Língua Natural Mini - Projeto Nº 2
Grupo 17

Luís Carlos Silva Aguiar 80950 , Pavlo Kovalchuk 81440

1 Introdução

O segundo projeto da cadeira de Língua Natural tem com objetivo desenvolver uma métrica de similaridade que permita identificar o tipo de uma nova questão sobre cinema baseando-se em questões já conhecidas. Para resolver este problema nós exploramos 2 abordagens, a primeira é baseada no calculo da similaridade das novas questões com as questões conhecidas utilizando o ***cosine similarity***, a segunda abordagem é baseada na utilização de algoritmos de **aprendizagem supervisionada**. Ambas as abordagens vão ser descritas com mais detalhe nas próximas secções.

2 Proposta de Solução

2.1 Pré-Processamento

Para ambos os conjuntos de questões quer as questões conhecidas quer as novas questões foram pré-processadas foi aplicada a mesma função de pré-processamento.
Para cada questão fornecida são removidos possíveis caracteres especiais, pontuação, palavras com apenas um caráter no meio e ou inicio das questões, e são também removidas *stop words*, cada palavra da frase é convertida para letra minúscula, e por ultimo cada palavra é convertida no seu lema. O calculo do lema é realizado com o método ***WordNetLemmatizer*** [1], o lema de cada palavra é obtido através do apoio de uma *WordNet* em inglês. A remoção de pontuação e de caracteres especiais é feita com recurso as expressões regulares [2] do *Python*. Esta função de processamento faz com que o numero de palavras extraídas seja mais reduzido, e as palavras que restam são consideras relevantes para uma determinada questão.

2.2 Representação dos Dados

De forma a conseguir aplicar os algoritmos propostos foi necessário converter as questões fornecidas e os seus respetivos tópicos para uma representação numérica. Para tal consideramos a representação de cada questão (quer as conhecidas, quer as novas questões) como um vetor de pesos (***Vector Space Model***). Para o cálculo dos pesos do vetor de cada questão considerámos duas abordagens, a primeira foi com a utilização do método ***CountVectorizer*** [3], com este método cada peso do vetor corresponde ao numero de vezes que cada palavra ocorre numa determinada questão.
Como segunda abordagem consideramos calcular cada peso do vetor segundo a formula do ***Term Frequency–Inverse Document Frequency*** (TfIdf) com

método ***TfidfVectorizer*** [3], de seguida segue-se a formula utilizada para o TfIdf

$$w_{i,j} = tf_{i,j} \times log\frac{|D|}{\{d' \epsilon D | t \epsilon d'\}}$$

onde:

- $\mathbf{w}_{i_j}$ - Representa o peso para cada palavra numa determinada questão.
- $\mathbf{tf}_{i,j}$ (Frequência do Termo) - Numero de vezes em que cada palavra ocorre numa determinada questão.
- **log** $\times \frac{|D|}{\{d' \epsilon D | t \epsilon d'\}}$ (Inverso da frequência) - Inverso da ocorrência de um determinado termo em toda a coleção de questões.
- $|D|$ - Numero total de questões na coleção.

Para determina a melhor representação testamos com o classificador ***KNeighborsClassifier*** [5] (com os parâmetros *default*), e verificamos que para o ***CountVectorizer*** obtemos uma *accuracy* de 85.7% e para o ***TfIDFVectorizer*** *accuracy* de 92.8%.
Assim concluímos que o melhor método para representar as questões sobre a forma de um vetor de pesos é o ***TfIDFVectorizer*** com os valores normalizados.
Por ultimo foi necessário converter cada tópico nominal para numérico, para tal utilizamos o método ***LabelEncoder*** [6], onde a cada tópico é atribuído um numero de 0 a 15.

2.3 Solução 1: Similaridade entre novas questões e questões conhecidas

Esta abordagem consiste em converter cada questão conhecida e nova questão sobre a forma de um vetor de pesos através do método ***TfIDFVectorizer***. De seguida é calculada a similaridade de cada nova questão com cada questão conhecida através da formula do ***cosine similarity*** [6], a similaridade é um valor real que varia entre 0 e 1, onde 0 é nada semelhante e 1 é totalmente semelhante.
Por ultimo para atribuir novos tópicos a cada nova questão, basta apenas escolher a questão conhecida que tem maior similaridade e atribuir esse tópico à nova questão, no caso de empate escolhemos o tópico que aparece mais vezes de entre os candidatos.
Para avaliar este método foram calculados as métricas de: *accuracy*, *precision*, *recall* e *f-scores* sobre a matriz de confusão.

2.4 Solução 2: Aprendizagem Supervisionada

Nesta segunda abordagem vamos desenvolver um classificador automático recorrendo algoritmos de aprendizagem supervisionada.
Tal como na abordagem anterior o primeiro passo é converter cada questão de treino e de teste para a forma de vetor de pesos através do método ***TfIDFVectorizer***.
De seguida criamos as listas *X_train*, *y_train* e *X_test* onde a lista *X_train* corresponde aos vetores de pesos de cada questão conhecida e a lista *y_train* corresponde ao tipo de questão correspondente a cada questão conhecida a estas duas listas vamos passar a chamar treino, a lista *X_test* corresponde aos vetores de pesos para cada nova questão, e a esta lista vamos passar a chamar teste.
Para esta vamos utilizar o ***KNeighborsClassifier*** [4] como classificador. De seguida cada classificador foi treinado com as listas de treino com o método *fit*, para obter os novos tópicos para as novas questões é aplicado o método *predict* de cada um dos classificadores para cada nova questão, o método *predict* vai retornar uma lista de tópicos para cada nova questão.

Para avaliar o desempenho de cada classificador calculamos as seguintes métricas: *accuracy*, *precision*, *recall* e *F-scores* sobre a matriz de confusão.

3 Resultados Obtidos

3.1 Resultados Solução 1

Para as novas questões fornecidas pelo corpo de docente da cadeira obtivemos o seguintes resultados para cada métrica: *Accuracy* - 88.09%, *Precision* - 65.38%, *Recall* - 57.53%, *F-Score* - 60.66%, e a seguinte matriz de confusão:

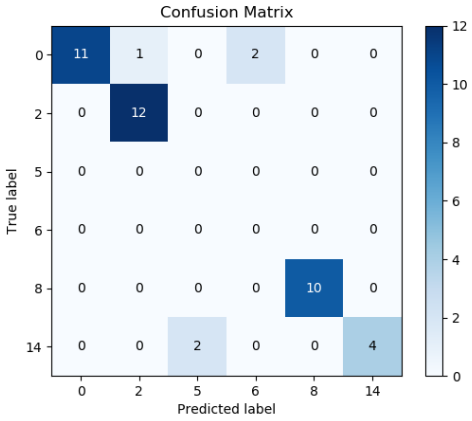

Figure 1: Matriz de Confusão Solução 1

3.2 Resultados Solução 2

Para a segunda abordagem inicialmente mantivemos os parâmetros do classificador ***KNeighborsClassifier*** e do ***TfIDFVectorizer*** como *default*, e obtivemos os seguintes resultados para as novas questões fornecidas: *Accuracy* - 97.61%,*Precision* - 80%, *Recall* - 78%, *F-Score* - 78.94%.
De forma a melhorar os nosso resultados, no método ***TfIDFVectorizer*** colocamos os parâmetros *max_features* = 115, *min_df*=5, *max_df*=0.7, onde reduzimos o efeito de *overfitting* reduzindo o numero de features, e apenas consideramos termos que ocorrem no máximo em 70% das questões e no mínimo em 5. De seguida para determinar o melhor valor para o parâmetro *n_neighbors* decidimos testar vários valores e através do **Figura 2** podemos concluir que o melhor valor para *n_neighbors* é 3. Após defenir *n_neighbors* com o valor 3 obtivemos a matriz de confusão da **Figura 3** e *Accuracy* - 100%,*Precision* - 100%, *Recall* - 100%, *F-Score* - 100%

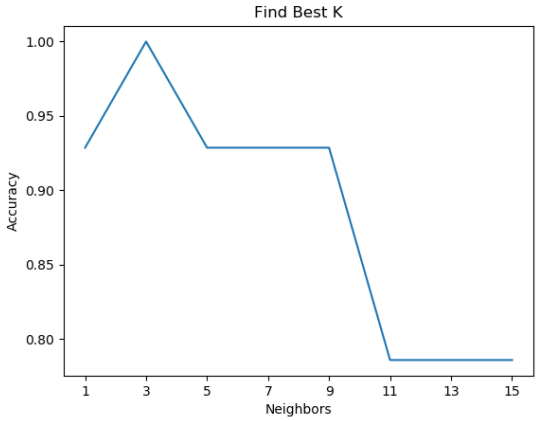

Figure 2: Variação da Accuracy com a variação do numero de vizinhos

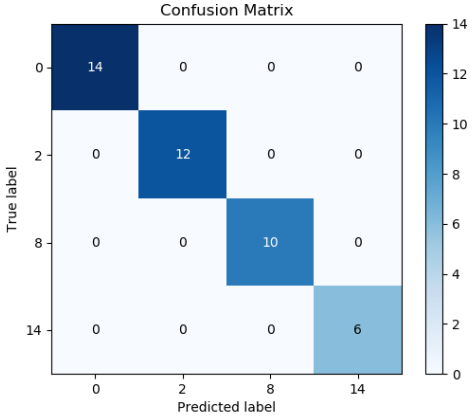

Figure 3: Matriz de Confusão Solução 2

4 Conclusões e trabalho futuro

Após testarmos ambas as abordagens propostas chegamos à conclusão que a abordagem baseada em algoritmos de aprendizagem supervisionada é a que apresenta melhores resultados. O método baseado na similaridade apenas funciona bem quando as instâncias de teste e treino são muito semelhantes, quando a tarefa é classificar uma instância nunca antes vista no treino este método apresenta resultados bastante baixos. Por outro lado o ***KNeighborsClassifier*** baseia-se na distância entre instâncias de teste e de treino para a atribuição de novos tópicos, mas no caso de serem novas instâncias este método tende a ser melhor, pois ele infere a nova classe a partir dos K vizinhos mais próximos isto faz com que o erro associado à classificação seja inferior. Como trabalho futuro nós sugerimos experimentar balançar os dados de treino utilizando técnicas como o **SMOTE** isto ira melhorar consideravelmente o desempenho do nosso classificador para todas as instâncias de teste. Também sugerimos que sejam aplicadas tecnicas de redução de dimensão tal como o ***Principal Component Analysis*** (PCA), pois o numero de *features* é superior ao numero de instâncias de treino, isto faz com o classificador fique *overfitted*, por ultimo seria interessante experimentar com outros modelos de classificação como por exemplo as **redes neuronais**.

5 Bibliografia

[1] - ***nltk.stem***: https://www.nltk.org/api/nltk.stem.html
[2] - ***re***: https://pypi.org/project/regex/
[3] - ***sklearn.feature_extraction.text***: http://scikit-learn.org/stable/modules/feature_extraction.html
[4] - ***sklearn.neighbors***:
http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html
[5] - ***sklearn.preprocessing***: http://scikit-learn.org/stable/modules/preprocessing.html
[6] - ***sklearn.metrics.pairwise***:
http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html

As matrizes de confusão foram obtidas através da bilioteca de:
https://scikitplot.readthedocs.io/en/stable/metrics.html, e o gráfico da bilioteca de https://matplotlib.org/, as metricas utilizadas para avaliar as abordagens foram calculadas utilizando a bilioteca: http://scikit-learn.org/stable/modules/classes.html