Task 6: Hygiene Prediction

Introduction

I've completed the task using python language and did it in iPython notebook (http://ipython.org/notebook.html). The original iPython notebook with source codes can be found in my dropbox: https://www.dropbox.com/s/hxdee7rburmmnrn/Capstone%20project%206.html?dl=0 . The report reflects the sequence of attempt and the achieved results.

1. Straightforward solution

First of all, I set a goal to myself to build a classifier with minimum effort and send it for evaluation to the server in order to make sure that I understood the output format correct and the target of task as well.

For that I read the file, split the input file into train set and data for prediction and pass it through TF-IDF vectorizer from python scikit-learn library (http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html). For TF-IDF vectorizer I specified to use built-in English stop-words set and passed also min and max threshold for DF to filter out misspelling (words with only once occurrence) and corpus specific stop-words (words that occur more than 80% of corpus documents). TF-IDF vectorizer uses the default simple tokenizer based on built-in RegExp constant for delimiters. In other words it has the unigrams text representation.

As classifier I chosen random forest classifier (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) that fits a number of decision trees to find the best proportion between over-fitting and precision. From my experience it has quite good precision in average for wide type of tasks.

This solution gave me 0.556337904331 and 11th place when I sent it. "Good start for so simple solution" was my thought.

2. Preprocessing improvements

The first idea on the way to improve my solution I decided to focus on pre-processing, particularly adding a good tokenizer and stemmer. Also my guess was that the punctuation doesn't play a role in comparison of review so I decided to purge them out. Only one exception I provided for symbol ' since it is used widely in reductions and it most likely will lead that words will be disturbed. After removing of all punctionation I also made all words to lower case.

I analyzed all existed in python nltk library tokenizers and stopped on StanfordTokenizer that actually is an interface to java implementation from Stanford set of tools for natural language processing (http://nlp.stanford.edu/software/tokenizer.shtml). It seemed to me as the most enhanced tokenizer, especially it's ability to split hard cases with reductions like "'I'll" -> ["I", "'ll"]. Another tokenizer that looks promisingly was TweetTokenizer, since it should tokenize smiles, like ":)" but unfortunately it as just released and experienced problems simply to compile it.

I've already investigated in the task the influence of different stemmers from nltk toolkit and as mentioned the best result I saw after simple Porter algorithm and with The WordNet Lemmatizer. I stopped on Porter stemmer for this task.

The last preprocessing step was to replace reduction like "'ll" to full variance "will". Because I anticipated that the preprocessing step will take a long time (here is a lot of work to do). I used Apache Spark to perform transformation in parallel.

I applied to preprocessed text the same sequence of feature extraction: TF-IDF with stop-words filtering and max filter.

I found on the website of scikit-learn library a good example with applying of most frequent used classifier for text mining (http://scikit-learn.org/stable/auto_examples/text/document_classification_20newsgroups.html#example-text-document-classification-20newsgroups-py). Based on results in this example I found out that ridge classifier (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeClassifier.html) that showed pretty good performance on text mining and tf-idf feature extraction. Thus I decided try to apply it here instead of random forest classifier.

This approach gave me 0.557980182843 and 20th place at time of submission. I guess this result was due to the use of enhanced pre-processing particularly due to a good tokenizer.

3. Attempts to apply other classifiers

For next turn I decided to analyze possibility to achieve better score by replacing classifier. I tried following options:

- Passive-Aggressive classifier (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.PassiveAggressiveClassifier.html).
- Naive Bayes classifier for multinomial models (http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html)
- Naive Bayes classifier for multivariate Bernoulli models (http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html). Like

MultinomialNB, this classifier is suitable for discrete data. The difference is that while MultinomialNB works with occurrence counts, BernoulliNB is designed for binary/boolean features.

All these attempts failed, in all cases I've got significant lower score, from 0.54386605395 to maximum 0.552773951129.

4. Union text features with additional information

Courses' authors proposed to not stick only to text extracted features but also to use additional information provided in a separate file. It contains number of reviews, average rate and postal index.

Quite usual technic to build features vector for categorical data is to use a binary one-hot (aka one-of-K) coding: one boolean-valued feature is constructed for each of the possible string values that the feature can take on. For this purpose scikit-learn library provides DictVectorizer (http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.DictVectorizer.html). I select categories and zip code as categorical data. For average rating and number of reviews I didn't use any additional feature extractor or transformer. I concatenated all these four type of features simple as additional column to already extracted feature matrix based on TF-IDF.

This approach have me score 0.560832638751 and 22th place on the board.

5. Selection of K best features and fine tuning of classifier

I noticed that the number of features after adding additional parameters is about 26.000 features, this is quite big space and most likely it could lead to low level of generalization. Based on this idea I decided to try to select lower number of feature, the criteria were used information level chi2 (http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html#sklearn.feature_selection.chi2). For this purpose ski-learn provides convenient class SelectKBest (http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html). In addition to this step I thought that most likely fine tuning of main parameters could give me additional score. As well in this case scikit-learn has an excellent tool for grid search (http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html).

I was quite surprised but grid search showed that the best number of features was in my case only 20. For classifier's parameters grid search showed slightly different parameters than I chosen from start.

This solution reached my best 0.562726857045 and 18th place at the time of submission.