Project 1: Language Modeling and Opinion Spam Classification
CS5740: Intro to NLP

Kaggle group name : Team Rocket

Saurabh Kumar Yadav **(sky36)** , Sudhanshu Khoriya **(sdk225)** , Soham Ray **(sr2259)**

Abstract

The burgeoning opinions on social media due to the availability of rich resources like online product reviews , hotel review sites have made it easier for the general public to choose and form an opinion of their own. These reviews and online opinions are informations that revolve around user's sentiments and feelings based on their experience. But as user looking up online for a specific review, differentiating between honest reviews with fictitious ones is difficult. In this project, we focus on designing our own language model that would predict and label a review to be fictitious or non-fictitious. We will then also benchmark the accuracy of our language model with Naive Bayes based classifier to analyse areas where the results differed.

1 Overall Goal and Workflow

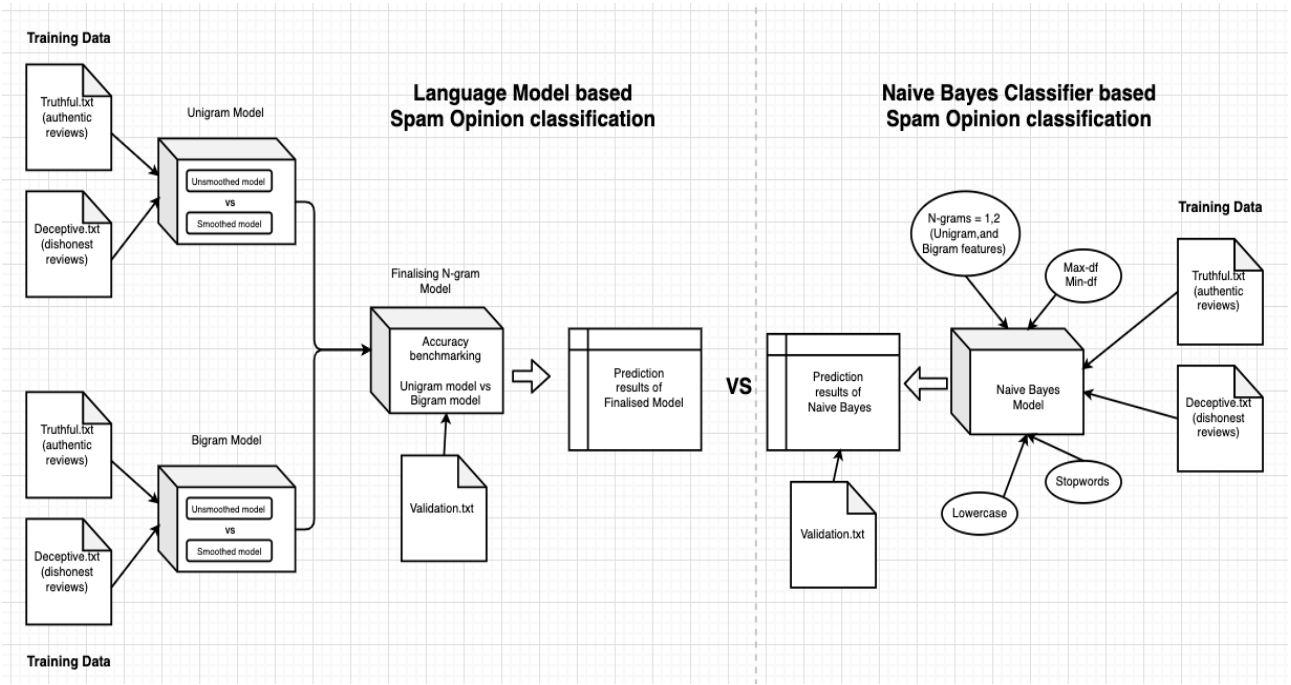

Figure 1: Workflow

In this project, we have built an n-gram based language model (Unigram and Bigram) and also investigated a feature based Naive Bayes model for the Opinion Spam Classification.
The project is divided into two parts. First we preprocessed the training data (truthful.txt and deceptive.txt) and built unsmoothed models of Unigram and Bigram, then followed by applying smoothing techniques. We then selected the best performing smoothing technique and compared the two language models on validation data.
After finalising the language model, we built a feature-based Naive Bayes model using scikit-learn library and trained it using the same corpus (truthful.txt and deceptive.txt).

2 Language Model

2.1 Data Preprocessing

Tokenization: The training set was organized as truthful.txt and deceptive.txt where each line was a single review. We created a BOW (bag of words) by reading each line and using whitespace as a separator to create word tokens and generating a bag of words. We converted all the words in lowercase as it reduces redundancy in our unigram bag of words. The punctuations were not removed as they are important in short reviews and we would lose some features which will help in identifying the reviews. The total vocabulary of truthful unigram was 6381 and total vocabulary in deceptive corpus was 5367.

2.2 Unigram

The unigrams were created by taking individual pre-processed words in each review in training data and storing their count in a data structure. This was done for both truthful and deceptive corpus.

2.3 Bigram

The bigrams were created by taking a pair of pre-processed words in each review in training data and storing their count in a data structure. This was done for both truthful and deceptive corpus.

3 Smoothing

We trained our model by incorporating Laplace add 1 smoothing and found out that the accuracy of predictions on our validation model was low as it unevenly distributed the probability mass in favour of the unknowns . So in order to minimize this effect we introduced add k smoothing.

Deciding K- value In order to find out the optimum value of k, we iterated over different values of k from 0.1 to 1.0 and compared the validation results for each of these values. The accuracy of prediction was maximum for k=0.1

4 UNK Handling

4.0.1 UNK Handling in Unigram

For handling unknown words in unigram we used the token UNK and introduced it into our vocabulary. Whenever a new word is encountered which is not in unigrams we considered it as unknown and calculated its probability accordingly. We chose this method to avoid assigning unnecessary higher count for unknown words and assigning lower counts to words in our corpus.

4.0.2 UNK Handling in Bigram

For handling unknown bigrams we used a similar approach as unigram and we got three cases such as : (unk,word), (word,unk) or (unk,unk) where either one word is unknown or both are unknowns. For all the cases we considered the count of the bigram as 1 as the count of corresponding bigrams in our modified corpus is 1. This ensures we are maintaining the original features i.e bigrams in our training corpus to avoid overfitting.

4.1 Perplexity Calculation

4.1.1 Unigram Model

Truthful vs Deceptive: The perplexity was calculated by the formula,

log(p) = -log(p1) - log(p2) - log(p3) - log(p4) . -log(pn),
where, p is perplexity
and p1,p2,p3..pn are the probability of words in the review.

The probability of smoothed words were calculated as :

$$\textbf{P(w)} = \frac{count(w)+k}{N+kV}$$

The perplexity of truthful model and deceptive smoothed models were compared and the lower perplexity label was assigned to the review.

4.1.2 Bigram model

Truthful vs Deceptive: Similar to our unigram perplexity calculation we calculated the probability of each bigram in each review and compared the perplexity of truthful and deceptive smoothed models . The label was assigned for the model that gave the lowest perplexity

$$\textbf{Probability } (P(w_{n-1}w_n) = \frac{[count(w_{n-1}w_n)+k]*count(w_{n-1})}{count(w_{n-1})+kV}$$

5 Language Model Classification

5.1 Test Results

The accuracy of unsmoothed models for unigram and bigram respectively were 30% and 25% as the probability of 0 was assigned for any unknown unigrams and bigrams in the validation set. The accuracy of unsmoothed models for unigram and bigram were on average 92% and 93% on validation set.

5.2 Benchmarking of Bigram vs Unigram

From the above test results we concluded that smoothed models performed much better on validation sets as compared to unsmoothed models.
The smoothed unigram and bigram models were almost identical in their accuracy so we tested both of them on the test set on Kaggle and the bigram model gave an accuracy of 91.4% and the unigram model gave an accuracy of 90.6% .

5.3 Classification

We classified the labels of the test set reviews by comparing the perplexity of individual reviews for truthful and decptive corpus and the one which gave lower perplexity was assigned to the review.

6 Naive Bayes Classification

6.1 Naive Bayes Model

The Naive Bayes model for text classification is based on Bayes Rule of probability. We represent a document(X) as a set of words and its frequency pairs. For each label y, a probabilistic model P(X— Y = y) is built. Finally, we select the label y which has the highest probability of generating X.

$$\hat{y} = \underset{x}{\operatorname{argmax}} P(X|Y = y) * P(y)$$

6.2 Bag of Words and N-grams

For Naive Bayes, we use the pythons scikit-learn library for classifier implementation and feature extraction. To do this, we first do some basic preprocessing:

Vectorizer A vectorizer converts a collection of text documents to a matrix of token counts. Its based on the bag of words model. CountVectorizer internally tokenizes the sentence, counts the words, and normalizes the text by giving words diminishing weightage based on their frequency. There are 2 types of vectorizer we explore, CountVectorizer and Tf-Idf Vectorizer.

CountVectorizer We first used the CountVectorizer. This has 2 functions:

- **fit():** Fit is used to create a vocabulary based on the training corpus
- **transform():** Transform creates a word vector for every review in a test corpus. The vector has dimensions equal to the vocabulary, with every word position storing the count of words

The CountVectorizer simply takes a count of the words present and forms a vector. Tf-Idf stands for Term-Frequency-Inverse-Document-Frequency. Here, Term Frequency stands for the number of times a term appears in a document, and weight is allocated accordingly. This is similar to the CountVectorizer. However, in Inverse document frequency, we allocate the weight as per how often a word appears across all documents. This signifies how important the term is to the document, disregarding common stop words. In our final model, we use the TfidfVectorizer. Its equation is given by:

$$w_{t,d} = (1 + log(1 + tf_{t,d})).log_{10}N/df_t)$$
where term frequency is given by $1 + log(1 + tf_{t,d})$
and inverse document frequency is $log_{10}N/df_t$

N-Grams In both vectorizers, we have the option of extending features on the basis of n-grams. The default value is unigram, however, we explore bigrams, trigrams and further. Due to a limited training corpus, the model performs best with bigrams. With trigrams, the model begins to overfit and accuracy drops.

6.3 Linguistic Features

We considered some linguistic features in our model design. As a part of data preprocessing, we implemented:

- **Stemming:** To prevent confusion among different forms of the same the word, such as go, going and gone, we use stemming as essentially, all forms mean the same thing. To do this, we stem our text corpus before submitting it for training.
- **Lemmatization:** Stemming uses a crude way of reducing words to their original form, it just crops them. Lemmatization removes inflection and is a better way of finding the root word.

  We implement both, and obtain better results with lemmatization.

7 Classification Feature Validation

In language model our classification features were the unigrams, bigrams and their corresponding BOW counts.
We validated these features by testing it on validation data and got the accuracy of 92% and 93% respectively.

In Naive Bayes classification model While creating the vectorizer, we explored multiple features. These include N-grams, Maxdf, Mindf, Stop-words, max_features and Lowercase functionality

8 Implementation Details

8.1 For language model

We implemented a bag of unigrams and bigrams and stored their count in a python dictionary as it makes it easy to retrieve the counts of corresponding unigrams and bigrams. We created one bag of words each for the truthful and deceptive corpus for both the models (give n_grams). We added 'UNK' token to the unigram dictionary and set it's count to 1 to handle the unknowns.

For unsmoothed model we calculated the probability of the unigrams and bigrams in the test data by using the functions **unigram_unsmoothed()** and **bigram_unsmoothed()**. The probabilities of truthful and deceptive models were compared for each review and corresponding labels were assigned.

For smoothed model the probabilities of unigram and bigrams were calculated by the functions **unigram_prob()** and **bigram_prob()**. The truthful and deceptive perplexity was calculated for each review by the function **perplex()** in log form. The perplexities were compared and the review was labelled with the lower perplexity among the two. For the final Kaggle results we used the smoothed bigram model. The calculation of probability of each review and assigning labels and creating the csv file takes place in the function **initialize()**

8.2 For Naive Bayes Model

- **N-grams**: unigram and bigram features
- **Max_df**: 0.9
- **Min_df**: 0.1
- **Stopwords**: Stopwords ignored due to tfidf
- **Max_features**: 60000
- **Lowercase**: True

```
vectorizer = TfidfVectorizer(
                sublinear_tf=True,
                lowercase=True,
                norm='l2',
                ngram_range=(1,2),
                max_features=60000,
                min_df=0.01,
                max_df=0.9)

clf = MultinomialNB().fit(X_train_tfidf, train[1])
```

9 Feature design, discussion

9.1 In Language Models

The features in our language model were the BOW of unigrams and bigrams. For designing the features we did not exclude any punctuations and stop words as it gives us more features for classification of reviews. People tend to use a lot of punctuations and stop words in reviews to articulate their opinion and we did not want to lose those features given the limited amount of training data. We included only one extra unknown to preserve the original features of our model.

9.2 In Naive Bayes Model

We considered following features,
N-grams: Tried several models with unigrams, bigrams, trigrams. We selected bigrams, since trigrams overfit the model due to limited training data.

Max-df: This stands for maximum document frequency. Words present in more than x% of documents are ignored. We take a high max_df score to ignore common words that occur in all documents. Due to a lack of training data, we are unable to ignore more words.

Min-df: This stands for minimum document frequency. Words present in less than x% of documents are ignored. In this parameter, drastic changes are observed. Setting min_df to 0.1 drops features from 60,000 to 83. This tells that that there are several rare words in the corpus.

Stop-words: Stop-words can be ignored or retained. Removing stop-words allows us to place more weightage on more indicative words. However, using Tf-idf automatically takes care of stopwords as it gives words commonly occuring across all documents lesser weightage.

Max_features: Maximum number of top features to consider. We consider a high number of features to maximize accuracy and prevent under-fitting

Lowercase: Convert text to lowercase before prepossessing to standardize text data

10 Error Analysis

10.1 Quantative

10.1.1 LM Confusion Matrix

	Predicted Truthful	Predicted Deceptive
Actual Truthful	109	19
Actual Deceptive	5	123

10.1.2 Naive Bayes Confusion Matrix

	Predicted Truthful	Predicted Deceptive
Actual Truthful	118	10
Actual Deceptive	12	116

10.2 Qualitative

For error analysis, we have 3 cases that we manually reviewed.
First, when Language Models is classifies the review correctly, however Naive Bayes fails.
An example of this is: ”My husband and I decided to take a trip to Chicago at the last minute and quickly chose the Conrad , not really knowing it ’s location . We were pleasantly surprised to find it was almost right on Michigan Ave. ... ”

Second, when Naive Bayes classifies the review correctly, however the language model fails.
An example of this is: ”Just back from spending Memorial Day weekend in Chicago . We decided to stay at the Talbott , and we reserved a room there over a month ago ... ”

Third, when both the language model and the Naive Bayes model fail to classify a review correctly.
An example of this is: ”If I did n’t have to stay here , I would n’t . Without exception , I swear I am not lying , every day of my stay brings a new problem . Over the past 3 months , I have stayed here 6 times for 3 nights at a time ; problems every time ... ”
No clear pattern can be observed qualitatively.

11 Programming libraries

- Pandas
- Collections in python
- math
- sckit-learn library
- nltk

12 Individual Contributions

To ensure everyone gained expertise in all areas, we distributed tasks accordingly.

12.1 Saurabh Kumar Yadav:

Saurabh worked on smoothing of bigram and unigrams as well as Naive Bayes Linguistic features

12.2 Soham Ray:
Soham worked on creating probability dictionary for the language model as well as feature extraction for the Naive Bayes model.

12.3 Sudhanshu Khoriya:
Sudhanshu worked on the unknown word handling, and model tuning of the language models as well as parameter tuning and data preprocessing for Naive Bayes.

13 Feedback

It's an interesting project that made us explore, how subtle changes in tones and words can help us differentiate truthful reviews from fictitious ones. And how we can design our language models that can help classify these spams reviews that would otherwise be difficult for a human to get it straight away.
Toying with various smoothing techniques also helped us better understand how to handle UNK words when encountered in a text corpus.It also gave us insights on Naive bayes and its comparison with LM.

14 Test Results
We observed a range of different test results with the Naive Bayes model dependent on the preprocessing applied. In general, accuracy ranged from 85% to 92% when tested on the validation dataset. We found that accuracy improved with the use of Tf-idf Vectorizer, bi-grams, low min_df values and high max_df values: this is inline with our intuition.
An average accuracy of 90% is observed with best accuracy at 92%. To prevent overfitting the validation dataset, which is limited in size, we go with the model proposed above, that is coherent with our intuition.

15 Conclusions
In this project, we explored multiple models for Opinion Spam Classification. In addition to trying out different models, we also experimented with different pre-processing techniques.
Largely, we explored two classification models: Language Models and Naive Bayes Classifier.
For Language Models, we got an average accuracy of 92% on the validation data set: we tried with both unigram and bigram approaches, with and without smoothing and unknown word unhandling.
However, the Naive Bayes model outperforms the Language Models, both in the validation set as well as the test set.
We found some common patterns in both models. Bi-grams are better than unigrams at generalizing new data. For the Naive Bayes model, using linguistic features and removing stopwords provided better results.

16 Kaggle Rankings

We ran our model against the test set on Kaggle, as a part of the Opinion Spam Classifier competition, and uploaded our predicted results.

We achieved the third highest score for Language Model Classification and second highest score for Naive Bayes Classification as on 09/27/2019.

Team Name : Team Rocket

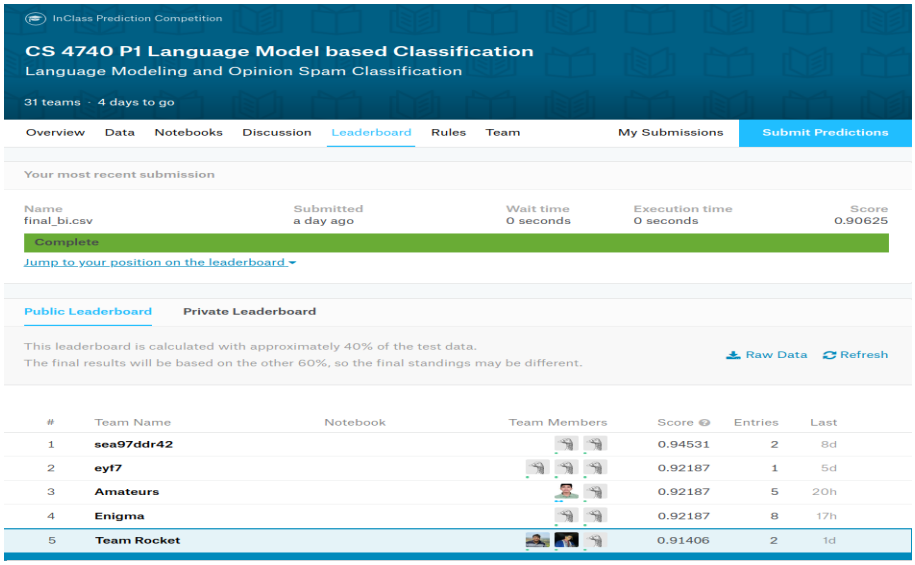

Figure 2: Language Model leaderboard ranking

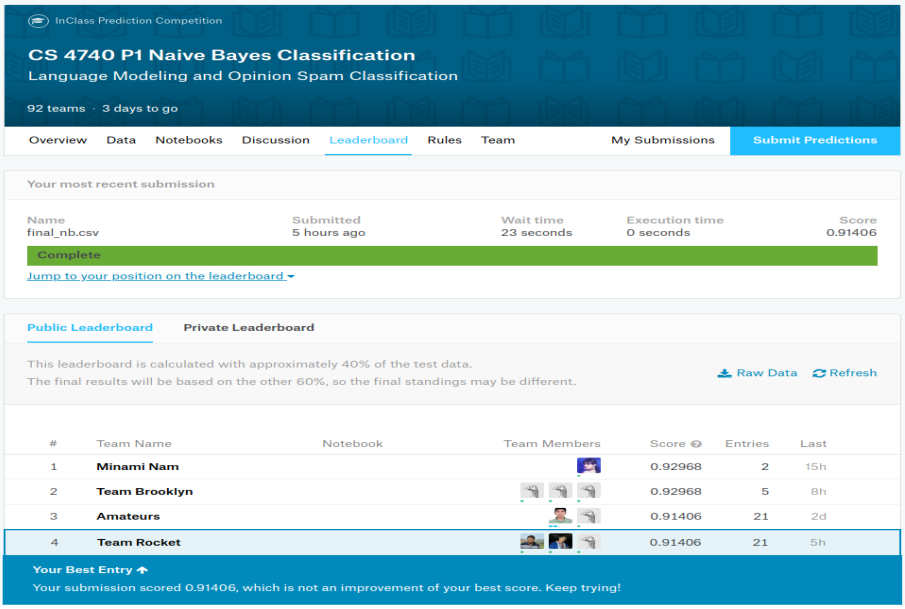

Figure 3: Naive Bayes leaderboard ranking

References

Andrew McCallum, Kamal Nigam. ”A Comparison of Event Models for Naive Bayes Text Classification.” (1998).