NLP Project 2 - Text Classification

Group 4: Alexandre Pires - 92414, Diogo Fouto - 93705

November 2021

1 Models

For this project, we developed two text-classification models: one based on the Naive-Bayes algorithm, our baseline, and a Linear Support Vector Machine. Both do text pre-processing.

1.1 Baseline: Naive-Bayes

Our first model consists of a pure – from scratch – implementation of the Naive-Bayes Classifier, as described in [1], with text pre-processing. We chose this as the baseline because it is a straightforward approach that produces respectable results.

The training and test data pre-processing consists of reorganizing the sentences as word arrays (each array is associated with a sentence and its respective category), removing the stop-words from the arrays, and lemmatizing the remaining words.

To train the algorithm, we feed it the pre-processed word arrays. This generates a set of probabilities for each word given a certain category (conditionals) and a set of probabilities for each category (priors). To get predictions, we feed the test set to the algorithm and it returns the most probable category for each sentence.

A noticeable flaw of this model is the lack of use of the "Answer" column present in the text files. We decided to leave the model as it is, though, to focus our attention on a more promising model.

1.2 Our best model: Linear Support Vector Machine

A "Linear Support Vector Machine" is a supervised machine learning algorithm used to find solutions to classification problems. As input, it takes data points with N attributes – N-dimensional vectors – and builds a hyperplane that divides the input into M distinct groups – the possible classifications for the input. Our model is based on such an algorithm.

The training and test data pre-processing consists of lowercasing each line of the datasets, removing its stopwords, and vectorizing it. Lemmatization and stemming yielded worse results, so we discarded them.

To train the algorithm – build the aforementioned hyperplane – we fit it to the pre-processed training data. Then, to get predictions, we feed the pre-processed test set to the algorithm and it returns the groups – out of the M distinct ones mentioned before – to which the sentences belong.

2 Experimental Setup

From the beginning, it was clear that a Naive-Bayes algorithm would be a good baseline, since it was simple to understand and easy to implement.

We added two text pre-processing techniques to the algorithm. First, stop-word removal: we run through each word array of the training and test sets and apply the *nltk*'s "english" stop-words set as a filter. Then, lemmatization: we feed the remaining words to the *nltk*'s *WordNet Lemmatizer*.

For our SVM model, we decided on the *LinearSVC* algorithm – support vector classifier with linear kernel – present in the *sklearn* library. Among the possibilities – support vector classifiers with sigmoid, polynomial, linear, or gaussian kernels – this was the one that yielded better results in both speed and accuracy.

To pre-process text in this algorithm, we convert the training text file to a dataframe (using the *Pandas* library) with the features "Category", "Question", "Answer", "Content", and "Noise". ("Content" is the concatenation of the "Question" column with two identical "Answer" columns; doubling the weight of the "Answer" column has resulted in a better accuracy.) (The "Noise" feature was created to accommodate possible whitespace trailing each line in the *.txt* file.) Then, we lowercase each row of the "Content" column, remove its stop-words (the *NLTK* "english" stop-words), and vectorize it using tf-idf for each word (*sklearn*'s *TfidfVectorizer* does all of this). The test file is, too, converted to a dataframe, with the exception that the "Content" column in the test dataframe is a simple concatenation of the "Question" and "Answer" columns.

While comparing an early implementation of the SVM model with the Naive-Bayes algorithm, it became clear to us that further text pre-processing and method-parameter tweaking in the SVM could be made in order to maximize the algorithm's performance. To address these, we disabled tf-idf vectorization smoothing and doubled the weight of the "Answer" column in the training set dataframe. (This resulted in the "Content" column described in the previous paragraph.) We also tested the *CountVectorizer* provided by *sklearn*, but it fell short of the *TfidfVectorizer*'s results.

The metrics we implemented are: Recall, Precision, F1-Score, Jaccard, and overall classification accuracy. Precision and Recall were essential. They helped us: understand if a model was over or under predicting a certain category; identify problems that generally occur in limited-sized training sets; and sug-

gest code implementation mistakes. F1-Score was extremely helpful, since it combines Precision and Recall. Jaccard was implemented to help us understand how much of each category we were correctly predicting. Lastly, overall accuracy was helpful to assess each model's average performance.

3 Results

Model	Train Accuracy	Test Accuracy
Naive-Bayes	0.9428	0.8520
SVM	0.9946	0.9000

Table 1: Accuracy of each model

Category	Recall	Precision	F1	Jaccard
Geography	0.85	0.73	0.84	0.64
History	0.87	0.81	0.89	0.70
Literature	0.87	0.87	0.93	0.77
Music	0.78	0.87	0.93	0.72
Science	0.92	0.91	0.95	0.84

Table 2: Metrics for Naive-Bayes

Category	Recall	Precision	F1	Jaccard
Geography	0.88	0.81	0.84	0.73
History	0.90	0.86	0.88	0.78
Literature	0.90	0.95	0.93	0.86
Music	0.89	0.92	0.90	0.82
Science	0.92	0.93	0.93	0.86

Table 3: Metrics for Linear Support Vector Machine

4 Error Analysis

Our best algorithm is 90% accurate, but it is not uniformly so across different types of data. When the algorithm tries to classify historical facts, for instance, it often errs. That is because those facts are not about "History". Here is an example: the sentence "His 1543 book *Concerning the Revolutions of the Celestial Spheres* started an astronomical revolution" has history traits, but it is actually about "Science". To prove this idea, we tested each category separately, and a pure "History" test set, indeed, had the poorest accuracy.

Another problem arises when classifying sentences with two or more clauses, for the latter often are about different topics. Consider the sentence "This group's Kerry Livgren wrote "Dust in the Wind" after reading a book of Native American poetry." Its category is "Music", but it combines two clauses with different categories: "This group's Kerry Livgren wrote "Dust in the Wind" is about "Music", but "after reading a book of Native American poetry" is about "Literature".

A similar issue is present in single-clause sentences: phrases within a sentence often deal with conflicting categories. For instance, "This drug marketed as Motrin was patented in Great Britain in 1964" contains three contradicting phrases: "This drug marketed as Motrin" (Science), "in Great Britain" (Geography), and "in 1964" (History). We provide a possible solution for these last two problems in the *Future Work* section.

Furthermore, there is often dissonance between a question and its answer. Consider the following problematic pair: "December 8, 1980 in New York City" and "John Lennon". The first sentence (the question) contains historical and geographical data, but its answer describes a musician, and the category is, indeed, "Music". This suggests that the answer is usually more relevant than the question, and, to address this, we tried to double the weight of the answer in the algorithm. This improved the results, but it wasn't enough. Alas, giving even more weight to the answer yielded worse results.

Lastly, the training set is limited (only 10000 sentences) and, while perusing it, we found some errors. For example, the sentence "General Mills makes this snack with an unusual funnel shape in 6 flavors" and its answer "Bugles" are considered "Music", which is clearly wrong. We believe that with a better training set, we would achieve a better accuracy score.

5 Future Work

Here we present some suggestions for future optimizations. First, it has been shown that "if complete model selection using the gaussian kernel has been conducted, there is no need to consider linear SVM" [2]. With this in mind, we expect we would yield better results if we had the time to carefully tweak a gaussian kernel for the support vector machine algorithm. Second, it it possible that a careful lemmatization implementation could improve the results of the SVM model. For instance, we could lemmatize words based on their Part-of-Speech (POS) tags, thereby having different lemmas for different tags. Third and last, to address some the problems described in *Error Analysis* – different categories for different clauses or phrases – we think it worthwhile to try to assign different categories for each clause and phrase, and, then, choose the most relevant one for the whole sentence.

We think these ideas – properly implemented and thoroughly tested – have the potential to yield even better results.

References

[1] D. Jurafsky and J. H. Martin, "Speech and language processing. vol. 3," *US: Prentice Hall*, 2014.

[2] S. S. Keerthi and C.-J. Lin, "Asymptotic behaviors of support vector machines with gaussian kernel," *Neural Computing and Applications*, 2003.