

## CS4248 Assignment 2 Writeup

### A0184679H

#### **Model:**

The assignment of fact-checking classifies a string of text into three categories: an important fact with label 1, a non-important fact with label 0, and a non-fact with label -1. Since this is a linear classification problem, I have explored two machine learning algorithms, which is the Naive Bayes model and the Logistic Regression model. I have chosen the Logistic Regression as my model for several reasons:

- Discriminative vs Generative: Naive Bayes (NB) is a generative classifier, whereas Logistic Regression (LR) is a discriminative classifier. NB learns the joint probability  $P(x, y)$  to select the most likely label  $y$  by calculating  $P(y | x)$  with Bayes rules. In other words, it assumes independence of each feature, whereas LR calculates the posterior  $P(y | x)$  directly through a linear combination of the weights and the features. Furthermore, LR can be tuned with penalty later to prevent overfitting with the training data.
- Input size of training data: A NB model outperforms LR in the case where the training data is small, but with this assignment the training data has 1032 entries, which is sufficiently large enough for a LR model's performance to outperform NB's.
- Scoring result: After running both models locally and after submitting to Kaggle, the LR yields a higher score than a NB.

To tune the hyperparameters of the logistic regression model, I used Python's scikit-learn library, GridSearchCV to find an optimum input over a combination of various parameters. The parameters to choose from include a penalty score of (L1, L2), the LR solver of ('lbfgs' as a default, 'liblinear' to accept L1 and L2 penalty, and 'sag' for Stochastic Gradient Descent learned in lecture). As a result, the `LogisticRegression(penalty = 'l2', solver = 'sag')` has the best accuracy out of all combinations.

#### **Pre-processing:**

I attempted several different preprocessing to the input text, as described below. The starting score without any preprocessing (only feature engineering included), is 0.6681.

- Lowercase: After converting all the text to lowercase. After applying, the model's accuracy decreased by 0.005.



- Stopwords: After the stopwords are increased from the training data, the model's accuracy decreased by 0.01.
- Punctuation: After punctuations were removed from the training data, the model's accuracy decreased by 0.006.
- Stemming: After each word is stemmed, the model's accuracy decreased by 0.03.
- Any combinations of the above will result in the decrease of the model's accuracy. Therefore, I decided not to include any preprocessing of the training data.

### **Feature Engineering:**

I explored 3 different feature engineering, explained below.

- Bag of words: This is an essential step in feature engineering as the inputs of latter features and the model requires the output of this feature. With the CountVectorizer library, I converted the data from string format into a document-term matrix, and fed the matrix as an input into subsequent features. With this feature alone, the model has an accuracy of 0.7983.
- Bag of n-grams: Another additional feature I explored is using ngrams from the training set. After several rounds of parameters tuning, the highest accuracy of the model is when unigrams and bigrams are used, with an astounding accuracy of 0.9848 when predicting the training set.
- Tf-Idf: After I used TfidfTransformer in the model to include the feature term frequency and inverse document frequency, surprisingly, the model's accuracy plummeted from 0.9848 to 0.6681.
- Penalty: As I am concerned that the model will overfit the training data and fail to generalize on the testing data, I attempted to apply penalty to penalize large weights by trying L1 (Lasso Regression) and L2 (Ridge Regression). The model's accuracy was 0.6086 after applying L1 penalty and 0.6681 after applying L2 penalty. The higher score, L2 penalty, was chosen.
- Regularization: The LR model has an additional parameter, C which is used to dampen the penalty effect of the training data. A larger value of C corresponds to a weaker regularization, and after tuning of different C values, the model's accuracy when the C value is 1000 has the highest accuracy out of all features and preprocessings, with a value of 0.9995 even with tf-idf included that initially lowered the accuracy.

- Therefore, a combination of Bag of Words, unigrams and bigrams, tf-idf, L2 penalty, regularization value of 1000 is used for the model to be able to predict a close to perfect result on the training set. After I uploaded the predictions to Kaggle, it yielded my highest score in this assignment, 0.82867.

```
~/desktop/cs4248/cs4248-assignment2 ➤ python3 assignment2.py  
score on validation = 0.9995503371431105  
~/desktop/cs4248/cs4248-assignment2 ➤   40s 19:17:59
```

### Analysis:

For a model to be able to train and predict data, the empirical form would be a CountVectorizer to convert the training data into a numpy array, and a machine learning algorithm (in this context, I chose LR) to train itself and generate predictions. I created a model with only a CountVectorizer and Logistic Regression without any hyperparameters, and this model has an accuracy of 0.8298. Earlier in the section, I presented my finding that preprocessing, in any form regardless of punctuation removal, stopwords removal, stemming, lowers the accuracy of the model. After inspection of the training data set, one possible explanation for this is that when the text is preprocessed, it might lose its original intended meaning, such as “U.S.” will be processed to “us”, which is an incorrect conversion and might affect the model’s accuracy. Similarly for stemming, the word “financing” and “finance” both appear in the training data, but after stemming, they will all be reduced to “financ”, which is not the intended way as these are derivational morphology instead of purely inflectional.

As for feature engineering, one interesting insight is the inclusion of the tf-idf feature. Before I included the tf-idf feature into the model, the model’s accuracy was 0.8298, but after I included it, the accuracy decreased to 0.7152. After analysing the training and testing data set, one possible reason that caused this surprising finding is that several words can have a huge count in the training set, but very sparse in the testing set. For example, the word “president” has an occurrence of 1099 in the training set, but only 47 occurrences in the testing set. As “president” is found in almost all documents (in this context, each row of text), its inverse document frequency will be very low, and the feature “president” will have a small weightage, but “president” is the best classifier for this classification task. Therefore, when

both data sets have imbalance word counts, tf-idf might affect the model's accuracy. However, to avoid overfitting the testing set, I included tf-idf as one of my features such that the model is more robust and is able to generalize on the hidden test set on Kaggle.

Another interesting finding is the usage of ngrams. I used various ngrams combinations when training the model, including only unigrams (1, 1), unigrams and bigrams (1, 2), and only bigrams (2, 2). I did not tune my model with trigrams and 4-grams, as I submitted an entry using trigrams and as expected, the model is "memorizing" the training set and has only a competitive score of 0.3829 on Kaggle. The best option of above is a mixture of unigrams and bigrams (1, 2), which increased the score of the empirical model from 0.8298 to an impressive 0.9945. My analysis is that in contrast to the Naive Bayes algorithm that assumes independence, ngrams provides the model more features to choose from, thus increasing the model's accuracy.

Next, the usage of penalty. I included the L2 penalty into the model, and the model has an accuracy of 0.6614, whereas the accuracy for L1 penalty is 0.6308. This decrease is expected as the model assigns penalty for heavy coefficients to prevent overfitting, and this is further supported when I submitted my predictions to Kaggle and yielded a competitive score of 0.7582. The reason I chose L2 over L1 is not only due to the higher score, but also the sparsity of the training set. I did some calculations with the dataframe and found out there are 349 unknown words encountered, and the L1 penalty simply zero out these unseen features. Hence, the L2 penalty is chosen.

Finally, the hyperparameter C, which is the inverse of regularization strength. I did further research and read that C is the inverse of coefficient,  $\lambda$  multiplied by the L2 penalty. A higher C value indicates a weaker regularization, and I tuned the model over different C values, including 0.001, 0.01, 0.1, 1, 100, 1000. The C value of 1000 has the highest score of the empirical model, increasing the original from 0.8298 to 0.9583 over 5000 iterations of stochastic gradient descent. This is expected as C increases,  $\lambda$  decreases, and less penalty is deducted.

## Library Dependencies

- [nltk](#) (version 3.5, for using PorterStemmer to stem words)
- [re](#) (version 3.9.2, for text preprocessing and remove punctuations)
- [nltk.corpus](#) (version 3.5, for filtering stopwords)
- [CountVectorizer](#) (version 0.24.1, for using bag of words feature)
- [TfidfTransformer](#) (version 0.24.1, for using tf-idf feature)
- [LogisticRegression](#) (version 0.24.1, for machine learning algorithm)
- [Pipeline](#) (version 0.24.1, for pipelining the output of CountVectorizer, TfidfTransformer, and LogisticRegression)

## Instruction to Run Code

After unzipping the file, simply navigate into the directory that contains the (.py) file, along with a training dataset named train.csv, and a testing dataset named test.csv. Run the command `python3 assignment2.py` and after approximately 40 seconds, the validation score should be outputted.

## 1A. Declaration of Original Work.

*By entering my Student ID below, I certify that I completed my assignment independently of all others (except where sanctioned during in-class sessions), obeying the class policy outlined in the introductory lecture. In particular, I am allowed to discuss the problems and solutions in this assignment, but have waited at least 30 minutes by doing other activities unrelated to class before attempting to complete or modify my answers as per the Pokémon Go rule.*

*Signed, A0184679H*

## Reference

*I give credit where credit is due. I acknowledge that I used the following websites or contacts to complete this assignment.*

- [sklearn.pipeline.Pipeline — scikit-learn 0.24.1 documentation \(scikit-learn.org\)](#), for reading Pipeline's documentation
- [sklearn.linear\\_model.LogisticRegression — scikit-learn 0.24.1 documentation \(scikit-learn.org\)](#), for reading LogisticRegression's documentation
- [sklearn.feature\\_extraction.text.CountVectorizer — scikit-learn 0.24.1 documentation \(scikit-learn.org\)](#), for reading CountVectorizer's documentation

- [sklearn.feature\\_extraction.text.TfidfTransformer — scikit-learn 0.24.1 documentation \(scikit-learn.org\)](#), for reading TfidfTransformer's documentation
- [2. Tuning parameters for logistic regression | Kaggle](#), for learning how to tune parameters for LR
- [Comparative Study on Classic Machine learning Algorithms | by Danny Varghese | Towards Data Science](#), for reading up to choose whether NB or LR
- [On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes \(neurips.cc\)](#), for understanding on performance between NB and LR
- [sentiment analysis - In general, when does TF-IDF reduce accuracy? - Stack Overflow](#), for understanding why does Tfidf affects my LR model
- [Intuitions on L1 and L2 Regularisation | by Raimi Karim | Towards Data Science](#), for learning key difference between L1 and L2 regularization
- [python - What is the inverse of regularization strength in Logistic Regression? How should it affect my code? - Stack Overflow](#), for understanding C parameter in LR
- [Machine Learning, NLP: Text Classification using scikit-learn, python and NLTK. | by Javed Shaikh | Towards Data Science](#), for understanding basic code setup of pipelining