Mark Gameng

CS 579

Project 2 – Fake News

Pre-processing

For pre-processing, all I needed to process was the titles, which were “title1_en” and “title2_en” columns. I did the usual pre-processing procedure. I lowered, removed capitalization, from everything to make it all unified. I then, removed punctuations and special characters in the titles as they may interfere with how the words are grouped up later. I probably should have looked at how advanced the word tokenize and lemmatization is from nltk library, as removing punctuations may have decreased its efficacy. After removing capitalization, punctuations, and special characters, I then tokenized the titles using nltk.word_tokenize, and then lemmatized the result using nltk.stem.WordNetLemmatizer. Doing all this will allow me to reduce morphological variation of the sentences and words. It will remove inflectional endings and only have the base or dictionary form of the words. Makes it easier to process and have a greater effect while training.

Here’s an example of what it did:

“< i > adorable < / i > < i > to cure cervical spondylopathy < / i > < i > and protrude from the waist, to drink Chinese medicine. < / i >” ➔ “i adorable i i to cure cervical spondylopathy i i and protrude from the waist to drink chinese medicine I”

“"If you do not come to Shenzhen, sooner or later your son will also come." In less than 10 years, Shenzhen per capita GDP will exceed Hong Kong.” ➔ “if you do not come to shenzhen sooner or later your son will also come in le than 10 year shenzhen per caput gdp will exceed hong kong”

Looking at some processed titles, it doesn’t fully clean up the titles but its good enough. You can see that <i> just returns “i” which is troublesome but finding and removing html entirely might take some time. Also, on the second example, you can see lemmatizing doing work. “less” ➔ “le”, “capita” ➔ “caput”

Training

First, its worth noting that the training dataset contains 256442 title combinations. Of those, 175598 are unrelated, 74238 agreed, and 6606 disagreed. So, there is a vast difference in label size. Only **28% have the label agreed and 2.5% are disagreed**. A good dataset will have the labels be more equal. Because we lack disagreed labels, it will be harder for a model to have a good accuracy for those. So, at the end you will probably see a good accuracy on unrelated titles while having a significantly lower accuracy on disagreed titles.

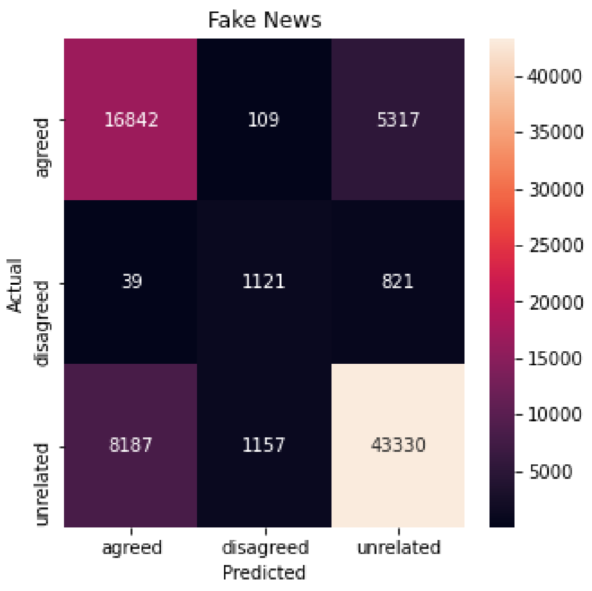
Now, to start training and evaluating models, I split up the training dataset into 70% training and 30% for evaluation. This gave me 179,485 title combinations for training which has 69% unrelated, 29%

agreed, and 2% disagreed. This is to be expected because it should follow the distribution of labels from the whole dataset since I used train_test_split from sklearn with stratifying on the label.

At first, I tried various models such as multinomial naïve bayes, k neighbors classifier, linear support vector classification, random forest classifier and logistic regression. I decided to stop training on k neighbors and random forest classifier as it takes too long compared to the others. So, I was left with multinomial naïve bayes, linear svc and logistic regression. Most of the parameters for the models were default but one of the important parameters that I adjusted was the number of iterations and the class weight to be balanced. I used the sklearn library for these.

The two base features that I wanted are bag of words and tf-idf, or term frequency-inverse document frequency. I knew that for most of the time tf-idf was better but I decided to just test and see the difference between the two. For the bag of words, I used CountVectorizer from sklearn and used it to build a vocabulary so I can use it for both the train and test dataset. Using the bag of words for each title in the pair, I combined them as one feature vector. I also thought about adding more features such as the similarity of sentences, using cosine similarity, as well as using n grams. For n-grams, my plan was to choose tf-idf n grams from 2 to 4 words, and then find the word sequences that was most correlated with each label using chi2. I had done this before on a separate project, and it helped a bit in increasing the accuracy, so I was hopeful it would do the same here. However, when I was testing and processing, it required too much memory. Maybe there was a way to lessen and solve this problem, but I couldn't figure it out so, I just left those features out as I didn't have enough memory for the data. Some of the feature engineering I tried out are on the bottom of the jupyter notebook include in the homework submission.

So, using bag of words as a base feature, I got 73% for multinomial naïve bayes, 79% for linear svc, 75% for logistic regression for the testing accuracy. For tf-idf as base feature, I got 76%, 79.6%, and 74%. Comparing tfidf with bag of words, tf-idf is a little better when comparing testing accuracy and F1 score. Specifically linear svc was the best.



As shown above, you can see that the model had a hard time classifying title pairs that disagreed, mostly due to the class imbalance. As I predicted, it performed the best for unrelated labels, then agreed, then the last is disagreed. The more data it has on a label, the better the model performs for that label.

Since the disagreed label only had 2.5% of data compared to others, it is expected to have a significantly lower accuracy than the other labels. Nevertheless, this model did relatively well in predicting the relation between the two pairs, resulting in a testing accuracy of 79.6%

Results

Looking at the testing results from various models and features. I decided to use linear svc with tf-idf as the base feature which got me 79.6% testing accuracy. Since the training and evaluation was done on 70/30% on the whole training dataset, I wanted to use the whole training dataset for training this time for the submission file. So, when I used all 100% of the training data for training, I got a training accuracy of 87% and F1 score of 87.4%. I then used that model to predict the given test dataset and input the predictions to a submission file, submission.csv.

References

Bird, Steven, Edward Loper and Ewan Klein (2009), *Natural Language Processing with Python*. O'Reilly Media Inc.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) **SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python**. *Nature Methods*, 17(3), 261-272.

API design for machine learning software: experiences from the scikit-learn project, Buitinck *et al.*, 2013.