

# Reddit classification

Pedro Escaleira  
Departamento de Electrónica  
Telecomunicações e Informática  
Universidade de Aveiro  
Aveiro, Portugal  
escaleira@ua.pt

Rafael Simões  
Departamento de Electrónica  
Telecomunicações e Informática  
Universidade de Aveiro  
Aveiro, Portugal  
rafaeljsimoes@ua.pt

**Abstract**—Neste relatório, iremos descrever a utilização de alguns dos algoritmos mais conhecidos e usados na área de Aprendizagem Automática e o seu comportamento perante diferentes situações, usando para isso um *dataset* de postagens criados na rede social *Reddit*. Mais concretamente, iremos demonstrar o estudo feito utilizando os algoritmos *Redes Neurais*, *Redes de Bayes* e *Regressão Logística* para catalogação de texto.

## I. INTRODUÇÃO

Aprendizagem Automática é uma área das ciências da computação com cada vez mais destaque nos dias que correm. Desde a sua utilização em campos como a saúde, sistemas bancários, *self driving cars*, nas aplicações de telemóvel do utilizador comum, entre outros, não é surpresa para ninguém que se tornou uma realidade sem a qual a sociedade em que vivemos seria muito diferente do que é hoje.

Este trabalho, proposto pela professora Petia Georgieva, do Departamento de Electrónica Telecomunicações e Informática da Universidade de Aveiro, teve o intuito de consolidar e instigar a utilização e pesquisa dos conceitos apresentados na disciplina de Tópicos de Aprendizagem Automática. Desta forma, estudamos o comportamento dos algoritmos **Redes Neurais**, **Redes de Bayes** e **Regressão Logística** no âmbito de um trabalho de classificação de texto, mais especificamente classificação de textos em *subreddits* obtidos da rede social *Reddit*<sup>1</sup>. Esta rede social é caracterizada por permitir a interacção pública e livre entre os seus utilizadores sobre os mais variados temas. De forma a que estas discussões e divulgação de informação sejam feitas de forma organizada e explícita, os temas são usualmente agrupados em *subreddits* que são, no fundo, grupos em que qualquer utilizador da rede social se pode conectar e discutir, ou apenas observar uma discussão, sobre esse assunto em específico.

A razão de termos escolhido este tema, foi que ia de encontro a uma das *features* que o projecto de informática dos dois autores requeria: analisar *tweets* e obter qual o tema principal que era discutido nos mesmos. Para mais informações sobre este projecto é favor consultar a página [1]. Sendo que, tal como no estudo discutido neste relatório, é suposto criarmos um modelo que tenha uma boa performance a detectar qual o *subreddit* associado a uma postagem e, sendo assim, tendo em

conta o conceito de *subreddit*, ligar qual o "tema de conversa" de uma determinada postagem<sup>2</sup> a ela mesma, pretendemos encontrar qual o modelo que tem a melhor performance em detectar os temas duma dada postagem, de forma a utiliza-lo para fazer essa mesma tarefa em relação aos *tweets* associados ao nosso projecto de informática.

Além da implementação e estudo dos algoritmos enumerados anteriormente, foi também feita a implementação e estudo aprofundado de pré-processamento e extracção de *feature* de texto.

## II. DADOS

### A. Dados originais

Os dados usados neste estudo foram obtidos e proporcionados pela *Evaluation AI*<sup>3</sup>. Estes consistem em *self-posts* feitos no *Reddit* entre 2016/06/01 e 2018/06/01 (2 anos). De forma a garantir alguma qualidade, os dados foram submetidos a filtragem por parte da *Evaluation AI*:

- *Posts* submetidas noutra língua que não o inglês.
- *Posts* submetidos por moderadores ou *admins* do *Reddit*.
- *Posts* cujo corpo de texto possuísse menos do que 256 ou mais do que 4096 caracteres.
- *Posts* duplicados.

Desta maneira, o *dataset* final é constituído por postagens de 1013 classes, sendo que cada uma possui 1000 postagens. Mais informações sobre o tratamento ao qual os dados foram submetidos ou sobre algum estudo feito sobre eles podem ser encontradas em [2]. As classes são, por isso, o nome dos *subreddits* i.e., um dado tema, e cada uma possui 1000 postagens (exemplos) feitas nesse *subreddit*.

### B. Divisão dos dados para o estudo

Apesar do *dataset* original possuir 1013 classes distintas, por uma questão de *hardware constraints*, decidimos usar apenas 50 para realizar o nosso estudo, sendo que a selecção das mesmas foi feita de forma aleatória. Desta forma, todo o trabalho foi feito usando 50 mil exemplos de *posts* distintos, já que cada classe, como já referido, possui 1000 amostras.

<sup>2</sup>Neste trabalho, tal como perceptível nesta introdução, é tida como premissa que um *subreddit* representa, em ultima instância, o "tema de conversa" associado às postagens feitas nesse *subreddit*.

<sup>3</sup>Os dados podem ser encontrados em: <https://www.kaggle.com/mswarbrickjones/reddit-selfposts>

<sup>1</sup>*Subreddits* consistem em comunidades do *Reddit*, onde os utilizadores associados falam entre si sobre um tema em concreto, associado a esse mesmo *subreddit*

Como recomendação dos proprietários do *dataset*, usamos a divisão 80%-20% para dados de treino/*cross validation* e de teste, já que os dados originais se encontravam organizados de forma a que, com esta divisão, houvessem exactamente 800 amostras para treino/*cross validation* e 200 amostras para teste em cada classe distinta. Nos gráficos da figura 1 é possível verificar a uniformidade esta distribuição.

### C. Pré-processamento dos dados

Sabendo que este trabalho se trata de um problema de **classificação de texto**, é inevitável que exista algum processamento dos dados de entrada (textos), já que a maioria dos modelos existentes não são compatíveis com entradas não numéricas. Para além de questões de conveniência, nesta etapa também é implementado um mecanismo de escolha de *features*.

Para uma fácil compreensão dos mecanismos usados neste passo, a explicação vai ser feita de uma forma enumerada:

1) **Conversão de texto para tokens:** Nesta fase os textos presentes no *dataset* são transformados numa lista de *tokens*:

- Divisão do texto numa lista de *tokens*. Ex.: `word_tokenize("At eight o'clock on Thursday morning.")` = ['At', 'eight', 'o'clock', 'on', 'Thursday', 'morning', '.'].  
• Todos os *tokens* são convertidos para letra minúscula.
- Pontuação é removida.
- É realizado o processo de lematização nos verbos presentes. Ex: inflected: inflect, ate: eat.
- São retirados os *tokens* que estão incluídos no conjunto de *stop words* inglesas.

Supondo o seguinte cenário, em que se pretende converter o texto "*Lemmatization (or lemmatization) in linguistics is the process of grouping together the inflected forms of a word*" para uma lista de *tokens* o output esperado seria: ['lemmatization', 'lemmatization', 'linguistics', 'process', 'group', 'together', 'inflect', 'form', 'word']

2) **Conversão da lista de tokens para uma matriz de *TF-IDF features*.** Apesar da transformação anterior em *tokens*, é necessário uma nova etapa para conseguir extrair *features* dos textos disponibilizados, para que estes sejam usados à posteriori nos modelos de *Machine Learning* em estudo. Para isso foi usado o algoritmo ***TF\*IDF***, que é uma técnica de extracção de informação que pesa a frequência de um termo (*TF*) i.e, o número de vezes que o termo aparece, e a frequência inversa de documento (*IDF*) i.e, uma pontuação que define o quão importante é um termo relativamente ao *corpus* total. O produto das pontuações *TF* e *IDF* de um termo é chamado de peso ***TF\*IDF***. A importância do termo no *corpus* cresce com o aumento da pontuação ***TF\*IDF*** do respectivo termo.

Para mais detalhes, é favor consultar [3].

Apenas de referir que nesta etapa foram considerados todos os uni-gramas e bigramas<sup>4</sup>. Após o processo estar completo este algoritmo retorna um conjunto de termos (palavras) com o respectivo valor ***TF\*IDF*** mapeado. Este conjunto de palavras define o vocabulário do nosso *corpus*.

3) **Diminuição do vocabulário proveniente do algoritmo**

***TF\*IDF***: O conjunto de palavras definido anteriormente reflecte a quantidade de *features* de entrada para o nosso modelo, sendo que, como é expectável, este pode possuir dimensões elevadas. Devido a este facto, é essencial reduzir o numero de *features* (numero de termos do vocabulário), para que os processos de treino sejam optimizados e para cumprir limitações de *hardware*. Apesar de aparentar que podemos retirar informação importante do nosso *corpus*, a diminuição do vocabulário torna-se pouco influente se removermos os termos com valor baixo de ***TF\*IDF*** (termos que não transmitem informação útil sobre o *corpus*).

Ordenando os termos de vocabulário por ordem decrescente do valor de ***TF\*IDF***, obtemos um gráfico como o apresentado na figura 2. Como se pode observar, este possui uma forma exponencial, sendo que os valores mais à esquerda (com maior valor de ***TF\*IDF***) representam as palavras mais importantes no *corpus* que são, por isso, as palavras mais importantes. Aqui reforça-se a ideia da necessidade de segmentar o vocabulário, pois este atinge dimensões na ordem dos milhões.

Para conseguirmos segmentar os termos do vocabulário, tendo em conta a forma de exponencial que este apresenta, foi usado o algoritmo *Kneedle* [4], que tem como objectivo seleccionar o ponto onde o vocabulário "deve ser cortado", sendo o ponto ideal o "joelho" (*knee*) do gráfico. Basicamente o algoritmo escolhe o ponto ideal que divide os termos importantes dos não importantes. Na figura 3, podemos observar o *knee\_point* que o algoritmo seleccionou, 7299. Isto implica uma redução de 99% no tamanho do vocabulário inicial.

Apesar da redução ser bastante significativa, 7299 termos de vocabulário continua a ser um número elevado de *features*, sendo que para isso uma nova segmentação é realizada com o novo vocabulário reduzido. Como demonstrado na figura 4, o novo vocabulário continua com a forma de exponencial. Posto isto, este processo é realizado até o algoritmo não conseguir encontrar o *knee point* - este processo pode ser visualizado no gráfico 5. Como se pode observar, caso a segmentação de dados chegue ao fim, na ultima iteração deste procedimento, o numero de *features* é muito reduzido (11), o que não permite ao modelo obter informação suficiente sobre os documentos. Para evitar isto, foi implementado um

<sup>4</sup>**n-grama:** É uma sequência contínua de N itens de uma determinada amostra de texto. Neste caso foram considerados n = 1 e n = 2

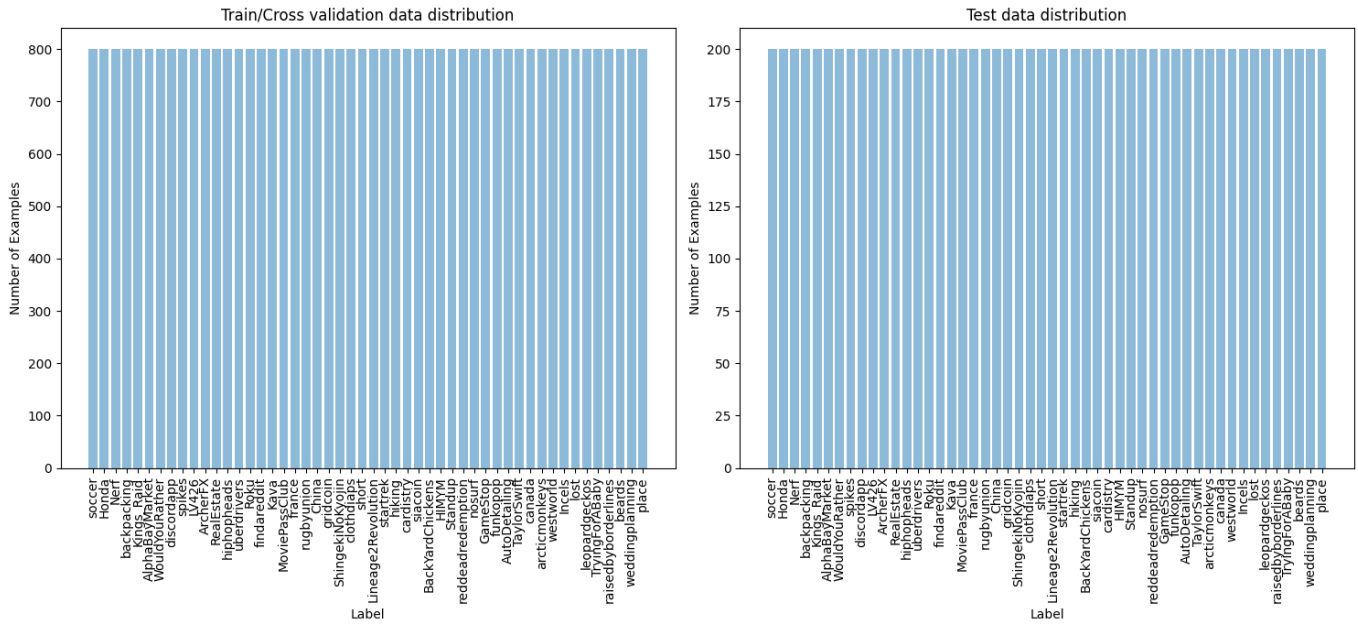


Fig. 1. Distribuição de dados por *label* (nomes dos *subreddits*) por *dataset*.

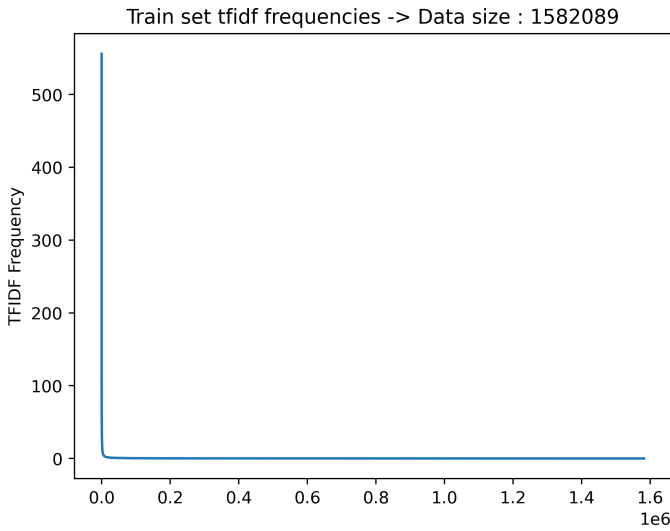


Fig. 2. Frequências TF\*IDF do corpus ordenados de forma decrescente

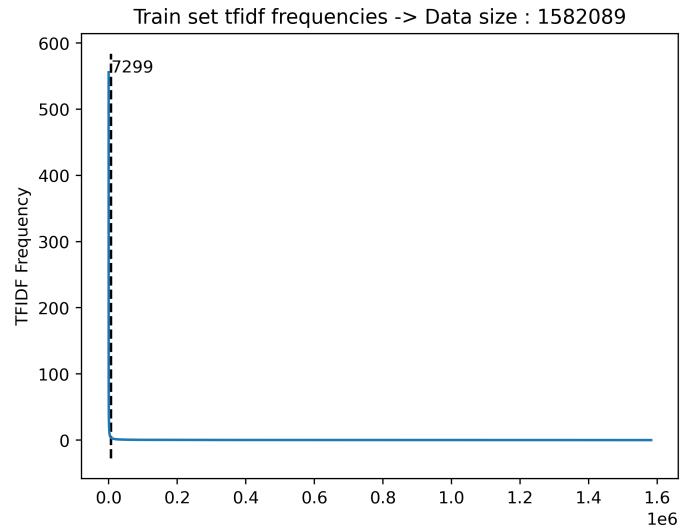


Fig. 3. Frequências TF\*IDF do corpus ordenados de forma decrescente com o *knee\_point*

parâmetro de limite, que define o numero mínimo de termos que o vocabulário deve ter. Por exemplo, se o limite for de 90 termos, o processo terminaria no terceiro gráfico da figura 5 e o vocabulário ficaria com 93 termos.

#### 4) Conversão da lista de *tokens* para uma matriz de *TF features* usando o novo vocabulário reduzido na etapa anterior

Após o processo de segmentação do vocabulário, o numero de *features* escolhido foi 600 termos.

Neste processo foi usado o algoritmo *TF*, que mapeia a frequência de um termo tendo em conta o vocabulário

definido anteriormente. Com estes valores temos uma noção de quantas vezes as palavras mais importantes são usadas nos documentos do *dataset*.

Após estas etapas os dados estão prontos para transitar para a fase de treino.

### III. ESTUDO COM REDES NEURONAIS

Sendo que o pressuposto das Redes Neurais é que tenham um bom desempenho, em termos da qualidade da prestação de modelos treinados usando o mesmo, foi a primeira abordagem que decidimos dar ao problema.

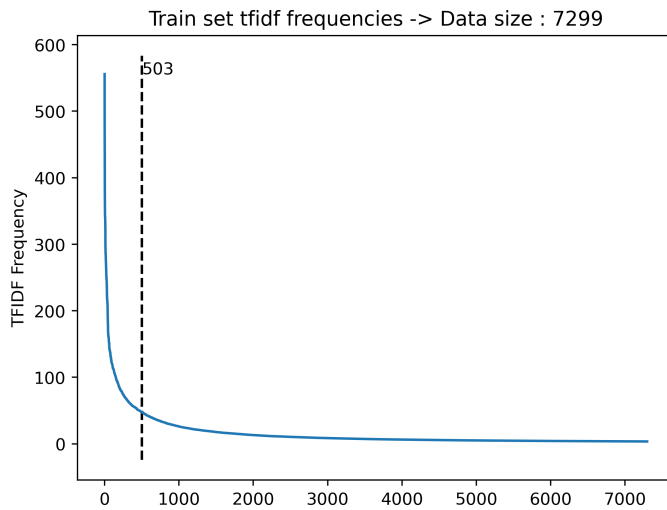


Fig. 4. Frequências TF\*IDF do corpus reduzido ordenados de forma decrescente com o *knee\_point*

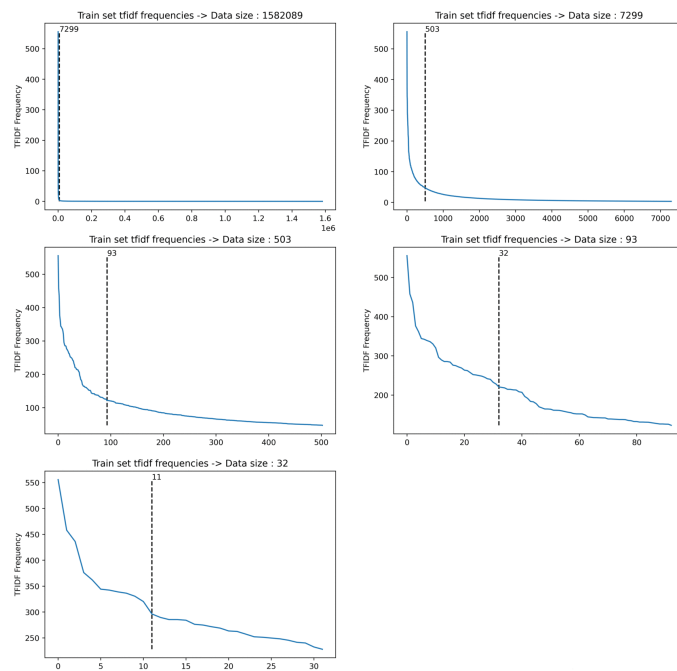


Fig. 5. Sequências de gráficos frequências TF\*IDF com o *knee\_point*

#### A. Estrutura usada

Apesar do *dataset* ter dimensões consideráveis para o *hardware* disponível, consideramos que este problema consegue ser resolvido com uma rede neuronal "normal". O uso de uma rede convolucional neste problema acaba por ser evitado, pois na fase de pré-processamento existiram métodos de selecção de *features* suficientes para o modelo ter uma performance boa, não sendo necessário que o modelo tenha a responsabilidade de fazer a procura de *features*. Então após a experimentação de alguns valores relativos ao número de neurónios e número de *hidden layers* consideramos uma rede neuronal com as

seguintes características:

Input → 200 Neurónios → 100 Neurónios → Output

Os valores dos hiperparâmetros usado foram:

- **Número de *features* (termos do vocabulário):** 600
- **Número de *epochs*:** 200
- **Batch size:** 500
- **Activation function:**
  - **Hidden Layers:** *Relu* [5]
  - **Output Layer:** *Softmax* [6]
- **Optimizer:** *Adam* [7]
- **Loss function:** *Categorical crossentropy* [8]

Desde o início da implementação deste modelo que foi usado o processo de treino-validação utilizando o ***K-Fold Cross validation***. Este método tem um parâmetro *k* que define o número de divisões que o conjunto será sujeito, cada porção segmentada é intitulada de *fold*. No processo de treino todos os *folds* são usados para teste uma vez, sendo que os outros restantes *k-1 folds* são usados para o processo de treino. Neste estudo, o valor de *k* definido foi 5 (***K-fold* com 5 splits**). Ainda foi adicionado um *callback* de *Early Stopping* com objectivo de suspender o treino quando o parâmetro a ser monitorizado (neste caso o *validation accuracy*) para de melhorar.

#### B. Resultados obtidos

Após as condições definidas anteriormente decidimos então testar a performance do modelo. Obtivemos os seguintes valores de *accuracy*, após retreino do modelo (com todos os dados de treino e *cross validation*):

- **Train Score:** 0.82
- **Test Score:** 0.71

Como se pode observar pelos resultados e pelo gráfico 6, o modelo tem tendência a realizar o fenómeno de *overfit*, pelo que as próximas subsecções têm como objectivo explicar os mecanismos usados para evitar este fenómeno. Apenas de salientar que devido ao *callback* de *early stopping*, o treino é parado, pois a *accuracy* do conjunto de validação não aumenta, o que acaba por evitar uma maximização do fenómeno de *overfit*.

#### C. Melhoria do modelo

Para evitar o fenómeno de *overfit*, decidimos acrescentar uma função de regularização nos pesos, *L2 regularization penalty* [9], nas *hidden layers*. No fundo, a regularização evita o fenómeno de *overfit*, restringindo os valores dos pesos, evitando que o modelo se adapte demasiado à forma dos dados de treino e não consiga generalizar tanto em dados de teste. Para isso, o algoritmo de regularização tem um parâmetro que define o impacto da regularização nos termos. Caso o valor seja muito elevado, o modelo não tem flexibilidade suficiente para se formatar à forma dos dados, pelo que irá ocorrer o fenómeno de *underfit*. Caso seja muito baixo, o fenómeno de *overfit* pode ocorrer, pois o impacto do termo regularizador é fraco. Como se pode observar no gráfico 7, para valores muito altos do factor de regularização, o modelo não consegue

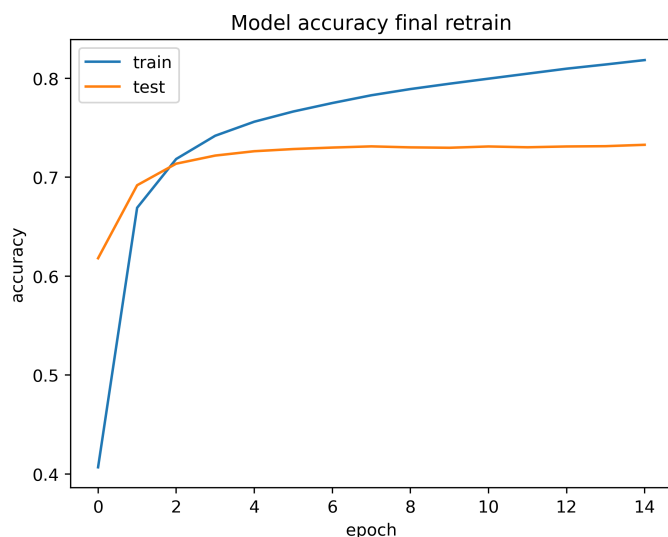


Fig. 6. Accuracy function após re-treino com o conjunto de treino na totalidade

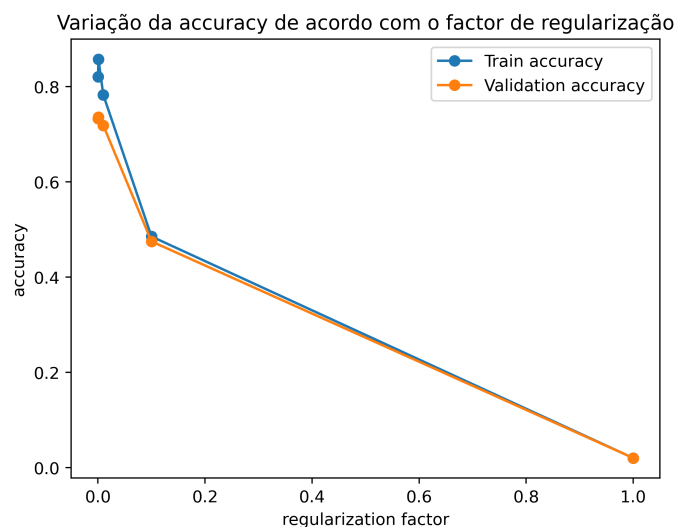


Fig. 7. Variação dos valores de accuracy de acordo com o factor de regularização

generalizar os dados. O que acabou por se tornar o melhor parâmetro foi o valor de 0.001, cuja diferença entre a accuracy do conjunto de validação e o conjunto de treino é menor.

Os valores após re-treino, usando o factor de regularização ótimo definido acima, são:

- **Train Score:** 0.82
- **Test Score:** 0.76

Como se pode observar no gráfico 8, houve uma melhoria relativamente à accuracy em dados nunca vistos pelo modelo, o que indica que o fenómeno de *overfit* diminui, o que consequentemente indica que o modelo obteve uma performance melhor.

#### D. Análise de sentimento como processo de complementação

Com a análise prévia dos textos fornecidos no *dataset* e com o conhecimento prévio da plataforma *Reddit*, foi observado que os textos têm alguma complexidade e variância no que toca à formalidade do mesmo. Para facilitar a compreensão de um determinado texto, foi usado um mecanismo capaz de qualificar um texto em positivo, negativo e neutro.

Após alguma pesquisa foi encontrado um repositório público de uma ferramenta para a finalidade descrita anteriormente - **VADER Sentiment Analysis** [10] baseando-se em regras léxicas e semânticas com intuito de retirar a expressão de sentimentos de dados provenientes de redes sociais.

Esta ferramenta aceita como dados de entrada qualquer sequência de termos alfa-numéricos e retorna algumas pontuações. Apesar dos variados valores de retorno foi usado apenas a seguinte pontuação - *compound score*:

- **Sentimento positivo:** pontuação  $\geq 0.05$
- **Sentimento neutro:**  $-0.05 < \text{pontuação} < 0.05$
- **Sentimento negativo:** pontuação  $\leq -0.05$

Como esta etapa foi usada como processo de complementação, este modelo foi aplicado a todos os

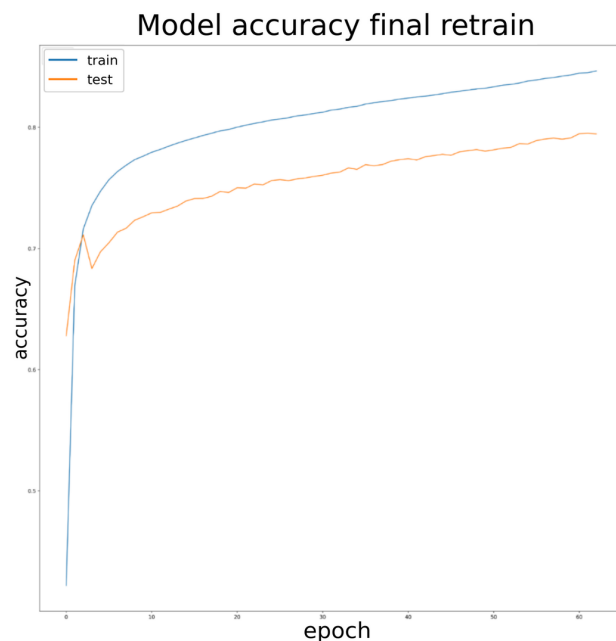


Fig. 8. Accuracy function após re-treino com o conjunto de treino na totalidade

textos do *dataset*, com objectivo de mapear um valor capaz de qualificar o sentimento dos textos. Essencialmente, o vector de *features* é composto pelos vectores de pontuações descritos na secção dos dados II-C mais a pontuação proveniente do método descrito.

Para efeitos de aprovação, o gráfico 9 apresenta os valores da accuracy do mesmo modelo de *Rede Neuronal* já apresentado durante esta secção, com e sem a *feature* que transmite o sentimento do texto. Como se pode observar, o modelo que dispunha da indicação do sentimento de texto obteve uma

melhor performance em termos de *accuracy* em ambos os conjuntos, o que indica que este valor é uma boa adição para a performance do modelo, já que ajuda a descodificar a parte semântica do texto que não é observável apenas com a noção de *tokens*.

#### E. Conclusões

Foram feitos esforços para tentar melhorar a performance deste modelo mas sem sucesso, muito devido à grande dificuldade de "*fine-tune*" das Redes Neurais, que muito facilmente convergem para o fenómeno de *overfit*.

### IV. ESTUDO COM REDES DE BAYES

**Redes de Bayes** é um modelo probabilístico que permite definir as relações de dependência entre determinadas variáveis e pode ser utilizado para calcular a probabilidade de ocorrência de determinados eventos associados a essas variáveis.

Ao pesquisarmos sobre outros estudos feitos de *machine learning* sob o mesmo *dataset* usado, encontramos a implementação [11]. Este utilizador do *Kaggle* conseguiu, para 5 classes, obter uma precisão de aproximadamente 0.72 e uma *accuracy* de aproximadamente 0.61<sup>5</sup> para os dados de teste.

#### A. Repartição dos dados

Para além da divisão descrita na secção II-B, fizemos uma segunda divisão nos dados reservados para treino/*cross validation*. Esta divisão foi de realizada de maneira a que obtivéssemos uma distribuição de 60%-20%-20% entre dados de treino, *cross validation* e teste, respectivamente. A distribuição final dos dados por *dataset* pode ser consultada nos gráficos da figura 10.

#### B. Resultados obtidos

Para obter os resultados que iremos descrever de seguida, adaptamos o código que encontramos no *Kaggle* deste autor e fizemos algumas alterações de forma a usar a nossa técnica de extracção de *features* dos dados, explicada na secção II-C, ao invés da utilizada pelo autor.

Desta forma, para 50 classes (ao invés das 5 usadas pelo autor original), obtivemos a variação de *accuracy* de acordo com a variação do valor de alfa<sup>6</sup> para os dados de treino e de *cross validation* apresentados no gráfico da figura 11. Como se pode verificar neste gráfico, foi obtido um valor máximo de *accuracy* = 0.845 para  $\alpha = 0.1$  quando submetemos o modelo aos dados de *cross validation*.

O nosso passo seguinte foi retrainar o modelo onde obtivemos maior valor de alfa, usando como dados de treino os dados previamente usados para treino e *cross validation*, ou seja, 80% dos dados usados. Os resultados das métricas de performance por classe e totais quando o modelo foi treinado nas condições referidas podem ser consultados na tabela I. Duma forma geral, podemos verificar que temos resultados aceitáveis, mas não extraordinários (a *accuracy* está longe de ser 100%).

<sup>5</sup>Apesar do autor não ter documentado esta métrica, nós executamos o código disponibilizado pelo mesmo, obtendo este valor, para 10 mil *features* (não podemos usar mais por falta de recursos físicos)

<sup>6</sup>*Learning rate*

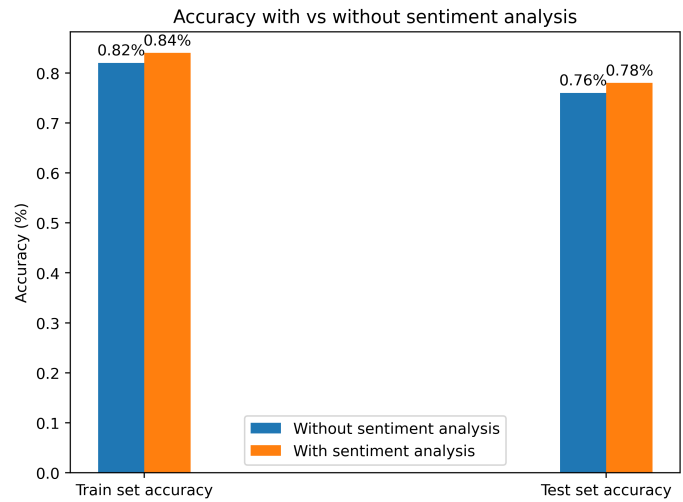


Fig. 9. *Accuracy* do mesmo modelo **com e sem** o valor que transmite o sentimento do texto

#### C. Conclusão

Apesar dos resultados finais obtidos terem sido aceitáveis, mas não extraordinários como demonstrado anteriormente, conseguimos obter um melhor valor de *accuracy* que o autor da solução apresentada no início desta secção (aproximadamente 0.61, para 5 classes, que ele obteve e 0.853, para 50 classes, que nós obtivemos). Sendo que a maior alteração que fizemos à sua solução foi a utilização de um pré-processamento e selecção de *features* distinta, concluímos que terá sido essa a chave para o nosso maior sucesso.

### V. ESTUDO COM REGRESSÃO LOGÍSTICA

A **Regressão Logística** é um algoritmo que permite, estatisticamente e a partir dum conjunto de observações de eventos passados, criar um modelo que possibilite a previsão de valores associados a um possível evento (caracterizado por um conjunto de variáveis).

Apesar de termos encontrado o trabalho apresentado na secção IV, encontramos também outros trabalhos que se insidiam na catalogação de texto, mas que não utilizavam o *dataset* que usámos. Um destes trabalhos foi um criado por um outro utilizador do *Kaggle*, em que usava, entre outros algoritmos, **Regressão Logística** para catalogar o tema de notícias da *BBC*. O seu estudo pode ser encontrado em [12]. Neste estudo é notado que foi atingida uma *accuracy* para os dados de teste de 0.79 para as 20 classes em estudo.

#### A. Repartição dos dados

A divisão dos dados neste estudo foi a mesma usada descrita na secção IV-A.

#### B. Resultados obtidos

Tal como na secção IV-B, começamos por adaptar o código do autor original. Neste caso, sendo que ele possuía vários algoritmos para realizar o seu estudo, foi necessário filtrar só

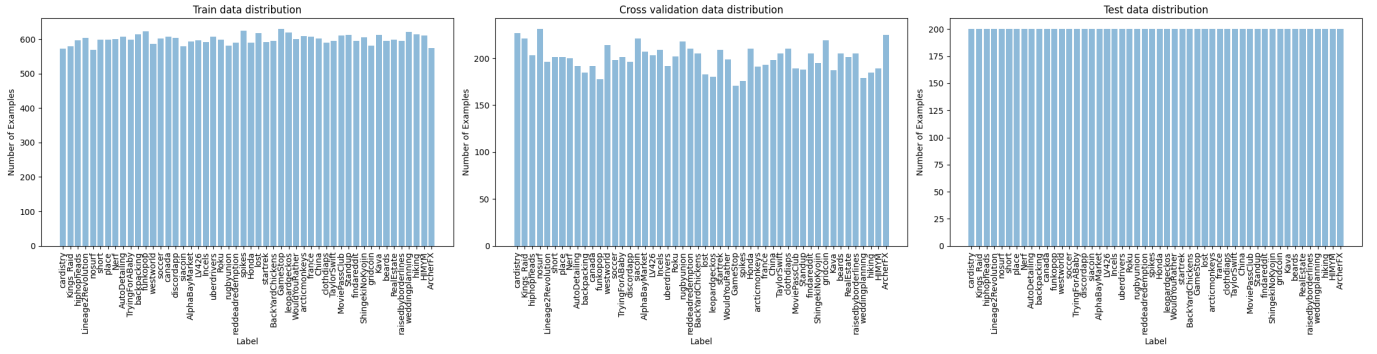


Fig. 10. Distribuição de dados por *label* por *dataset*, havendo discriminação entre dados de treino e de *cross validation*.

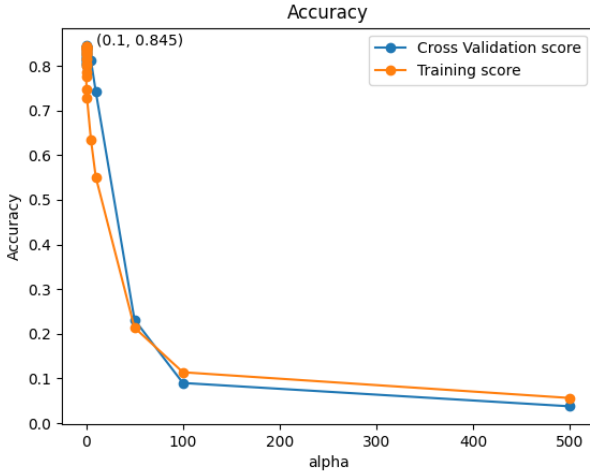


Fig. 11. Gráfico da variação da *accuracy* para valores de  $\alpha$  entre  $1 \cdot 10^{-10}$  e  $5 \cdot 10^2$ , para os dados de treino e de *cross validation*

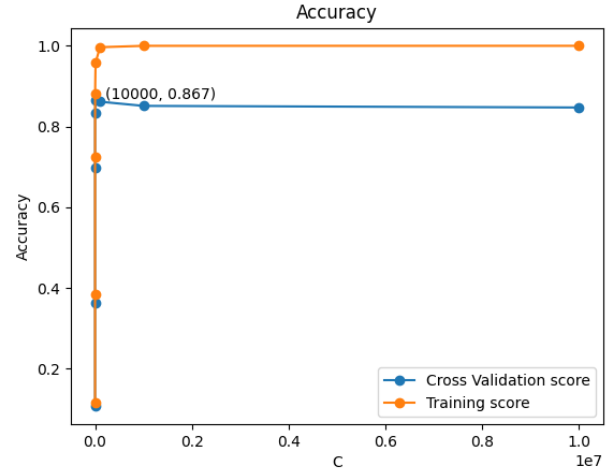


Fig. 12. Gráfico da variação da *accuracy* para valores de  $C$  entre  $10^0$  e  $10^7$ , para os dados de treino e de *cross validation*

o código correspondente ao estudo de *Regressão Logística*. Tal como no estudo das Redes de Bayes, neste caso usamos também a técnica de extração de *features* explicada na secção II-C ao invés da usada pelo autor.

Ao fazermos o estudo da variação do hiperparâmetro  $C$ <sup>7</sup>, para 5000 iterações com *early stopping*, obtivemos a variação de *accuracy* apresentada no gráfico da figura 12. Com este gráfico pode-se concluir que o maior valor da *accuracy* para os dados de *cross validation* foi obtido para um valor de  $C = 10^4$ .

Desta forma, o nosso passo seguinte foi retrainar o melhor modelo obtido ( $C = 10^4$ ) com os dados usados previamente para treino e *cross validation*. As métricas de performance obtidas quando submetido o modelo final aos dados de teste podem ser consultadas na tabela II.

### C. Conclusão

Tal como na secção IV, conseguimos neste exemplo obter uma performance melhor que a do autor do estudo original.

<sup>7</sup>Parâmetro de regularização. Quanto maior o seu valor, maior será a *regularization strength*

Enquanto que este obteve uma *accuracy* de teste de aproximadamente 0.79 para apenas 20 classes, nós conseguimos obter uma *accuracy* de aproximadamente 0.876.

## VI. CONCLUSÕES

Finalizado este estudo, foi nos possível consolidar o nosso conhecimento relativamente aos algoritmos mais populares usados em *Machine learning*, mais especificamente em processos de classificação multi-classe.

O foco deste trabalho foi fazer a classificação de textos de *posts* da plataforma *Reddit*. Inicialmente pensávamos que a iniciação deste trabalho fosse mais fácil na medida em que este se trata de um problema de *text classification* e existe muita informação relativa a este tópico. Após algumas tentativas de aplicar os algoritmos mais genéricos de *text classification*, apercebemos-nos que estes não tinham uma boa performance, justificável pela imprevisibilidade dos textos no que toca à formalidade, o que torna a obtenção de *features* mais difícil.

Numa fase inicial o esforço foi todo focado na etapa de pré-processamento dos textos, pois esta é bastante importante



TABLE I  
MÉTRICAS DE PERFORMANCE PARA  $\alpha = 0.1$

Class	Accuracy	Recall	Precision	F1 Score
0	0.855	0.855	0.905	0.879
1	0.79	0.79	0.903	0.843
2	0.885	0.885	0.868	0.876
3	0.92	0.92	0.944	0.932
4	0.66	0.66	0.75	0.702
5	0.83	0.83	0.79	0.81
6	0.88	0.88	0.907	0.893
7	0.915	0.915	0.88	0.897
8	0.78	0.78	0.729	0.754
9	0.95	0.95	0.931	0.941
10	0.835	0.835	0.879	0.856
11	0.92	0.92	0.92	0.92
12	0.805	0.805	0.92	0.859
13	0.935	0.935	0.921	0.928
14	0.88	0.88	0.846	0.863
15	0.895	0.895	0.937	0.916
16	0.965	0.965	0.889	0.926
17	0.92	0.92	0.953	0.936
18	0.845	0.845	0.837	0.841
19	0.815	0.815	0.867	0.84
20	0.905	0.905	0.896	0.9
21	0.945	0.945	0.926	0.936
22	0.84	0.84	0.853	0.846
23	0.72	0.72	0.623	0.668
24	0.965	0.965	0.928	0.946
25	0.64	0.64	0.715	0.675
26	0.875	0.875	0.888	0.882
27	0.97	0.97	0.965	0.968
28	0.965	0.965	0.858	0.908
29	0.695	0.695	0.675	0.685
30	0.685	0.685	0.811	0.743
31	0.79	0.79	0.81	0.8
32	0.94	0.94	0.935	0.938
33	0.805	0.805	0.782	0.793
34	0.81	0.81	0.72	0.762
35	0.945	0.945	0.945	0.945
36	0.865	0.865	0.779	0.82
37	0.895	0.895	0.836	0.865
38	0.85	0.85	0.944	0.895
39	0.91	0.91	0.74	0.816
40	0.925	0.925	0.845	0.883
41	0.76	0.76	0.694	0.726
42	0.825	0.825	0.864	0.844
43	0.925	0.925	0.916	0.92
44	0.67	0.67	0.753	0.709
45	0.93	0.93	0.964	0.947
46	0.815	0.815	0.795	0.805
47	0.87	0.87	0.93	0.899
48	0.79	0.79	0.849	0.819
49	0.845	0.845	0.96	0.899
Macro Average	0.853	0.853	0.856	0.853

TABLE II  
MÉTRICAS DE PERFORMANCE PARA  $C = 10^4$

Class	Accuracy	Recall	Precision	F1 Score
0	0.9	0.9	0.893	0.998
1	0.805	0.805	0.957	0.895
2	0.85	0.85	0.949	0.902
3	0.9	0.9	0.965	0.954
4	0.795	0.795	0.909	0.858
5	0.91	0.91	0.917	0.92
6	0.895	0.895	0.986	0.929
7	0.91	0.91	0.964	0.993
8	0.815	0.815	0.868	0.829
9	0.95	0.95	0.996	0.983
10	0.785	0.785	0.943	0.901
11	0.935	0.935	0.985	0.976
12	0.815	0.815	0.925	0.997
13	0.94	0.94	0.985	0.961
14	0.835	0.835	0.85	0.879
15	0.865	0.865	0.957	0.943
16	0.96	0.96	0.988	0.992
17	0.91	0.91	0.987	0.991
18	0.855	0.855	0.991	0.891
19	0.875	0.875	0.967	0.911
20	0.895	0.895	0.955	0.897
21	0.96	0.96	0.944	0.941
22	0.845	0.845	0.8	0.947
23	0.775	0.775	0.928	0.764
24	0.955	0.955	0.992	0.999
25	0.73	0.73	0.896	0.885
26	0.87	0.87	0.911	0.925
27	0.96	0.96	0.995	0.983
28	0.935	0.935	0.897	0.968
29	0.845	0.845	0.868	0.942
30	0.82	0.82	0.983	0.969
31	0.875	0.875	0.933	0.889
32	0.91	0.91	0.98	0.983
33	0.84	0.84	0.92	0.972
34	0.85	0.85	0.946	0.837
35	0.955	0.955	0.981	0.986
36	0.905	0.905	0.872	0.87
37	0.89	0.89	0.918	0.987
38	0.87	0.87	0.954	0.884
39	0.89	0.89	0.92	0.998
40	0.905	0.905	0.896	0.921
41	0.745	0.745	0.892	0.837
42	0.835	0.835	0.98	0.896
43	0.9	0.9	0.977	0.949
44	0.88	0.88	0.986	0.899
45	0.945	0.945	0.989	0.964
46	0.835	0.835	0.974	0.957
47	0.915	0.915	0.959	0.941
48	0.865	0.865	0.932	0.984
49	0.905	0.905	0.985	0.978
Macro Average	0.876	0.876	0.943	0.933

para a performance geral do modelo. Nesta etapa é feita a transformação de texto no formato "raw" em vectores numéricos capazes de transmitir informações sobre os textos. Num problema deste tipo, a dimensão das *features* de entrada é igual ao tamanho do vocabulário definido para o *corpus* em questão. Logo, devido a isto, foi necessário fazer a segmentação do vocabulário original, pois este atinge dimensões nas ordens dos milhões, o que acaba por ser um número elevado de *features* de entrada do modelo.

Foi possível concluir que especificamente para este *dataset*, o algoritmo que obteve melhor performance foi o de *Logistic Regression*. Contudo, é de ter em atenção que o treino de

modelos que usem este algoritmo para uma grande quantidade de *features* e de dados de treino se torna penosamente lento. Por exemplo, para o melhor modelo que obtivemos, o modelo demorou aproximadamente 4 horas a ser treinado, o que em algumas implementações pode não ser o mais aceitável. Sendo assim, os modelos de redes neuronais e de redes de *bayes* apresentados, apesar de não terem tido uma prestação tão boa, podem ter uma utilização mais aceitável em circunstâncias em que o tempo de treino tenha de ser reduzido (estes dois modelos demoraram entre 2 a 3 minutos a serem treinados).

Neste trabalho reforçou-se ainda mais a ideia que a performance de um modelo de *machine learning* está muito



relacionada com a qualidade e processamento do *dataset*, daí que muita das vezes um esforço inicial seja para regularizar e otimizar o conteúdo do *dataset*, de forma a que o modelo consiga tirar partido da qualidade de selecção de *features* e ecossistema do *dataset*. Este facto provou-se na prática neste trabalho quando, se compara o uso de dois algoritmos (redes de *bayes* deste relatório e rede de *bayes* de um *paper*) iguais com processamentos de dados diferentes, o esforço colocado nesse aspecto neste trabalho trouxe benefícios na performance geral do modelo.

## VII. AGRADECIMENTOS

Queremos aqui deixar um agradecimento especial ao doutor engenheiro Mário Antunes do Instituto de Telecomunicações, que se demonstrou sempre disponível para esclarecer algumas dúvidas que fomos colocando e fez questão de acompanhar a evolução do nosso projecto.

## REFERENCES

- [1] D. Silva, P. Escalreira, P. Oliveira, and R. Simões, “Minerva,” June 2020. [Online]. Available: <https://detiuaveiro.github.io/social-network-mining/code/public-portfolio/index.html>
- [2] M. S. Jones, “The reddit self-post classification task (rspct) : ahightly multiclass dataset for text classification (preprint),” Oct 2018. [Online]. Available: [https://evolution.ai/blog\\_figures/reddit\\_dataset/rspct\\_preprint\\_v3.pdf](https://evolution.ai/blog_figures/reddit_dataset/rspct_preprint_v3.pdf)
- [3] B. Góralewicz, “The tf\*idf algorithm explained,” Mar 2018. [Online]. Available: <https://www.onely.com/blog/what-is-tf-idf/>
- [4] V. Satopaa, J. Albrecht, D. Irwin, and B. Raghavan, “Finding a “kneedle” in a haystack: detecting knee points in system behavior.” [Online]. Available: <https://sustainablecomputinglab.org/wp-content/uploads/2014/09/simplex.pdf>
- [5] “Keras layer activation functions.” [Online]. Available: <https://keras.io/api/layers/activations/#relu-function>
- [6] “Keras layer activation functions.” [Online]. Available: <https://keras.io/api/layers/activations/#softmax-function>
- [7] “Keras optimizers.” [Online]. Available: <https://keras.io/api/optimizers/adam/>
- [8] “Keras loss functions.” [Online]. Available: [https://keras.io/api/losses/probabilistic\\_losses/#categorical\\_crossentropy-class](https://keras.io/api/losses/probabilistic_losses/#categorical_crossentropy-class)
- [9] “Keras layer weight regularizers.” [Online]. Available: <https://keras.io/api/layers/regularizers/#l2-class>
- [10] “Valence aware dictionary and sentiment reasoner.” [Online]. Available: <https://github.com/cjhutto/vaderSentiment>
- [11] M. S. Jones, “The reddit self-post classification task (rspct) : ahightly multiclass dataset for text classification (preprint),” Oct 2018. [Online]. Available: [https://evolution.ai/blog\\_figures/reddit\\_dataset/rspct\\_preprint\\_v3.pdf](https://evolution.ai/blog_figures/reddit_dataset/rspct_preprint_v3.pdf)
- [12] Balatmak, “Text classification pipeline newsgroups20,” May 2019. [Online]. Available: <https://www.kaggle.com/balatmak/text-classification-pipeline-newsgroups20>