README/REPORT

”TORIAS!” TEAM

Allan Zelener, Jordan Slavit, and Hernisa Kacorri

Homework 1 – Web Page Classification

NLP/ML/Web, FALL 2013

Graduate Center CUNY

Introduction/Readme

Problem Space

We chose to use the academic dataset provided by Yelp. Yelp is a business rating website. The academic dataset contains almost 14,000 different businesses and over 330,000 reviews about those businesses. The businesses range from restuarnts to flower stores and are located around the United States.

Each review consists of user generated text ranging is length from only a few characters to a few thousand characters.

It is our hypothesis that reviews written by users in different regions can be differentiated by building a classifier. Unfortunately, there are was not enough data in the set with varying regions in NY. Since reviews are also associated with universities, we assumed the same hypothesis would apply to reviews written near certain universities.

To limit the set, we chose 5 to classify reviews near 5 universities:

- Columbia
- Harvard
- Princeton
- CMU
- University of Illinios - Urbana

There are thousands of reviews for each school, but we realized that it would be more efficient to limit our set to only restaurants and only specific types of restaurants. Therefore, we chose to limit our restaurant categories to only "American" and "Chinese". We chose these two types of restaurants because there were at least 300 reviews per type near each school. By limiting our set to just these restaurant types we hypothesized that we wouldn't be learning the outliers, but instead a general solution.

We selected our data using SQL queries and simply divided the first 225 reviews per restuaurant type/school combo and assigned it to training data. We assigned the subsequent 75 reviews to the test data set.

Data Collection

The Yelp code is provided in JSON format. To join the data across businesses and reviews, we loaded the data into a MySQL database. To see the schema for the database, please see **r_yelp.sql**.

To export the data into files, we created a Python file called export_gold.py This file contains the SQL scripts that generated each of the test and train files.

Codebase

- The code consists of 3 files
- hw1.py – This is the main file that executes the training and testing on all of our 4 representations. There are different parameters needed to run the scripts. Using no parameters will run the program with simple defaults.
    - –r = which representation to use. This is numeric, 0-3 and corresponds to the 4 representations detailed below.
    - –te = test file, without this file, the default test_gold.txt will be used.
    - –tr = train label, without this file, the default train_gold.txt will be used.
    - –o = output file, the default results.txt will be used if none is populated
    - –ci = input classifier file, assumming a model has already been learned.
    - –co = output classifier file, a place to store the learned model.
    - –p = should we run the pipeline tests, default is false.
    - –it = should we run training, or just testing. Default is True, that training is done.

Examples:

Run representation 3 on the training data

```
python2.7 hw1.py -it True -r 3
```

Run representation 3 and export the model

```
python2.7 hw1.py -it True -r 3 –co outputClassifier3.pkl
```

Run the representation 3 and use the existing model

```
python2.7 hw1.py –it False -r 3 –ci outputClassifier3.pkl
```

Run the representation 3 and export the file to a different file

```
python2.7 hw1.py -it True -r 3 -o results_3.txt
```

To Run all our representations, execute **sh runAll.sh**

This will use the default train data and execute all 4 representations. The results are ouputted into 4 files

- 0_representation.txt
- 1_representation.txt
- 2_ representation.txt
- 3_ representation.txt

Required Python Libraries
Nltk
Scipy
Sklearn
Numpy
Argparse

Representations and Experiments

We evaluated 4 document representations in this homework. We started with a baseline and came up with 3 more routines to generate a document representation to investigate how changing parameters such use of different tokenizers and … etc. could affect the classification task.
All 4 representations feed the same multinomial Naive Bayes classifier that was fine-tuned on the training data and the adopted document representation, which learns class prior probabilities (fit-prior = 'True') and has a Laplace/Lidstone smoothing parameter (alpha = 0.1 or alpha = 0.01).

Representation 1

We finetuned on the training data 5 parameters of the sklearn TfidfVectorizer by exploring the following grid space:

```
parameters = 'tfidf__min_df': (0.5,0.75,1, ),
          'tfidf__max_features': (None, 5000,10000,30000,),
          'tfidf__ngram_range': ((1,1), (1,2),),
          'tfidf__use_idf': (True, False),
          'tfidf__norm': ('l1', 'l2'),
           'clf__alpha': (.01,0.1),
           'clf__fit_prior': (True, ),
```

We chose as our first representation the one with the higher cross-validation (number of folds 3) score on the training data at 54.5% with alpha parameter = 0.1 for the classifier. So, the first representation is a TF-IDF matrix of unigram features, where the term vectors are normalized with the l2 norm. It resulted in a total of 9237 features. When building the vocabulary terms that have a term frequency strictly lower than 1 were ignored.

```
vectorizer = TfidfVectorizer(max_features=None,
                             min_df=1,
                             ngram_range=(1, 1),
                             norm='l2'
                             use_idf=True)
```

Representation 2

We replaced the TfidfVectorizer with a CountVectorizer and a custom weighting scheme using subcategory based entropy. The core idea is that if we observe additional categories that are well correlated with the objective of the classification we can increase the weight of terms that appear frequently in only a few categories. We accomplish this with the inverse of the following category entropy weight:

$$-\sum_c p(c|t)\log p(c|t) = -\sum_c \frac{n_{c,t}}{df(t)}\,(\log n_{c,t} - \log df(t))$$
$$= \log(df(t)) - \frac{1}{df(t)}\sum_c n_{c,t}\log n_{c,t}$$

where $df(t)$ is document frequency of term t and $n_{c,t}$ is the number of documents of category c containing term t. Then as before we tuned some of the parameters using GridSearch:

parameters = 'vect__max_df': (0.5,),
'vect__max_features': (None, 5000,10000,30000,),
'vect__ngram_range': ((1, 1),(1,2),),
'catent__norm': ('l1','l2'),
'clf__alpha': (.01,0.1),
'clf__fit_prior': (True,)

Again, we chose for our second representation the one with the higher cross-validation (number of folds 3) score on the training data at 53.5% using alpha=0.1 for the classifier. The second representation is an inverse category entropy matrix of unigram features, where the term vectors are normalized with the l2 norm. This representation resulted in a total of 4725 features. When building the vocabulary terms that have a term frequency strictly higher than 0.5 were ignored.

```
featureset = CountVectorizer(max_df=0.5,
                             max_features=None,
                             ngram_range=(1,1))
vectorizer = InvCatEntTransformer(train_subcats,
                                  norm='l2')
```

Representation 3

We overrided the default string tokenization step of the TfidfVectorizer with NLTK WordNetLemmatizer() while preserving the preprocessing and n-grams generation steps. Both unigrams and bigrams were considered in the feature extraction. We further used a stop words list and when building the vocabulary. We finetuned on the training data the rest of the parameters of the sklearn TfidfVectorizer by exploring the following grid space:

parameters = 'tfidf__min_df': (1,),
'tfidf__max_features': (None, 5000,10000,30000,),
'tfidf__ngram_range': ((1,2),),

```
'tfidf__tokenizer': (LemmaTokenizer(), ),
'tfidf__use_idf': (True,False),
'tfidf__norm': ('l1','l2'),
'tfidf__stop_words': ('english',),
'clf__alpha': (.01,0.1),
'clf__fit_prior': (True, )
```

We ended up with a third document representation (with cross-validation score 53.9%) that is a TF-IDF matrix of unigram and bigram features (totally 65775), where the term vectors are normalized with the l2 norm. Terms that have a term frequency strictly lower than 1 were ignored.

```
vectorizer = TfidfVectorizer(ngram_range=(1, 2),
                             norm='l2',
                             tokenizer=LemmaTokenizer(),
                             min_df=1,
                             stop_words='english',
                             max_features=None,
                             use_idf=True)
```

Representation 4

Last we wanted to apply the NLTK implementation of LancasterStemmer(), so we overrided the default string tokenization step of the TfidfVectorizer with it while preserving the preprocessing and n-grams generation steps. Instead of unigrams and bigrams this time we experimented with features made of character n-grams, and particulary we considered 4-grams and 5-grams. We finetuned on the training data the rest of the parameters of the sklearn TfidfVectorizer by exploring the following grid space:

```
parameters = 'tfidf__min_df': (0.5,0.75,1, ),
        'tfidf__max_features': (None, 5000,10000,30000,),
        'tfidf__ngram_range': ((4,4), (5,5)),
        'tfidf__tokenizer': (StemTokenizer(), ),
        'tfidf__use_idf': (True,False),
        'tfidf__norm': ('l1','l2'),
        'tfidf__stop_words': ('english', None),
        'tfidf__analyzer':('char_wb',),
        'clf__alpha': (.01,0.1),
        'clf__fit_prior': (True, )
```

So, our last document representation (with cross-validation score 54.8%) that is a TF-IDF matrix of features made of 4-gram characters (totally 14896), where the term vectors are normalized with the l2 norm. Terms that have a term frequency strictly lower than 1 were ignored.

```
vectorizer = TfidfVectorizer(analyzer='char_wb',
                             ngram_range=(4, 4),
                             norm='l2',
                             tokenizer= StemTokenizer(),
                             min_df=1,
                             stop_words='english',
                             max_features=None,
                             use_idf=True)
```

Representation Comparison

Table 1, is given to compare the performance of our classifier, using the features extracted by each of the adopted document representations. We report the cross-validation scores in the training data and accuracy in the testing data using all of the features resulted during the vectorization of the reviews and second using only the top 5000 features, ordered by term frequency across the training "corpus", for each of the representations. The sparsity of the feature space in both cases is given. We observe that the third representation performs slightly better for the given test data.

Table 1: Comparison of implemented document representations.

	Cross-Validation [Train Data]	Number of Features	Accuracy [all features]	Accuracy [top 5000 features]
Rep.1	54.5%	9237 non-zero:86	62.9%	61.8% non-zero:83
Rep.2	53.5%	4725 non-zero:72	60.5%	60.5% non-zero:72
Rep.3	**53.9%**	**65775 non-zero:122**	**64.8%**	**62.6% non-zero:62**
Rep.4	54.8%	14896 non-zero:315	62.4%	59.2% nonzero:292

Linguistics Review and Conclusions

Our initial interest in classifying restaurant reviews based on location was not merely to maximize classification accuracy but to study the factors that most contribute to increased accuracy. As such our ideal representation would highlight the most discriminative features and identify the nature of their contribution. Our goal was to identify linguistic differences in regional writing styles but we soon realized that other factors such as the prior distribution of restaurant cuisines given the region, region identifying terms, and sparsity of informal writing were also contributing factors.

We developed several representations that dealt with these issues individually and indicate the potential contribution of these factors, however an ideal representation would explain all factors simultaneously. One way this could work is by assigning factor-specific weights to each feature, as in our category-based entropy weighted representation where the weights are based on the distribution of restaurant cuisines conditioned on the given term. A mixture of weights like these would help explain how for a given example, what may be the explanation for its classification. Also automatically discovering possible factors from web metadata and linguistic models would further improve the possible explanation.

However simply using tokens that appear in the original text does not appear to be sufficient due to the use of informal language and structure. This is what motivated us to experiment with stemming, lemmaization, and character n-grams. In order to understand a feature's contribution we require an accurate count independent of minor variation which requires a more compact feature representation. Our relatively shallow corpus of 1125 training examples also motivates significant dimensionality reduction to find terms that contribute to the best explanations.

Ultimately the appeal of this representation is not to maximize accuracy on an arbitraily chosen testing set but to move towards utilizing the tools of machine learning for statistical analysis and modeling in a way that is better suited for data analysis tasks in linguistics and other fields. An ideal feature representation should by its construction represent the underlying relationships of its source domain.