Sentence Pair Classification

Bejenariu David-Cosmin
group 407

I. Introduction

Sentence Pair Classification is the first assignment of the Practical Machine Learning course. The task is hosted as a Kaggle competition and consists of training a machine learning model that labels sentence pairs from 0 to 3 by their similarity.

II. Working with the given dataset

The dataset contains a total of 64173 entries of which 58114 make the training set, 3059 the validation set and the rest of 3000 are used for testing. Each entry is in the form of a JSON containing the two sentences, along with their corresponding label (except for the testing dataset) and a unique guid.

The training dataset is split as follows:

- Class 3: **28500** occurrences
- Class 2: **25722** occurrences
- Class 1: **1300** occurrences
- Class 0: **2592** occurrences

This set is obviously unbalanced. Given the evaluation method of the competition where the Macro F1-score metric is used for ranking, focusing on the 2 most significant classes is not a good strategy. Thus, I tested two approaches for providing balance between classes:

1. **Reduce the dataset**

The first and simplest approach was to reduce the occurrences of the two most significant classes to roughly about 10-15k samples.

2. **Augment the dataset**

Instead of deleting samples of the two most frequent classes, I added more samples to the least frequent ones. An initial idea which would have performed better on an English dataset was to use **SynonymAugmenter** from *nlpaug* to add more samples that would use the existing ones where randomly chosen words would have been replaced with their synonyms. Given the fact that the

dataset consists of sentences in Romanian language, I opted for using the existing samples where random words have been swapped to generate more samples.

III. Data preprocessing

The data preprocessing combined the general techniques used when working with text: **lowercase conversion**, **stop words/special characters/punctuation removal**, **lemmatization** and **tokenization**.

The process of turning the actual text into features followed two main paths:

1. **Word embeddings**

The training sentence pairs were space concatenated into single sentences which were then fed to a **Word2Vec** vectorizer from *gensim.models* library. From here, the three datasets were processed by the vectorizer and the resulting features were either passed further to training/testing the model or to an additional processing step that would calculate the cosine similarity between each pair of sentences.

Cosine similarity is calculated by measuring the cosine of the angle between two vectors. It quantifies the similarity of their directions, ranging from -1 (dissimilar) to 1 (similar), with 0 indicating orthogonality. The formula involves the dot product and the magnitudes of the vectors. Higher cosine similarity implies greater similarity between the vectors or documents.

2. **Bag of words**

The concatenated sentences were processed into features using **TF-IDF** vectorizer from the *scikit-learn* library. TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer is a numerical representation of text documents, that assigns weights to words based on their frequency in a document and rarity across the entire corpus, capturing the importance of words in distinguishing documents.

Each pair of sentences was processed into feature vectors of about 225k elements. Given the large number of features that would influence the computation time and resources, setting the *max_features* property of the vectorizer to a smaller amount was taken into consideration.

IV. Model training

As a last preparation step before feeding the training feature vectors into a model, I took scaling the numerical features into consideration. **Scaling** is a preprocessing step that transforms numerical data to a common scale, preventing features with larger magnitudes from dominating. For that, I opted for **Standard Scaler**, which standardizes numerical features by subtracting the mean and dividing by the standard deviation. This transformation ensures that the scaled features have zero

mean and unit variance, bringing all features to a common scale and aiding machine learning algorithms that are sensitive to the scale of input features.

As the focus of this project turned out to be the data preprocessing step, I did not proceed to train more complex models such as neural networks and made use of two different classifying algorithms: **Random Forest** and **Support Vector Machines**. For hyperparameter tuning I ran a **Grid search** for each type of model, adjusting the *n_estimators* and *criterion* parameters for Random Forest and the *kernel*, *C* and *gamma* parameters for SVMs.

```
grid_params = {'n_estimators': [10, 50, 100], 'criterion': ['gini',
'entropy']}
rf_model = RandomForestClassifier(n_jobs=-1)

grid_search = GridSearchCV(rf_model, grid_params, cv=5, scoring='f1_macro',
verbose=1)
grid_search.fit(x_train, y_train)
```

```
grid_params = {'C': [0.1, 1, 10], 'gamma': ['auto', 'scale'], 'kernel':
['linear', 'rbf']}
svc_model = SVC(class_weight=class_weights)

grid_search = GridSearchCV(svc_model, grid_params, cv=5, scoring='f1_macro',
verbose=1)
grid_search.fit(x_train, y_train)
```

SVM training also used class weights which were computed by the **compute_class_weight** function from *sklearn.utils*, to provide more balance between the 4 classes.

V. Results

Classification reports and confusion matrices are as follows:

- **Approach 1**: Data augmentation + Word2Vec vectorizer + Random Forest Classifier

Confusion Matrix:

0	0	63	11
0	0	61	11
0	0	970	165
0	0	1079	699

Classification Report:

	Precision	Recall	F1-score	Support
0	0.00	0.00	0.00	74
1	0.00	0.00	0.00	72
2	0.45	0.85	0.59	1135
3	0.79	0.39	0.52	1778
Accuracy	-	-	0.55	3059
Macro avg	0.31	0.31	0.28	3059
Weighted avg	0.62	0.55	0.52	3059

- **Approach 2:** TF-IDF vectorizer + SVC

Confusion Matrix:

2	0	62	10
1	6	51	14
3	6	942	184
5	2	710	1061

Classification Report:

	Precision	Recall	F1-score	Support
0	0.18	0.03	0.05	74
1	0.43	0.08	0.14	72
2	0.53	0.83	0.65	1135
3	0.84	0.83	0.70	1778
Accuracy	-	-	0.66	3059
Macro avg	0.50	0.38	0.38	3059
Weighted avg	0.70	0.66	0.65	3059

Relevant notes:

- The second approach (TF-IDF + SVM) had the **highest score** (0.60188 on the full testing dataset) with best parameters: C=1, kernel='rbf', gamma='scale, no data augmentation or scaling and minimum text preprocessing (lowercasing, special character removal, tokenization).
- The other approach that used word embeddings along with Random Forest classification resulted in a significantly lower performance (0.3 score on the testing dataset).
- Cosine similarity is a valid measure for sentence pair classification task. Although, in this context, a single feature was not enough to train an accurate model. The more features, the better (I could not find a threshold where the model would have started overfitting).
- It was a good idea to simply concatenate each pair of sentences and extract the features using the trained vectorizer.
- Other combinations between the two feature extraction methods and the two classification algorithms did not perform any better.
- It was difficult to find the right balance for augmentation on the training dataset and the validation results indicated more false positives on the less relevant classes than expected. Finally, not tuning the dataset was the most effective approach. Reducing the dataset was not a good practice either.