UNIVERSITY OF MICHIGAN
Department of Electrical Engineering and Computer Science
EECS 445 Introduction to Machine Learning
Winter 2015

Project 1: EVERYONE'S A CRITIC
Issued: 01/23 Due: 02/6 at 9am

1 Introduction

In this project you will be working with Twitter data. Specifically, we have supplied you with a number of tweets that are reviews/reactions to movies,

e.g., *"@nickjfrost just saw The Boat That Rocked/Pirate Radio and I thought it was brilliant! You and the rest of the cast were fantastic!* < 3*"*.

You will learn to automatically classify such tweets as either positive or negative reviews. (Please note that these data were selected at random and thus the content of these tweets do not reflect the views of the course staff.) You will employ Support Vector Machines (SVMs), a popular choice for a large number of classification problems. We will cover SVMs in Lectures 5 and 6. For this project (and subsequent projects), we will use **Python2.7** with the **numpy** and **scikit_learn** packages. We strongly recommend **Anaconda** (available from `http://continuum.io/downloads` for all platforms). It comes with both packages already installed.

To get started, download *project1.zip* from CTools. It contains the following data files, *tweets.txt*, *labels.txt* and *held_out_tweets.txt*. The file *tweets.txt* contains 630 tweets about movies. Each line in the file contains exactly one tweet, so there are 630 lines in total. If a tweet praises or recommends a movie, it is classified as a positive review and labeled $+1$, otherwise it is classified as a negative review and labeled -1. These labels are provided separately in *labels.txt*. These labels are ordered, i.e., the label for the i^{th} tweet in *tweets.txt* corresponds to the i^{th} number in *labels.txt*. We have also provided you with a file $held\_out\_tweets.txt$ containing 70 tweets for which we have withheld the labels.

The python file *proj1_skeleton.py* contains skeleton code for the project. It has the following functions implemented.

- `read_vector_file(fname)`
- `write_label_answer(vec, outfile)`

It also provides specifications for other functions that you will implement.

- `extract_dictionary(infile)`
- `extract_feature_vectors(infile, word_list)`
- `cv_performance(clf, X, y, k=5, metric='accuracy')`
- `select_param_linear(X, y, k=5, metric='accuracy')`
- `select_param_rbf(X, y, k=5, metric='accuracy')`
- `conf_interval(clf_trained, X_test, y_test, metric='accuracy')`

2 Feature Extraction [15pts]

We will use a *bag-of-words* model to convert each tweet into a feature vector. A bag-of-words model treats a text file as a collection of words, disregarding word order. The first step in building a bag-of-words model involves building a "dictionary". A dictionary contains all of the **unique** words in the text file. For this project, we will be including punctuations in the dictionary too. For example, a text file containing *"John likes movies. Mary likes movies2!!"* will have a dictionary {'John':0, 'Mary':1, 'likes':2, 'movies':3, 'movies2':4, '.':5, '!':6}. Note that the (key,value) pairs are (word, index) where the index keeps track of the number of unique words (size of the dictionary). This dictionary will come in handy later.

Given a dictionary containing d unique words, We can transform the n variable-length tweets into n feature vectors of length d, by setting the i^{th} element of the j^{th} feature vector to 1 if the i^{th} dictionary word is in the j^{th} tweet, and 0 otherwise.

(a) Start by implementing the `extract_dictionary(infile)` function. You will use this function to read all unique words contained in *tweets.txt* into a dictionary (as in the example above). Initially, you can take a simplistic approach to this problem, treating any string of characters (that does not include a space) as a "word". You should also extract and include all unique punctuations. Your function should return dictionary of d unique words/punctuations. You can handle punctuations in a number of different ways. For simplicity you can treat each punctuation mark as a separate entry in the list. [Hint: you might find the list returned by `string.punctuation` and the `replace()` method useful]

(b) Next, implement the `extract_feature_vectors(infile, word_list)` function. For each tweet i, construct a feature vector of length d, where the j^{th} entry in the feature vector is 1 if the j^{th} word/punctuation in the dictionary is present in tweet i, or 0 otherwise. For n tweets save the feature vectors in an (n, d) feature matrix, where the rows correspond to tweets (examples) and the columns correspond to words (features). Maintain the order of the tweets as the appear in the file during this step.

(c) Using the functions you implemented in (a) and (b) transform the tweets in *tweets.txt* into a (630,d) feature matrix. Then, use the `read_vector_file(fame)` helper function to read the corresponding labels into a (630,) array. Next, split the feature matrix and corresponding labels into your training and test sets. **The first** 560 **tweets will be used for training and the last** 70 **tweets will be used for testing.**

 All subsequent operations will be performed on these data.

3 Hyperparameter Selection for a Linear-Kernel SVM [15pts]

Next, we will learn a classifier to separate the training data into positive and negative tweets. For the classifier, we will use SVMs with two different kernels: linear and radial basis function (RBF). Note we will make use of the `sklearn.svm.SVC` class. For this project, we will explicitly set only three of the initialization parameters of `SVC()`: the `kernel`, `gamma`, and `C`. In addition, we will use two methods in the `SVC` class: `fit(X,y)` and `predict(X)`.

As discussed in Lectures 5 & 6, SVMs have hyperparameters that must be set by the user. For both linear- and RBF-kernel SVMs, we will select the hyperparameters using 5-fold cross-validation (CV). Using 5-fold CV we will select the hyperparameters that lead to the 'best' mean performance across all 5 folds. The result of a hyperparameter selection often depends upon the choice of performance measure. Here, we will consider the following performance measures: **Accuracy**, **F1-Score**, **AUROC**, **Precision**, **Sensitivity**, and **Specificity**.

All measures, except sensitivity and specificity, are implemented in `sklearn.metrics` library, you can use `sklearn.metrics.confusion_matrix` to calculate the other two.

(a) To begin, implement the function `cv_performance(clf, X, y, k=5,metric='accuracy')` as defined in the skeleton code. Here you will make use of the `fit(X,y)` and `predict(X)` methods in the SVC class. The function returns the mean k-fold CV performance for the performance metric passed into the function. The default metric is 'accuracy', however your function should work for all of the possible metrics listed above.

You may have noticed that the proportion of the two classes (positive and negative) are not equal in the training data. When dividing the data into folds for CV, you should try to keep the class proportions roughly the same across folds.

You must implement this function without using the `scikit_learn` implementation of CV. However, you may employ the class `sklearn.cross_validation.StratifiedKFold()` for splitting the data.

In your write-up, briefly describe why it might be beneficial to maintain class proportions across folds.

(b) Now implement the `select_param_linear(X, y, k=5,metric='accuracy')` function to choose a value of C for a linear SVM based on the training data and the specified metric. Your function should call your CV function (implemented earlier) passing in instances of `SVC(kernel='linear', C=c)` with different values for C, e.g., $C = [10^{-3}, 10^{-2}, ..., 10^{2}]$.

(c) Finally, using the training data from Section 2 and the functions implemented here, find the best setting for C for each performance measure mentioned above. Report your findings in tabular format. The table should have two columns, one for the names of the performance measures and one for the corresponding values of C. You `select_param_linear(X, y, k=5,metric='accuracy')` function returns the 'best' C given a range of values, how does the 5-fold CV performance vary with C?

4 Hyperparameter Selection for an RBF-kernel SVM [10pts]

Similar to the hyperparameter selection for a linear-kernel SVM, you will perform hyperparameter selection for an RBF-kernel SVM.

(a) Describe the role of the additional hyperparameter $gamma$ for an RBF-kernel SVM.

(b) Implement the `select_param_rbf(X, y, k=5, metric='accuracy')` function to choose a setting for C and $gamma$ via a grid-search. Your function should call your CV function (implemented earlier) passing in instances of `SVC(kernel='rbf', C=c, gamma=g)` with different values for C and $gamma$. Explain what kind of grid you used and why.

(c) Finally, using the training data from Section 2 and the function implemented here, find the best setting for C and $gamma$ for each performance measure mentioned above. Report your findings in tabular format. The table should have three columns, for performance measure, C and $gamma$. How does the CV performance vary with the hyperparameters of the RBF-kernel SVM?

Based on the results you obtained in Section 3 and Section 4, choose a hyperparameter setting for the linear-kernel SVM and a hyperparameter setting for the RBF-kernel SVM. Explain your choice.

Using the training data extracted in Section 2 and the `fit` method in the SVC class, train a linear- and an RBF-kernel SVM with your chosen settings.

5 Test Set Performance - Bootstrap Confidence Intervals [15pts]

In this section you will apply the two classifiers learned in the previous section to the test data from Section 2. Once you have generated labels for the test data you will measure performance. However, instead of measuring performance as a single point, you will calculate the 95% confidence interval for each performance estimate.

We will generate the 95% confidence interval using bootstrapping (sampling with replacement). This method is described below:

For a set of n test examples, randomly sample n examples (sampling with replacement). This represents a bootstrap sample of the test set. Apply your classifier to the bootstrap sample and calculate performance as before. Repeat this process 1000 times, generating 1000 bootstrap samples and 1000 estimate of performance (for a given metric). The bootstrap 95% confidence interval is then given by the 2.5 and 97.5 percentiles of these 1000 values.

(a) Implement `conf_interval(clf, X, y, metric='accuracy')` which returns the value of a performance measure along with its 95% confidence interval (calculated using bootstrapping), given data and a classifier. Note that the instance of `SVC` passed into the function as `clf` must already have been fit to the training data. In this function you will make use of the `predict(X)` method in the `SVC` class.

(b) Use your function from part 5(a) and the two classifiers from the previous section to measure performance on the test data. Report the results. Be sure to include the name of the performance metric employed, the performance on the test data and the corresponding confidence interval. How does the test performance of your two classifiers compare?

6 Feature Engineering [20pts]

In this Section, you will try to improve the performance of your classifier by changing the feature vectors. You could do this a number of different ways. E.g., try using a different method for extracting words from the tweets, or try using only a subset of the features, or try using the number of times a word occurs in a tweet as the feature rather than just a 0 or 1 indicating presence or absence. You could also try scaling or normalizing the data.

Implement one alternative method of your design for extracting feature vectors and compare its performance on the test data. Use cross validation to select the hyperparameters. Your report should contain:

(a) A clear explanation of how you changed the feature set, including the code you used to compute the features.

(b) An explanation of why you thought this alternative feature set might improve performance.

(c) A description of the experiment you conducted to compare it to the original feature-computation method, and the results of that experiment.

Note: it's OK if it turns out that your new method performs worse than the original one, just explain and document your results.

7 Explanation [10pts]

Suppose you had to explain the results of your learning to someone not familiar with computer science or machine learning (e.g., a businessman or film director). Using a single paragraph plus one optional graph or table, explain what you have found. Do not describe the classifier's performance, but rather what it has *learned* about tweets.

8 Held-out data [15pts]

As with *tweets.txt*, build a feature matrix from *held_out_tweets.txt*. Note that, this file may contain words that are not in *tweets.txt*. Ignore such words and use the dictionary you built when reading *tweets.txt* to read this file.

Predict the labels for the held-out data using your 'best' classifier you have found thus far and write the labels using `write_label_answer(vec, outfile)` helper function to a text file. The name of the text file must be same as your uniqname. (For example, *srayand.txt*). Note that you can use *all* of the data from *tweets.txt* to train your final classifier (you are no longer restricted to just the first 560 tweets).

REMEMBER Submit your project report by 9am on Feb. 6th to CTools. Include your code as an appendix (copy and pasted) in your report. Upload your file with the labels to the held-out data as a separate .txt file.