade deste algoritmo é que torna possível a conversão da frase para letras minúsculas. Apresenta-se, de seguida, um exemplo deste algoritmo:

- **Input**: "This ubject wa s quitte asdfaasdfjnjoawsdf confusinng, whic h Slows downthe learning."
- **Output**: "this subject was quite confusing which slows down the learning."

3.3 Separação de palavras e frases

Podia haver situações onde as frases ou palavras não se encontravam perfeitamente separadas, pelo que se decidiu fazer uma separação baseada num autómato. Como exemplo, têm-se as seguintes opiniões:

- This type of question is not **good.It** should be changed.
- Underst**oodWit**hout a doubt.

O autómato usado para resolver estes tipos de problemas é simples. Todas as letras maiúsculas são identificadas por **X**, as minúsculas por **x** e a pontuação por um ponto **.**. Assim, vão criar-se os estados do autómato, por exemplo: **Xx.**, **xX.**, **.xX-** e, caso a palavra não siga uma das estruturas léxicas do autómato, um espaço vai ser adicionado para separar as palavras.

3.4 Tokenização da frase

A frase anterior era necessária para a facilitação desta tarefa, onde se tem que separar uma frase num *array*. Visto que se trata de um modelo de *Machine Learning*, este *array* vai corresponder à lista de argumentos que representam o comentário. Para esta tarefa, foi usada a biblioteca *nltk* que permite fazer este processo com uma das suas funções, *word_tokenize()*.

3.5 Remoção de *stop words*

Stop words são palavras que são consideradas irrelevantes no que diz respeito à análise de sentimentos. Quer sejam conjunções ou mesmo palavras com apenas uma letra, nunca vão expressar qualquer tipo de sentimento e, por isso, devem ser removidas para tornar o processo de classificação mais eficiente. As palavras escolhidas para *stop words* são da bibilioteca *nitk corpus*. Alguns exemplos seriam: *but*, *again*, *s*, *am*, *or* e *L*.

3.6 Reestruturação dos dados

Como parte final do pré-processamento, foi realizada uma reestruturação geral dos dados para se poder criar um *dataset* a partir do qual um modelo podia aprender.

Começou por se criar um dicionário com todas as palavras que ocorrem mais do que duas vezes em todas as opiniões. Para cada opinião, atribuí-se o valor de 0 ocorrências para cada palavra individual do dicionário, e depois percorre-se a versão *tokenizada* da opinião, incrementando-se o valor das palavras que nela aparecem, de forma a criar um *dataset* com a soma das ocorrências de palavras por opinião. Como por exemplo:

```
1  Dicionário = ['low', 'bad', 'awfull', 'good', 'service', 'very']
2
3  Frase processada = ['service', 'very', 'very', 'bad']
4
5  Linha representante da critica no dataset = [0,1,0,0,0,1,2]
```

Esta estruturação efetuada no *dataset* permite a um modelo aprender quais as combinações de palavras que correspondem a opiniões positivas ou negativas.

Ainda mais, os *datasets* gerados foram guardados com o formato $.ftr(feather)$, este formato é extremamente eficiente em tamanho, tempos de leitura e tempos de escrita.

3.7 Balanceamento do *dataset*

Esta fase do pré-processamento é umas das mais importantes deste projeto, visto que foi uma das partes que mais influenciou os resultados obtidos pelos modelos.

Inicialmente, utilizou-se apenas o *SMOTE*, uma técnica de *oversampling* das classes minoritárias. No entanto, os modelos continuavam com bastante dificuldade a classificar as classes minoritárias do *dataset* (Negativo e Neutro).

Devido aos maus resultados apresentados pelos modelos, foi aplicado *RandomOverSampler*[6] às classes minoritárias e *RandomUnderSampler*[6] à classe maioritária, o que trouxe uma melhoria significativa em quase todas as métricas, gerados pelo *classification_report* da biblioteca do *sklearn*, em todos os modelos criados, tanto de *machine learning* como *deep learning*.

Por último, foi ainda testada uma técnica de *undersampling* que tira partido dos *Tomek links*[6]. Um *Tomek link* acontece quando duas opiniões/*samples* pertencem a classes diferentes, todavia são os vizinhos mais próximos um do outro.[6] Esta técnica consiste em remover apenas essas amostras da maioria das classes, numa tentativa de isolar cada uma da classe. No entanto, não se obteve melhorias significativas em relação ao *RandomUnderSampler* após *RandomOverSampler*, logo manteve-se este método.

Posto isto, o *dataset* foi testado para várias distribuições de classes, sendo que algumas delas vão ser apresentadas na imagem abaixo :

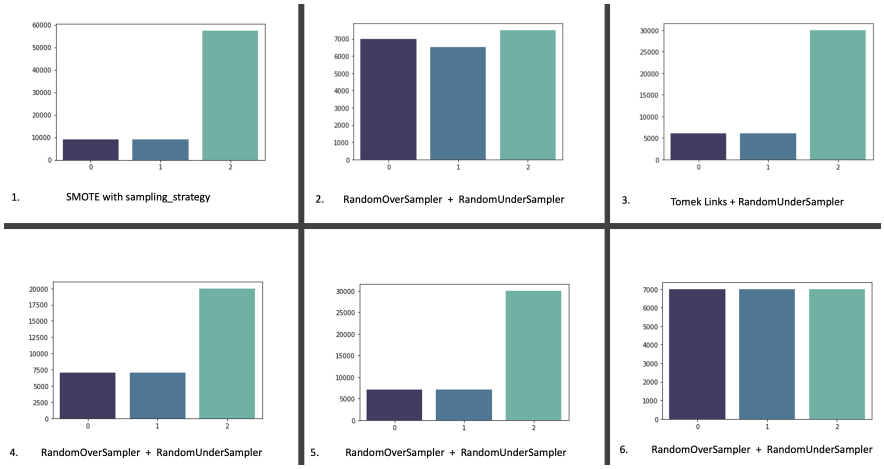

Figura 3. Distribuições finais do dataset

A configuração que foi usada como final para treinar os modelos de *machine learning* e *deep learning* foi a distribuição 5, visto que apresentou, na generalidade, melhores resultados que as outras.

No entanto, foi com a configuração 6 que se obteve os melhores resultados nas classes minoritárias, para o caso da rede neuronal (*CRNN*), que será explicada posteriormente na subsecção 5.3. Após esses resultados, também esta foi testada nos modelos de *machine learning*, todavia, sem o mesmo efeito.

4 Machine Learning

Após concretização do pré-processamento, pode então passar-se à aplicação dos algoritmos de *machine learning*, de forma a tentar classificar as diferentes opiniões.

O *dataset*, usando a distribuição 5, foi então dividido em 70% para o treino e 30% para o teste, e o grupo decidiu aplicar cinco algoritmos distintos, de forma a avaliar qual o algoritmo que melhor conseguia prever as classes[4][8]. São estes: *Logistic Regression*, *Naive Bayes*, *Random Forest*, *Support Vector Machines* e *XGBoost*.

4.1 Logistic Regression

Um dos algoritmos mais conhecidos mundialmente para problemas de classificação é *Logistic Regression*[11]. Trata-se um algoritmo muito utilizado e não tem propriamente hiperparâmetros que precisem de ser otimizados, o que faz com que seja de simples aplicação.

Foi então aplicado este algoritmo sobre a classificação de opiniões e os resultados podem ser visualizados de seguida (Figura 4).

	precision	recall	f1-score	support
0	0.49	0.56	0.52	1473
1	0.25	0.33	0.29	1494
2	0.96	0.94	0.95	28664
accuracy			0.89	31631
macro avg	0.57	0.61	0.59	31631
weighted avg	0.91	0.89	0.90	31631

Figura 4. Resultados do *predict* usando *Logistic Regression*.

Como se pode observar, a maior dificuldade está na classificação das opiniões negativas (0) e neutras (1), atingindo uma *precision* de 49% para as opiniões negativas e 25% para as neutras, o que é muito baixo. Já no *recall*, para as opiniões negativas é 56% e para as neutras 33%, o que é também muito baixo. Porém, tem-se que as opiniões positivas (2) estão com um nível de acerto bastante bom, apresentando uma *precision* de 96% e de *recall* 94%. A *accuracy* final deste algoritmo para este *dataset* foi de 89.36%, dado que a quantidade de opiniões positivas é muito superior relativamente às negativas e neutras.

4.2 Naive Bayes

O *Naive Bayes*[1] é um dos algoritmos muito recorridos no que diz respeito à classificação de texto. Este tem duas variantes mais usadas, que estão otimizadas para classificação de texto, e são estas *Multinomial Naive Bayes* e *Complement Naive Bayes*, sendo que esta última é mais otimizada para *datasets* desbalanceados.

De seguida, pode observar-se todos os valores obtidos usando primeiramente *Multinomial Naive Bayes* (Figura 5):

	precision	recall	f1-score	support
0	0.46	0.56	0.51	1473
1	0.22	0.36	0.27	1494
2	0.97	0.92	0.94	28664
accuracy			0.88	31631
macro avg	0.55	0.61	0.57	31631
weighted avg	0.91	0.88	0.89	31631

Figura 5. Resultados do *predict* usando *Multinomial Naive Bayes*.

Tal como referido anteriormente, opiniões negativas e neutras são muito difíceis de serem corretamente classificadas. Apresenta uma *precison* de 46% para as negativas e 22% para as neutras. Quanto ao seu *recall*, apresenta um valor de 56% e 27% respetivamente. As opiniões positivas continuam com uma boa *precision*, com 97%, e um *recall* de 92%. Quanto à *accuracy* do algoritmo, esta é de 87,90%.

Após a aplicação de *Multinomial Naive Bayes*, foi também aplicado o *Complement Naive Bayes*, em que os seus valores são os seguintes (Figura 6):

	precision	recall	f1-score	support
0	0.34	0.71	0.46	1473
1	0.20	0.27	0.23	1494
2	0.97	0.90	0.94	28664
accuracy			0.86	31631
macro avg	0.51	0.63	0.55	31631
weighted avg	0.91	0.86	0.88	31631

Figura 6. Resultados do *predict* usando *Complement Naive Bayes*.

Analisando o *Complement Naive Bayes*, e comparando com o *Multinomial Naive Bayes*, verifica-se que apenas o *recall* das negativas melhorou, passando de 56% para 71%. A *accuracy* final foi de 86.41%.

4.3 Random Forest

Foi também aplicado *Random Forest*[10]. Trata-se de um algoritmo de *ensemble*. Este algoritmo é ótimo para ser aplicada a otimização de hiperparâmetros, uma vez que tem imensos parâmetros que podem ser otimizados tais como: *max_depth*, *max_samples*, *min_samples_leaf*, *n_estimators*, etc.

De forma a saber que valor de *max_depth* usar para uma otimização de hiperparâmetros, foi calculado, para valores entre 0 e 50, qual o erro médio que o algoritmo tem para cada um dos valores. Tendo estes valores de erros médios, desenhou-se um gráfico que irá representar estes valores, de forma a saber-se quais valores a utilizar para a otimização de hiperparâmetros (Figura 7).

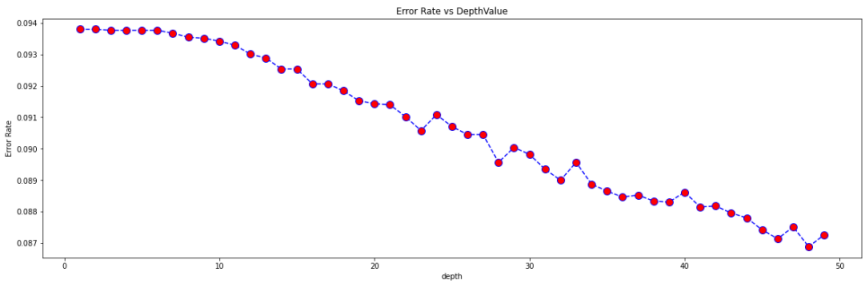

Figura 7. Gráfico que representa os erros para os diferentes valores de *max_depth*.

Seguidamente, utilizando os valores 45, 46 e 48, aplicou-se o *GridSearchCV* que, de forma exaustiva, vai procurar quais os parâmetros que dão uma maior *accuracy* na classificação deste *dataset*.

Após obtidos estes parâmetros, aplicaram-se os mesmos no algoritmo e avaliou-se quais os resultados obtidos pelos mesmos. São estes (Figura 8):

	precision	recall	f1-score	support
0	0.77	0.14	0.24	1473
1	0.50	0.01	0.02	1494
2	0.91	1.00	0.95	28664
accuracy			0.91	31631
macro avg	0.73	0.38	0.41	31631
weighted avg	0.89	0.91	0.88	31631

Figura 8. Resultados do *predict* usando *Random Forest*.

Com este algoritmo, e para os parâmetros usados, obervou-se que a *precision* de negativas e neutras aumenta para 77% e 50% respetivamente. No entanto,

estes valores não apresentam relevância, como se pode comprovar através do *recall*, cujo valor é extremamente baixo, sendo de 14% e 1% respetivamente. Nas positivas, apresenta uma *precision* de 91% e *recall* de 100%, o que indica que este algoritmo se está a focar nos resultados positivos, a forma a obter a melhor *accuracy* possível (com um valor de 91.23%).

4.4 Support Vector Machines

O *Support Vector Machines*[12] é um algoritmo que é ideal para otimização de hiperparâmetros. No entanto, este exige maior atenção na quantidade de parâmetros a utilizar, uma vez que é um algoritmo que demora muito tempo a obter resultados. Foi então realizada uma otimização de hiperparâmetros neste, usando *GridSearchCV*, para os parâmetros *C*, *gamma* e *kernel*.

Após identificação dos melhores parâmetros para a obtenção de uma melhor *accuracy*, estes foram usados no treino do algoritmo. Após realizado o teste deste, foram obtidos os resultados que se seguem (Figura 9).

	precision	recall	f1-score	support
0	0.58	0.41	0.48	1473
1	0.28	0.18	0.22	1494
2	0.94	0.97	0.96	28664
accuracy			0.91	31631
macro avg	0.60	0.52	0.55	31631
weighted avg	0.89	0.91	0.90	31631

Figura 9. Resultados do *predict* usando *Support Vector Machines*.

A *precision* para a classe *0* e *1* é de 58% e 28% respectivamente, e o seu *recall* é de 41% e 18%. Para a classe *2*, apresenta um *precision* de 94% e *recall* de 97%, dando uma *accuracy* final de 91.02%.

4.5 XGBoost

Por fim, aplicou-se o *XGBoost*[9]. Este algoritmo, tal como o *Random Forest* previamente referido, tem um parâmetro *max_depth*. Este parâmetro foi calculado para valores entre 0 e 50, na tentativa de encontrar o valor de *max_depth* que apresentava menor erro médio. Este gráfico é (Figura 10):

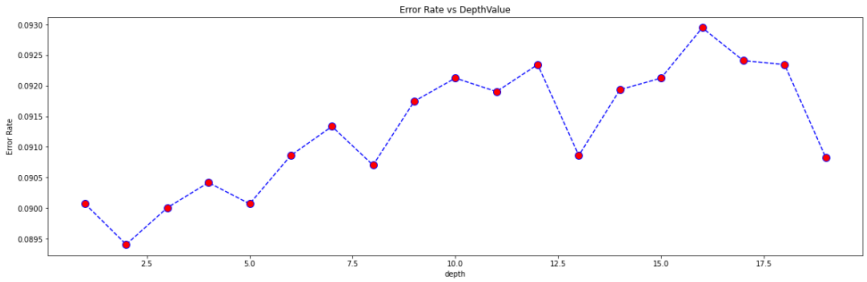

Figura 10. Gráfico que representa os erros para os diferentes valores de *max_depth*.

Depois de obtidos os menores erros, aplicou-se uma otimização de hiperparâmetros, usando o *GridSearchCV*. O parâmetros que se tentaram otimizar foram: *min_child_weight*, *gamma*, *subsample*, *colsample_bytree* e *max_depth*, sendo que os valores deste último são os valores que apresentavam menores erros médios.

Após o *fit* do treino e da previsão usando o teste, obteve-se os seguintes resultados (Figura 11):

	precision	recall	f1-score	support
0	0.62	0.36	0.45	1473
1	0.28	0.15	0.19	1494
2	0.94	0.98	0.96	28664
accuracy			0.91	31631
macro avg	0.61	0.49	0.53	31631
weighted avg	0.89	0.91	0.90	31631

Figura 11. Resultados do *predict* usando *XGBoost*.

A classe *0* e classe *1* apresentam 62% e 28% de *precision* respetivamente, e o seu *recall* é de 36% e 15%. Continua a verificar-se o problema em classificar opiniões negativas e neutras. Já na classe *2*, esta tem de *precision* 94% e de *recall* 98%, dando uma *accuracy* final de 91.06%.

4.6 One Class Classification

Devido ao enorme desbalanceamento do *dataset*, foi ainda aplicado um conceito existente em *machine learning* denominado *One Class Classification*[3]. Trata-se de um mecanismo que tem técnicas que permitem detetar *anomalies* e, assim, tentar classificar melhor as classes minoritárias. Este realizará uma classificação binária, sendo que, para tal, é necessário definir a classe *normal* e a classe considerada como anomalia, processo realizado antes do *fit* de um algoritmo. Numa primeira fase, a classe *normal* é a classe negativa, pelo que o seu valor passa de 0 para 1, e as classes *anomaly* serão o 1 e 2, que representam neutra e positiva, e os seus valores serão alterados para -1, representando então *anomaly*. Posto isto, é realizado o *fit* do algoritmo e averiguados os resultados obtidos. Finalmente, são definidas a classe 1 (neutra) como *normal* e as resposta como *anomaly*.

Foram aplicados os seguintes algoritmos para esta técnica: *Logistic Regression*, *Naive Bayes*, *Random Forest*, *One Class Support Vector Machines* e *XGBoost*.

Como os resultados obtidos não são de destaque relevante, estes não serão apresentados.

4.7 Análise de resultados

Os resultados dos diversos algoritmos aplicados podem ser observados na tabela que se segue.

	Prec 0	**Prec 1**	**Prec 2**	**Rec 0**	**Rec 1**	**Rec 2**	**Accuracy**
Logistic Regression	0.49	0.25	0.96	0.56	0.33	0.94	0.894
Multinomial NB	0.46	0.22	0.97	0.56	0.36	0.92	0.879
Complement NB	0.34	0.20	0.97	0.71	0.27	0.90	0.864
Random Forest	0.77	0.50	0.91	0.14	0.01	1.00	0.912
Support Vector Machines	0.58	0.28	0.94	0.41	0.18	0.97	0.910
XGBoost	0.62	0.28	0.94	0.36	0.15	0.98	0.911

Tabela 1. Resultados dos diferentes algoritmos.

A análise dos resultados permite verificar que classificar opiniões negativas e neutras é muito difícil.

No que diz respeito a opiniões negativas, o melhor modelo é *Logistic Regression*, uma vez que é o que apresenta melhores valores da combinação de *precision* e *recall*, sendo o que apresenta melhor *f1-score* para as opiniões negativas. Para as opiniões neutras, o melhor modelo foi também *Logistic Regression*, sendo este que apresenta melhores valores de *precision*, *recall* e, consequentemente, *f1-score*. Para opiniões positivas, o optou-se por não destacar nenhum algoritmo, uma vez que todos os algoritmos se apresentam, de certa forma, bons a classificar os mesmos.

Posto isto, e analisando os resultados, o algoritmo mais promissor é *Logistic Regression*. No entanto, com uma correta otimização de hiperparâmetros, outros algoritmos poderiam apresentar uma melhoria significativa quando comparando com este último.

5 Deep Learning

Os algoritmos de *Deep learning* são construídos por vários níveis de representação obtidos através da composição de métodos simples, porém não lineares, começando por aplicar as transformações no primeiro nível de representação da informação (*raw input*) e terminando na camada de representação mais específica. Com a composição destas transformações, estes algoritmos são capazes de extrair informações relevantes. Para tarefas de classificação, as últimas camadas são essenciais, uma vez que representam aspetos mais específicos do *input* que são importantes para a discriminação e suprimem características irrelevantes. [5]
Sendo as redes neuronais um dos algoritmos mais utilizados nesta área, foram estudadas várias variações das mesmas, sendo que a primeira camada destas redes foi uma *layer* de *word embeding*.
Praticamente, *word embedding* são uma forma de representação de palavras onde existem relacionamentos entre as mesmas. Estas estão num espaço n-dimensional (neste caso 300d), onde as palavras que têm o mesmo significado têm uma representação semelhante, ou seja, quando duas palavras são semelhantes, serão representadas por vetores que serão colocados muito próximos num espaço vetorial, como se pode ver na seguinte imagem:

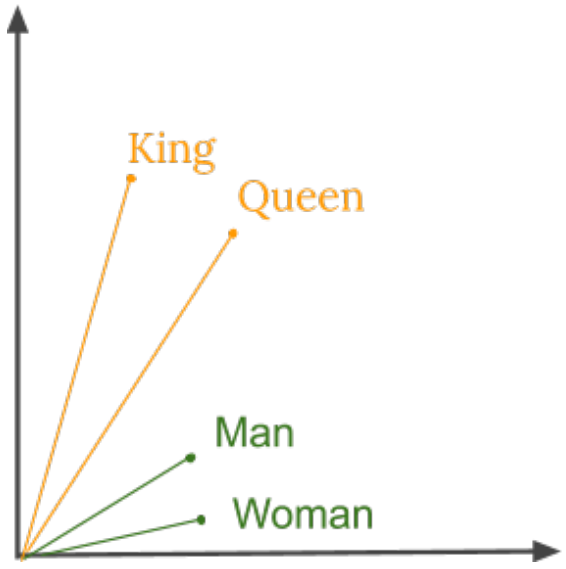

Figura 12. Representação dos vetores de palavras

Para a implementação das diferentes variantes das redes neuronais, foi utilizada a biblioteca do *keras* em *python*. De seguida, serão demonstradas as variantes estudadas.

5.1 CNN

As *Convolutional Neural Networks* são um tipo de rede neuronal robusta e frequentemente utilizadas para reconhecimento e pré-processamento de imagens. Todavia, em vários estudos recentes[7], estas também têm resultados relevantes no âmbito de classificação de texto. Esta rede é composta por várias camadas, sendo que as primeiras camadas servem para extrair características do *input* e as camadas densas, ou escondidas, servem para classificação. Deste modo, foi criada uma rede com várias camadas convolucionais, sendo que, posteriormente, foi usada a biblioteca *kerastunner* para encontrar os melhores hiperparâmetros para a mesma. Após este processo, foi então criada a seguinte rede:

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, None)]            0
_________________________________________________________________
embedding_2 (Embedding)      (None, None, 300)         5280300
_________________________________________________________________
conv1d_6 (Conv1D)            (None, None, 416)         125216
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, None, 416)         0
_________________________________________________________________
conv1d_7 (Conv1D)            (None, None, 496)         206832
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, None, 496)         0
_________________________________________________________________
conv1d_8 (Conv1D)            (None, None, 320)         159040
_________________________________________________________________
global_max_pooling1d_2 (Glob (None, 320)               0
_________________________________________________________________
flatten_2 (Flatten)          (None, 320)               0
_________________________________________________________________
dense_6 (Dense)              (None, 64)                20544
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_7 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_8 (Dense)              (None, 3)                 195
=================================================================
Total params: 5,796,287
Trainable params: 515,987
Non-trainable params: 5,280,300
_________________________________________________________________
```

Figura 13. Constituição da rede neuronal convolucional otimizada

A rede acima representada foi treinada com um *batch size* de 64 e 10 epochs10, sendo que também foram usadas *callbacks* para guardar o modelo com menor *loss*, redução do *learning rate* dependendo da *loss*, entre outras. Após o treino, foi avaliado o modelo no *dataset* de teste, obtendo o seguinte resultado.

	precision	recall	f1-score	support
Negative	0.55	0.64	0.59	929
Neutro	0.33	0.20	0.25	1003
Positive	0.96	0.97	0.96	19157
accuracy			0.92	21089
macro avg	0.61	0.60	0.60	21089
weighted avg	0.91	0.92	0.91	21089

Figura 14. Resultados do modelo da CNN

É possível verificar que esta rede classifica razoavelmente as opiniões negativas, com algumas falhas na classificação de opiniões neutras e boa performance a classificar as opiniões positivas, como já seria de esperar, uma vez que são os mais abundantes no *dataset* e não forma gerados "artificialmente". De forma a verificar os casos que deveriam ser negativos e foram neutros, apresenta-se a matriz de confusão:

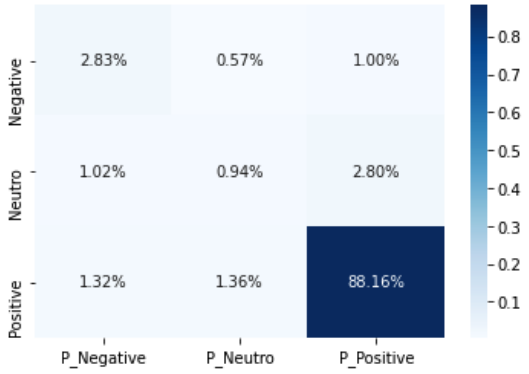

Figura 15. Matriz de confusão do modelo da CNN

Através desta matriz, é possível verificar que a rede está a classificar alguns casos positivos como negativos, e alguns casos negativos como positivos, tornando-se assim num modelo médio, visto que estes casos não deveriam existir.

5.2 RNN

A Recurrent Neural Network(RNN) é uma generalização da rede neural *feedforward*(CNN) que tem uma memória interna. As RNNs utilizam a mesma função para cada *input*, enquanto o *output* depende do cálculo passado, ou seja, após produzir o *output*, este é copiado e enviado de volta para a rede recorrente. Em cada decisão tomada, considera-se o *input* atual e esse *output* que aprendeu

com o *input* anterior.
No entanto, ao contrário das redes neuronais *feedforward*, onde os *inputs* são independentes uns dos outros, nas RNNs pode usar-se o seu estado interno (memória) para processar sequências de *inputs*, tornando assim todos os *inputs* relacionados uns com os outros.
Estas redes podem ser constituídas por camadas LSTM ou GRU, e existem várias diferenças entre as mesmas,nomeadamente:

- Ambas as abordagens estão a utilizar uma forma diferente de junção da informação do *input* anterior, uma vez que o fluxo de gradiente nas LSTMs provém de três caminhos diferentes (portões), observando-se uma maior variabilidade na descida do gradiente em comparação com as GRU, pois só provém de dois caminhos distintos.
- Se se pretende um modelo mais rápido e compacto, as *GRU* seriam a melhor escolha, devido a terem menos parâmetros.
- Por outro lado, caso se esteja a lidar com grandes conjuntos de dados, o maior poder expressivo das LSTM pode levar a resultados superiores. Em teoria, as células LSTM devem lembrar-se de sequências mais longas do que as GRU e superá-las em tarefas que exijam correlações de longo alcance de modelação.

Posto isto, foi testado com ambas as camadas, sendo que as que apresentaram melhor performance foram as LTSM. Assim sendo, foi criada a estrutura de uma RNN e otimizada através do *kerastuner*, obtendo assim os melhores hiperparâmetros para esta rede. Apresenta-se de seguida a rede obtida:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, None, 300)         5280300
_________________________________________________________________
lstm_1 (LSTM)                (None, None, 128)         219648
_________________________________________________________________
global_max_pooling1d_4 (Glob (None, 128)               0
_________________________________________________________________
dense_12 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_13 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_14 (Dense)             (None, 3)                 195
=================================================================
Total params: 5,512,559
Trainable params: 232,259
Non-trainable params: 5,280,300
_________________________________________________________________
```

Figura 16. Constituição da rede neuronal recorrente otimizada

A rede previamente apresentada foi também treinada com um *batch size* de 64 e 10 epochs, sendo que também foram usadas as mesmas *callbacks* que nas CNN. Após o treino, foi obtido o seguinte resultado:

	precision	recall	f1-score	support
Negative	0.62	0.55	0.58	929
Neutro	0.36	0.35	0.35	1003
Positive	0.96	0.97	0.97	19157
accuracy			0.92	21089
macro avg	0.65	0.62	0.64	21089
weighted avg	0.92	0.92	0.92	21089

Figura 17. Resultados do modelo da RNN

Como se pode verificar, este modelo apresenta melhor *precision* do que o anterior e ainda com um maior *recall* nas opiniões neutras. Porém, apresenta menor *recall* nas opiniões negativas. No caso das opiniões positivas, ambos apresentam a mesma performance. É de realçar que a média de *f1-score* é de 64 %, o que já é razoável.

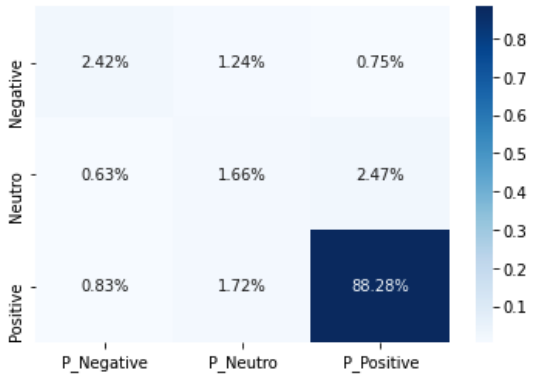

Figura 18. Matriz de confusão do modelo da RNN

Através desta matriz, é possível verificar que o problema anterior já foi melhorado, visto que já existem mais casos positivos classificados como neutros do que casos positivos classificados como negativos, sendo que o mesmo acontece para os negativos classificados como positivos, o que é um bom indício. É de realçar que esta rede também classifica mais casos neutros como positivos do que negativos, o que também é favorável. Em suma, este é o melhor modelo até ao momento.

5.3 CRNN

Após o resultado destas duas variantes de redes neuronais, decidimos combinar as duas, colocando primeiro as camadas convolucionais e posteriormente as camadas LSTM. Quanto às camadas densas(*hidden layer*), dado que o *kerastunner* retornou os mesmos hiperparâmetros para as duas variantes, foram usadas as mesmas configurações, como podemos ver na seguinte imagem:

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, None, 300)         5280300
_________________________________________________________________
conv1d_12 (Conv1D)           (None, None, 416)         125216
_________________________________________________________________
max_pooling1d_9 (MaxPooling1 (None, None, 416)         0
_________________________________________________________________
conv1d_13 (Conv1D)           (None, None, 496)         206832
_________________________________________________________________
max_pooling1d_10 (MaxPooling (None, None, 496)         0
_________________________________________________________________
dropout_7 (Dropout)          (None, None, 496)         0
_________________________________________________________________
conv1d_14 (Conv1D)           (None, None, 320)         159040
_________________________________________________________________
max_pooling1d_11 (MaxPooling (None, None, 320)         0
_________________________________________________________________
lstm_3 (LSTM)                (None, None, 128)         229888
_________________________________________________________________
global_max_pooling1d_6 (Glob (None, 128)               0
_________________________________________________________________
dense_18 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_8 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_19 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_20 (Dense)             (None, 3)                 195
=================================================================
Total params: 6,013,887
Trainable params: 733,587
Non-trainable params: 5,280,300
_________________________________________________________________
```

Figura 19. Constituição da rede neuronal convolucional recorrente otimizada

Esta rede foi treinada com várias configurações (incluindo as que foram usadas para os modelos anteriores). Todavia, já com vista ao modelo híbrido, foi feita uma tentativa de classificação mais eficaz para os casos negativos e positivos (classes minoritárias do *dataset*), combatendo a classe positiva com o *Vader*. Apresenta-se, de seguida, os resultados obtidos para a mesma.

	precision	recall	f1-score	support
Negative	0.44	0.63	0.52	929
Neutro	0.14	0.67	0.24	1003
Positive	0.99	0.78	0.87	19157
accuracy			0.77	21089
macro avg	0.53	0.69	0.54	21089
weighted avg	0.93	0.77	0.83	21089

Figura 20. Resultados do modelo da CRNN

Como se pode verificar nos resultados obtidos, esta rede apresenta o maior *recall* nas classes minoritárias, beneficiando a classe neutra, visto ser a classe mais complicada de prever. Quanto à previsão, esta apresenta um baixo resultado para a classe neutra, fazendo com que sejam classificados bastantes opiniões positivas como neutras, como se pode verificar na matriz de confusão apresentada em seguida.

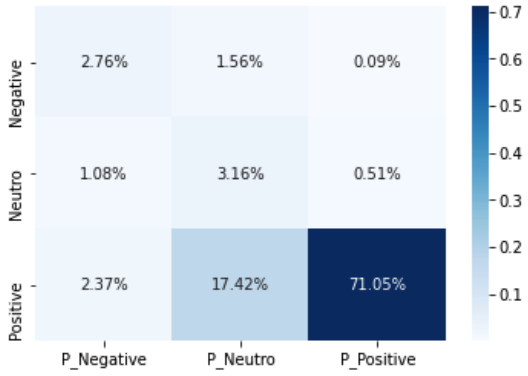

Figura 21. Matriz de confusão do modelo da CRNN

É ainda de realçar que apenas 18 opiniões (0.09% do *dataset* teste) que eram negativos foram classificadas como positivas, o que é um ótimo resultado. Posto isto isto, este foi o modelo escolhido para modelo híbrido, visto ser o que classifica melhor as classes minoritárias.

6 Modelo Híbrido

Por fim, o objetivo principal deste trabalho está na criação de um modelo híbrido que incorpora o *output* do modelo léxico desenvolvido com *VADER* no ano anterior, e o melhor modelo de *Machine Learning* que foi obtido. O modelo para *Machine Learning* aplicado nesta secção foi o representado na secção 5.3 CRNN.

Para fazer a junção dos dois modelos, tentou-se observar os pontos fracos de cada um dos modelos, tendo-se chegar ao seguinte algoritmo.

Caso de decisão	Modelo Vader	Modelo Machine Learning	Resultado
1	2.7 <= x <= 3.3	Neutro	Neutro
2	x <2.7	Neutro	Negativo
3	x >3.3	Neutro	Positivo
4	x >= 3.15	Positivo	Positivo
5	2.85 <x <= 3.15	Positivo	Neutro
6	2.85 <= x <3.15	Negativo	Neutro
7	NULL	Positivo	Positivo
8	NULL	Neutro	Neutro
9	NULL	Negativo	Negativo

Tabela 2. Algoritmo de decisão do resultado entre os dois modelos.

Na tabela demonstrada em cima, x representa o valor retornado pelo modelo *VADER* entre 0 e 5. Por outro lado, o modelo de *machine learning* (ML) retorna um de três valores, 0 ou 1 ou 2, correspondentes a negativo, neutro, positivo, respetivamente.

Estes diferentes casos têm todos a sua razão de existir e vão ser explicados de seguida:

- **Caso 1:** Visto que o modelo *VADER* possui uma definição de neutro demasiado restritiva, apenas quando o *score* é 3.0, decidiu definir-se uma *range* de valores do quais se considerou aceitável considerar a critica neutra;
- **Caso 2:** Se o resultado obtido pelo *VADER* for inferior a 2.7 e o resultado de *ML* for neutro, devido ao reduzido número de frases neutras a partir das quais se treinou o modelo de *ML*, decidiu-se dar mais peso ao *output* do modelo léxico;
- **Caso 3:** Se o resultado obtido pelo *VADER* for superior a 3.3 e o resultado de *ML* for neutro, devido ao reduzido número de frases neutras a partir das quais se treinou o modelo de *ML*, decidiu-se dar mais peso ao *output* do modelo léxico;
- **Caso 4:** Se ambos derem positivo, o resultado é positivo;
- **Caso 5:** Se o resultado for por volta de valores neutros no modelo *VADER* e o de *ML* for positivo, o resultado final vai ser neutro;

- **Caso 6:** Se o resultado for por volta de valores neutros no modelo *VADER* e o de *ML* for negativo, o resultado final vai ser neutro;
- **Caso 7/8/9:** Se nenhum dos casos anteriores se verificarem, vai ser escolhido como resultado final a polaridade devolvida pelo modelo de *Machine Learning*.

6.1 Resultados obtidos com Modelo Híbrido

O modelo híbrido foi testado através do uso de uma pequena porção do *dataset*. Neste caso, fez-se uso das primeiras 1001 opiniões presente no *dataset*. Tendo-se obtido os seguintes resultados:

$$
\begin{aligned}
&Número\ de\ opiniões\ avaliadas\ =\ 1001\\
&\\
&Precisão\ do\ modelo\ =\ 90.10\%\\
&\\
&Opiniões\ Negativas\ =\ 57\\
&Opiniões\ Neutras\ =\ 54\\
&Opiniões\ Positivas\ =\ 890\\
&\\
&Opiniões\ Negativas\ classificadas\ como\ Negativas\ =\ 56\\
&Opiniões\ Negativas\ classificadas\ como\ Neutras\ =\ 0\\
&Opiniões\ Negativas\ classificadas\ como\ Positivas\ =\ 1\\
&\\
&Opiniões\ Neutras\ classificadas\ como\ Negativas\ =\ 6\\
&Opiniões\ Neutras\ classificadas\ como\ Neutras\ =\ 11\\
&Opiniões\ Neutras\ classificadas\ como\ Positivas\ =\ 37\\
&\\
&Opiniões\ Positivas\ classificadas\ como\ Negativas\ =\ 29\\
&Opiniões\ Positivas\ classificadas\ como\ Neutras\ =\ 26\\
&Opiniões\ Positivas\ classificadas\ como\ Positivas\ =\ 835
\end{aligned}
$$

É de realçar que a única opinião que está classificada como negativa no *dataset* e o modelo híbrido classificou como positiva é a seguinte:

- **"the tutorials is not for average students it takes a very high skill to understand them also the quiz and the assignments a too tough hope you can make them a bit easy the last one in with graphs and trees was a bouncer if the maths required is so high please take a week to first explain the match fresher required in course and then start the main course."**

Esta frase, dependendo do contexto pode ser negativa, todavia, dado que os modelos não têm noção do contexto, esta também poderia ser neutra, tal como o modelo de *machine learning* a classificou porém, o *VADER*, como a classificou como extremamente positiva, a classificação final foi positiva. Este é um problema da classe *target*, pois foi calculada a partir do número de estrelas dadas pelo utilizador.

No entanto, como se pode averiguar, tanto para opiniões positivas como para negativas, o *dataset* teve uma performance muito boa. O mesmo já não se verifica para opiniões neutras. Estas eram de muito difícil classificação, pois é difícil para um computador encontrar as nuances na frase que indicam um caractere neutro na sua polaridade. De qualquer forma, o grupo considera ter consigo atingir um resultado bastante promissório no modelo híbrido.

7 Conclusão e Trabalho futuro

Com a realização deste trabalho, foi possível verificar a utilidade de *machine learning* e *deep learning* para a classificação de textos, neste caso opiniões. Também foi possível averiguar a importância que um modelo híbrido pode ter num sistema de classificação como o presente.

Os principais desafios durante o desenvolvimento deste projeto prenderam-se em vários pontos. Em primeiro lugar, é difícil arranjar *dataset* com um tamanho considerável na área de ensino. Posto isto, o *dataset* escolhido tem uma adversidade, visto que a classe *target* foi baseada nas estrelas dada pelo utilizador, e por isso pode induzir em erro o modelo que está a ser treinado.

Por outro lado, como trabalho futuro pode-se propor o estudo de novas técnicas de geração de texto, visto que o balanceamento do *dataset* tem bastante influência na performance dos modelos.

Por fim, outro ponto que poderia ser mais aprimorado seria as abordagem para o modelo híbrido, nomeadamente, junção de vários modelos e/ou até com o dicionário léxico *VADER*, fazer uma otimização dos pesos escolhidos para cada um dos modelos apresentados no modelo híbrido.

Em suma, conclui-se que o trabalho desenvolvido se encontra bastante favorável e cumpre todos requisitos pretendidos pela equipa docente, no contexto de investigação de vários modelos para a realização do modelo híbrido, uma vez que, conseguimos chegar a soluções bastante pertinentes. Posto isto, o grupo considera ainda que este projeto foi uma mais valia para o aumento do seu conhecimento nestas áreas.

Referências

[1] *1.9. Naive Bayes — scikit-learn 0.24.2 documentation*. Inglês. 2007. URL: https://scikit-learn.org/stable/modules/naive_bayes.html , visited at 27/05/2021.

[2] Jan Adona. *Dataset*. Inglês. 7 de ago. de 2017. URL: https://www.kaggle.com/septa97/100k-courseras-course-reviews-dataset , visited at 27/05/2021.

[3] Jason Brownlee. *One Class Classification Algortihm*. Inglês. 14 de fev. de 2020. URL: https://machinelearningmastery.com/one-class-classification-algorithms/ , visited at 27/05/2021.

[4] Lopes João Miguel Ferreira. "Análise de Sentimentos em Conteúdos Textuais". Em: (2018).

[5] Yann Lecun, Yoshua Bengio e Geoffrey Hinton. "Deep learning". Em: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[6] Ajinkya More. "Survey of resampling techniques for improving classification performance in unbalanced datasets". Em: (ago. de 2016).

[7] Hao Peng et al. "Large-Scale Hierarchical Text Classification with Recursively Regularized Deep Graph-CNN". Em: *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW 18* (2018). DOI: 10.1145/3178876.3186005.

[8] Anjuman Prabhat e Vikas Khullar. "Sentiment classification on big data using Naïve bayes and logistic regression". Em: (2017), pp. 1–5. DOI: 10.1109/ICCCI.2017.8117734.

[9] *Python Package Introduction — xgboost 1.5.0-dev documentation*. Inglês. 2007. URL: https://xgboost.readthedocs.io/en/latest/python/python_intro.html , visited at 27/05/2021.

[10] *sklearn.ensemble.RandomForestClassifier — scikit-learn 0.24.2 documentation*. Inglês. 2007. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html , visited at 27/05/2021.

[11] $sklearn.linear_model.LogisticRegression|scikit-learn0.24.2documentation$. Inglês. 2007. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[12] *sklearn.svm.SVC — scikit-learn 0.24.2 documentation*. Inglês. 2007. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html , visited at 27/05/2021.