Jonathan Chang
CS598 – Data Mining Capstone

Task 6 Report

1. Overview

In this task, our goal is to help customers in deciding which restaurants to dine at based upon its hygiene standards, namely, whether or not it was able to pass a hygiene inspection. Given a dataset with concatenated text reviews of a restaurant along with some additional features such as the cuisines offered, zip code, number of reviews, and average rating, corresponding labels to indicate whether or not that particular restaurant has passed the latest public health inspection test. As described in the problem statement, the first 546 entries correspond to our labeled training data, and the remaining 12753 entries are unlabeled, and we need to predict labels for them.

2. Description and Comparison of Methods

2.1 Data Preprocessing

For data preprocessing, I copied many of the preprocessing steps that I had learned from Task 1, and utilized gensim's preprocessing function with most of the default filters which would essentially convert the text to lowercase, strip any "bad" characters (e.g. tags, punctuation, whitespaces), remove stop words, and then stem the text and return a list of the tokens. I then used NLTK's WordNet Lemmatizer in order to convert all the tokenized words to their base form. Ultimately, I believe that these data preprocessing steps are a useful way to "normalize" the text to meaningful and important words to pass into language models and classifiers.

2.2 Text Representation/Feature Engineering/Modeling

```
%%time
def test_classifier(clf, X, y, vectorizer, text_col='text'):
    pipeline = Pipeline([
        ('union', ColumnTransformer(
            [('cuisines_offered', CountVectorizer(), 'cuisines_offered'),
             ('zipcode', OneHotEncoder(dtype='int', handle_unknown='ignore'), ['zipcode']),
             ('num_reviews', CountVectorizer(token_pattern='\d+'), 'num_reviews'),
             ('avg_rating', CountVectorizer(token_pattern='\d+'), 'avg_rating'),
             ('text', vectorizer, text_col)],
            remainder='passthrough',
        )),
        ('clf', clf)
    ], verbose=False)
    scores = cross_val_score(pipeline, X, y, cv=5, scoring= 'f1')
    print(clf)
    print(scores)
    cv_score = np.average(scores)
    return cv_score
```

My first order of business was to construct a process that would allow me easily test different text representations and machine learning models for the classification portion. Luckily, sklearn pipelines allowed me to streamline the process of transforming my input features and

evaluating a specified model. The ColumnTransformer was especially useful because it allowed me to specify my different feature columns of heterogenous types and apply a transformation to them. For example, I wanted to use OneHot Encoding for the 30 zip codes represented in the dataset. Or, I wanted to use a CountVectorizer() to represent the different cuisine types as a feature. Of course, the text column was important, and I primarily used a Bag of Words representation via the CountVectorizer() and the TfidfVectorizer(). Generally, TfidfVectorizer performed better than CountVectorizer because CountVectorizer merely counted the word frequencies, but Tfidf was able to weight the words and adjust for the importance of words using inverse document frequency. As mentioned below, I briefly attempted using word2vec embeddings without much boost in performance.

The sklearn pipeline object was very useful because it not only allowed me to combine the various column transformations, but it also allowed me to add a classifier as a secondary step. For ease of use and comparison, I tried the following model implementations in sklearn: Naïve Bayes, Support Vector Machine, Random Forest, and Gradient Boosting. Although model selection is important – for instance, SVM and Naïve Bayes are very popular for text classification algorithms, especially with limited training data – I believe that the feature engineering (specifically for text representation) was the most important part of this task.

2.3 Methods Comparison

For most problems, I would usually just use train_test_split in order to split the training data into roughly 80% for training and 20% for validation. However, because we know that because our dataset is imbalanced, it is better to have a cross-validation strategy. Thus, for evaluation purposes, I used sklearn's cross_val_score function in order to perform cross-validation of model performance across 5 distinct folds. I believe that this is a better approach because the models may perform very differently on different data splits. Below is a table showing the results

Model	**Preprocessing**		**No Preprocessing**	
	TFIDF	***BOW***	***TFIDF***	***BOW***
Naïve Bayes	0.63048	0.63761	0.62783	0.62817
SVM	0.63372	0.46879	0.63988	0.46910
Logistic Regression	0.62070	0.62312	0.61886	0.60590
Random Forest	0.62331	0.62991	0.60047	0.63589
Gradient Boosting	0.59142	0.61679	0.57390	0.59019

Because of the discrepancy in data distribution between our training dataset and the hidden leaderboard dataset, it is hard to accurately compare performance between my model

attempts. For instance, certain models that worked well locally didn't necessarily translate to higher F1 score on the leaderboard, and we likely have to tune specifically to a given text, as, according to the No Free Lunch Theorem every model makes specific assumptions about the dataset, and certain models will perform better than others in certain edge cases.
Generally however, SVM and Naïve Bayes as they have historically been used successfully in text classification and tended to work better on the leaderboard probably because they are less prone to overfitting.

2.5 Ensembling

I did attempt to create an ensemble of the various models I tried using a package called mlens. Essentially I used all the models (e.g. RandomForest, LogisticRegression, Naïve Bayes, etc…) as base learners, and then passed them to a second layer meta learner that blends together the results (basically learning from the previous class predictions of the base learners). In theory, this should have work well because of the diversity of uncorrelated base learners should help the overall final meta estimator be able to overcome the separate models' individual weaknesses, and as we can see from the table, many of the cross-validated scores seemed very uncorrelated. Given more time, I would have liked to play with ensembling even more, and especially tune the final meta layer since I only used a simple LogisticRegression Classifier and was not able to achieve a better result on the leaderboard at the time of this report.

3. Best Performing Method

My best performing model on the leaderboard was able to achieve a score of 0.7801, and it was ranked 10th at the time. However locally, it only achieved an average F1-Score of 0.63043 across 5 folds.

```
pipeline = Pipeline([
    ('preprocess', ColumnTransformer(
        [('cuisines_offered', CountVectorizer(), 'cuisines_offered'),
         ('zipcode', OneHotEncoder(dtype='int', handle_unknown='ignore'), ['zipcode']),
         ('num_reviews', CountVectorizer(token_pattern='\d+'), 'num_reviews'),
         ('avg_rating', CountVectorizer(token_pattern='\d+'), 'avg_rating'),
         ('text', TfidfVectorizer(
                stop_words='english',
                strip_accents='unicode',
                min_df=3,
                max_df=0.5,
                ngram_range=(1, 3),
                max_features=500), 'preprocessed_texts')],
        remainder='passthrough',
    )),
    ('clf', MultinomialNB())
], verbose=False)
```

Toolkit used: sklearn
Text preprocessing: stop_word removal, ngrams(1,3)
Text features: TFIDF vectors, maximum of 500 features
Additional features: CountVectorizer() for cuisines, num_reviews, and avg_rating, and OneHotEncoder() for zipcodes
Learning algorithm used: Naïve Bayes – MultinomialNB()

4. Future Work

I did try to using gensim's word2vec interface to train a word2vec model that I could use for word embeddings. I passed these feature vectors (I chose a size of 300) to an XGBoost Classifier and got a local score of 0.62015, which wasn't really an improvement over my Tfidf feature representation and I did not have much time to explore this further. Given more time, I'd like to try out additional word embeddings including pretrained GLOVE or FastText embeddings to represent the words in the text, and evaluate their performance. Ideally I would be able explore some deep learning model architectures such as a simple feed forward Neural Network or LSTM trained upon those word embeddings.

However, I think I would really like to build out a more robust pipeline for experiments and integrating the various pieces of text classification pipelines. For instance, it wasn't a trivial task to integrate word embeddings with the sklearn pipeline and column transformer I was already using. I'd also want to apply a deep learning model at the end of the pipeline instead of the fairly vanilla sklearn classifiers I used.

Finally, I would want to look into hyperparameter optimization and utilizing GridSearchCV for finding the best model parameters for the various classifiers I chose to use. I'd also want to look at how to handle imbalanced learning by either undersampling or specifying class weights for some of my models. These could probably help me to improve my leaderboard performance.

Task 6 code can be found here: https://github.com/jonchang03/cs598-dm-capstone/blob/master/task6/Task6.ipynb

References

- https://scikit-learn.org/stable/auto_examples/compose/plot_column_transformer.html#sphx-glr-auto-examples-compose-plot-column-transformer-py – example for column transformer
- https://radimrehurek.com/gensim/parsing/preprocessing.html - preprocessing library used
- https://www.kaggle.com/baghern/a-deep-dive-into-sklearn-pipelines - great resource on pipelining
- https://towardsdatascience.com/nlp-performance-of-different-word-embeddings-on-text-classification-de648c6262b -
- https://mlens.readthedocs.io/en/0.1.x/getting_started/ - reference for introductory ensembling
- https://www.kdnuggets.com/2019/09/no-free-lunch-data-science.html - No Free Lunch Theorem