Classification of Restaurant Reviews to Predict Hygiene Conditions

1. Learning Algorithms

For this task I analyzed the results of 7 sklearn algorithms for experimental purposes: Naive Bays (multinomialNB), SGD classifier, Logistic Regression, SVM (in the form of Linear SVC and NuSVC), Random Forest, AdaBoost. In addition, I tried the xgb. XGBClassifier whose results are well commented in the literature, and the StackingClassifier from the mlextend Python module. I also tried to use the RandomUnderSampler and RandomOverSampler from the imblearn Python module because the test data set is very imbalanced.

I used CountVectorizer and TfidfTransformer from sklearn.feature_extraction with each classifier because this seemed to be appropriate for the task where you don't want to use words that are too frequent and non-characteristic. I tried to vary the minimum document frequency min_df and maximum document frequency max_df in order to exclude more frequent and non-representative words on the one hand, and very rare words on the other.

2. Text Representation

Out of curiosity and for experimental purposes, I used 9 different text representations to see which one performs better: unigrams only, bigrams only, unigrams + bigrams, bigrams + trigrams, ngrams with N = 1 through 3, 1 through 4, 1 through 5, 1 through 6, 1 through 7. Naturally, I quickly discovered that a very large number of N does not yield any benefits and excluded ngrams with N = 1 through 7 and 1 through 6 from my analysis.

In addition, also for experimental purposes, I used classifiers on raw text (only stopword removal), with stemming (nltk Porter Stemmer), and with lemmatization (nltk). Moreover, I tried using the CountVectorizer's built-in 'english' stopword list and my own stopword lists ranging from 10,000 to 1500 most frequent English words [1] which I supplemented with the lemur stopwords from the Text Information Systems course, Python stopwords, and some other popular lists.

3. Additional Features

I did use the additional features for my experiments because I believe they provide valuable additional extra-linguistic information, and the prediction results were worse when I tried classifiers without them. Such useful information may include, for example, the average restaurant rating and zipcodes.

4. Analysis

To test classifiers locally, I performed an 80/20 split of the training dataset containing 546 data instances, trained each classifier on 80% and left out the remaining 20%, after which I calculated the F1 score of the performance of each classifier on the left out 20% of the dataset (unseen by the classifier). Table 1 is a typical table that I analyzed to select the best classifier and the best set of parameters. I made several similar tables by varying min_df, max_df, ngrams, stopwords, and introducing stemming and lemmatization.

	Unigrams only	Bigrams only	1+2	2+3	1 thru 3	1 thru 4	1 thru 5
MultinomialNB	0.655	0.606	0.649	0.606	649	0.649	0.649
SGD	0.618	0.623	0.642	0.642	0.642	0.642	0.642
Logistic Regression	0.617	0.641	0.61	0.627	0.61	0.61	0.61
LinearSVC	0.595	0.593	0.655	0.631	0.667	0.667	0.667
NuSVC	0.621	0.643	0.684	0.655	0.649	0.649	0.649
Random Forest	0.598	0.585	0.654	0.552	0.593	0.661	0.655
AdaBoost	0.598	0.585	0.561	0.611	0.482	0.482	0.482

Table 1. F1 score for min_df=15, max_df=0.25 (raw text)
* green - best score, gray - next to the best

The values of min_df and max_df from Table 1 turned out to be the most efficient for most of the classifiers, with some individual cases when a specific classifier performed better with different parameters. As you can see, Naive Bays tends to perform better with unigrams, and SGD with ngram where N >= 2. Usually, AdaBoost seemed to be the weakest classifier compared with others.

	Unigrams only	Bigrams only	1+2	2+3	1 thru 3	1 thru 4	1 thru 5
MultinomialNB	0.672	0.606	0.673	0.606	0.684	0.684	0.684
SGD	0.642	0.623	0.667	0.636	0.667	0.667	0.667
Logistic Regression	0.648	0.629	0.642	0.621	0.642	0.642	0.642
LinearSVC	0.619	0.655	0.655	0.655	0.667	0.667	0.667
NuSVC	0.65	0.65	0.606	0.661	0.559	0.559	0.559
Random Forest	0.624	0.613	0.623	0.606	0.631	0.63	0.649
AdaBoost	0.557	0.547	0.56	0.472	0.552	0.552	0.552

Table 2. F1 score for min_df=15, max_df=0.45 (raw text)

Table 2 shows that Naive Bays may reach the local score of almost 0.684 under certain conditions. In general, the most robust classifiers (having high F1 scores under many circumstances) turned out to be SGD, SVM, and Random Forest. The latter has a tendency of showing different scores for the same set of conditions when run repeatedly which has to do with its nature - each time the classifier builds a different tree. The best F1 leaderboard score on raw text with only stopword removal was achieved after all by MultinomialNB and equaled **0.828**. Surprisingly, my additional stopword lists with a different number of words showed good local results, but did not help with the leaderboard, and I ended up using the built-in ‘english’ stopword list of CountVectorizer as the most efficient one. Also, CountVectorizer performed better than TfidfVectorizer.

The introduction of stemming resulted in some surprises locally . Table 3 shows the best F1 score that I managed to achieve locally at that time.

	6500	1500	9500	3000 Google
MultinomialNB	0.596	0.661	0.614	0.685
SGD	0.631	0.648	0.649	0.667
Logistic Regression	0.595	0.636	0.642	0.623
LinearSVC	0.582	0.632	0.613	0.625
NuSVC	0.584	0.604	0.577	0.542
Random Forest	0.642	0.684	0.618	0.66
AdaBoost	0.719	0.678	0.673	0.6

Table 3. F1 score for min_df=15, max_df=0.25 (Porter stemmer) for different number of stopwords (derived from the list of most frequent English words in [1])

However, AdaBoost’s performance was not confirmed by the leaderboard which means the high local score is just a result of overfitting. The best F1 leaderboard score that I achieved with stemming was **0.8404**, and this was done with the Porter Stemmer (nltk), MultinomialNB, and ngram = (1,3).

When I tried to improve my classifier, and trained it on the entire train dataset of 546 data points, the leaderboard results were actually worse than when I just trained the classifier on only 80%. I am not sure what this has to do with. Also, classifier results with { nltk tokenizing, POS tagging, and lemmatization with POS} were worse than with stemming; in addition, lemmatization took a lot more time than stemming, so I excluded this method.

The use of XGBClassifier did not improved my score. Hoping to improve it further, I used RandomUnderSampler and RandomOverSampler from the imblearn module. The undersampler performed better, especially when combined with StackingClassifier from the mlextend module. The way I used the undersampler was this: I took the prediction results for all the remaining 12,000+ data points that gave me the best learderboard score and used them as labels for the test set. Using this configuration I managed to achieve only marginal improvement, my best F1 score is **0.8435** which gave me the first place on the leaderboard at the time of the report. This was achieved with ngrams = (1,3) and built-in ‘english’ stopwords.

To recap, here are the required minimum details about my best method:

- Toolkit: sklearn, imblearn, mlextend
- Text preprocessing: stopword removal (built-in ‘english’ stopword list from CountVectorizer), Porter Stemmer (nltk), punctuation removal (regex)
- Text features: CountVectorizer, TfidfTransformer, unigrams + bigrams + trigrams, min_df=1, max_df=0.35
- Learning algorithm: StackingClassifier from mlextend with MultinomialNB + LinearSVC or RandomForest + SGDClassifier and LogisticRegression as meta classifier (all from sklearn), boosted by RandomUnderSampler from imblearn

5. Conclusion

I experimented with a very wide range of many different parameters and this resulted in F1 scores ranging from lower sixties to mid eighties with the top one being 0.8435 which gave me the first place at the time of the report. It seems like in order to get the best results one needs to tune the parameters of a classifier to a specific text corpus which means there is probably no universal solution, at least with the classifiers that I tried.

A different approach is needed for significant improvements, and I would start with deep learning.

References

[1]. Most frequent 10,000 English words: https://github.com/first20hours/google-10000-english