HMC CS 158, Fall 2017
Problem Set 6 Project: Everyone's a Critic

Goals:

- To practice implementing a full ML pipeline for one application of machine learning: sentiment analysis.
- To investigate the performance of various SVM classifiers across several metrics and through hyperparameter selection and bootstrapping.
- To try your hand at making better predictions!

For this assignment, you can work individually though you are encouraged to work with a partner. You should sign up for partners on Canvas (People → PS6 Groups). If you are looking for a partner, try Piazza. If, after trying Piazza, you are having trouble finding a partner, e-mail Jessica.

Submission

You should submit any answers to the exercises in a single file `writeup.pdf`. This writeup should include your name and the assignment number at the top of the first page, and it should clearly label all problems. Additionally, cite any collaborators and sources of help you received (excluding course staff), and if you are using slip days, please also indicate this at the top of your document.

Your code should be commented appropriately. The most important things:

- Your name and the assignment number should be at the top of each file.
- Each class and method should have an appropriate doctsring.
- If anything is complicated, it should include some comments.

There are many possible ways to approach the programming portion of this assignment, which makes code style and comments very important so that staff can understand what you did. For this reason, you will lose points for poorly commented or poorly organized code.

When you are ready to submit, make sure that your code compiles and remove any debugging statements. Then rename the top-level directory as `<username1>_<username2>_ps6`, with the usernames in alphabetical order (e.g. `hadas_yjw_ps6`). This directory should include the electronic version of your writeup and main code, any files necessary to run your code (including any code and data provided by staff), and follow the same structure as the assignment directory provided by staff. So, for this assignment, your directory should have the following structure:

- `<username1>_<username2>_ps6/`
  - `data/`
    - `tweets.txt`
    - `labels.txt`
    - `held_out_tweets.txt`
    - `<username1>_<username2>_twitter.txt` (extra credit)
  - `source/`
    - `twitter.py` (with your modifications)
  - `writeup.pdf`
  - `twitter.pdf` (pdf printout of `twitter.py`)

This assignment is adapted from course material by Jenna Wiens (UMich).

Package this directory as a single `<username1>_<username2>_ps6.zip` file, and submit the archive. Additionally, to aid the staff in grading, submit your `pdf`'s as separate files, and submit your predictions as a separate file as well.

Introduction

In this project, you will be working with Twitter data. Specifically, we have supplied you with a number of tweets that are reviews/reactseions to movies[1],

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. To do this, you will employ Support Vector Machines (SVMs), a popular choice for a large number of classification problems.

Download the code and data sets from the course website. It contains the following data files:

- `tweets.txt` contains 630 tweets about movies. Each line in the file contains exactly one tweet, so there are 630 lines in total.
- `labels.txt` contains the corresponding labels. 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 ordered, i.e. the label for the i^{th} tweet in `tweets.txt` corresponds to the i^{th} number in `labels.txt`.
- `held_out_tweets.txt` contains 70 tweets for which we have withheld the labels.

Skim through the tweets to get a sense of the data.

The python file `twitter.py` contains skeleton code for the project. Skim through the code to understand its structure.

1 Feature Extraction [2 pts]

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, 'likes':1, 'movies':2, '.':3, 'Mary':4, 'movies2':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).

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.

[1] 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. :-)

(a) **(1 pts)** We have implemented `extract_words(...)` that processes an input string to return a list of unique words. This method takes a simplistic approach to the problem, treating any string of characters (that does not include a space) as a "word" and also extracting and including all unique punctuations.

Implement `extract_dictionary(...)` that uses `extract_words(...)` to read all unique words contained in a file into a dictionary (as in the example above). Process the tweets in the order they appear in the file to create this dictionary of d unique words/punctuations.

(b) **(1 pts)** Next, implement `extract_feature_vectors(...)` that produces the bag-of-words representation of a file based on the extracted dictionary. That is, 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 in the dictionary is present in tweet i, or 0 otherwise. For n tweets, save the feature vectors in a feature matrix, where the rows correspond to tweets (examples) and the columns correspond to words (features). Maintain the order of the tweets as they appear in the file.

In `main(...)`, we have provided code to read the tweets and labels into a $(630, d)$ feature matrix and $(630,)$ label array. **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 this data.**

2 Hyperparameter Selection for a Linear-Kernel SVM [8 pts]

Next, you will learn a classifier to separate the training data into positive and negative tweets. Let us start off with a standard (linear-kernel) SVM as implemented by `sklearn.svm.SVC` with `kernel='linear'`.

SVMs have hyperparameters that must be set by the user. For the linear-kernel SVM, you will explicitly only search across `C`. Using 5-fold cross-validation (CV), you will select the hyperparameter that leads to the 'best' mean performance across all 5 folds.

(a) **(2 pts)** The result of a hyperparameter selection often depends upon the choice of performance measure. Here, you will consider the following performance measures: **accuracy**, **F1-Score**, **AUROC**, **precision**, **sensitivity**, and **specificity**.

Implement `performance(...)`. Most measures are implemented in `sklearn.metrics` library. You can use `sklearn.metrics.confusion_matrix(...)` to calculate the missing metrics. When using the `sklearn` functions, **watch the order of your parameters, and make sure to use the `labels` parameter if you want the confusion matrix cells to correspond to the order presented in-class**.

Take a look at `cv_performance(...)`, which returns the mean k-fold CV performance for the performance metric passed into the function. As usual, we use `SVC.fit(X,y)` to train our SVM, but in lieu of using `SVC.predict(X)` to make predictions, we use `SVC.decision_function(X)`, which returns the (signed) distance of the samples to the separating hyperplane.

(b) **(1 pts)** 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. In your write-up, briefly describe why it might be beneficial to maintain class proportions across folds. Then, in `main(...)`, use

`sklearn.cross_validation.StratifiedKFold(...)` to split the data for 5-fold CV, making sure to stratify using only the training labels.

(c) **(2 pts)** Now, implement `select_param_linear(...)` to choose a setting for C for a linear SVM based on the training data and the specified metric.

(d) **(3 pts)** Finally, using the training data from Section 1 and the functions implemented here, find the best setting for C for each performance measure mentioned above. Report your findings in tabular format (up to the fourth decimal place):

C	accuracy	F1-score	AUROC	precision	sensitivity	specificity
10^{-3}						
10^{-2}						
10^{-1}						
10^{0}						
10^{1}						
10^{2}						
best C						

Your `select_param_linear(...)` function returns the 'best' C given a range of values. How does the 5-fold CV performance vary with C and the performance metric? If you observe anything surprising, comment on what might have caused these results.

3 Hyperparameter Selection for an RBF-kernel SVM [8 pts]

Similar to the hyperparameter selection for a linear-kernel SVM, you will perform hyperparameter selection for an RBF-kernel SVM (`kernel='rbf'`). This time, you will search across `gamma` and `C`.

(a) **(2 pts)** The `scikit-learn` documentation tells us that the RBF kernel is specified by $\exp(-\gamma||\boldsymbol{x} - \boldsymbol{z}||^2)$, where $\gamma > 0$. Describe the role of the additional hyperparameter γ for an RBF-kernel SVM. How does γ affect generalization error?

(b) **(3 pts)** Implement `select_param_rbf(...)` to choose a setting for C and γ via a grid search. Your function should call `cv_performance(...)`, passing in instances of `SVC(kernel='rbf', C=c, gamma=gamma)` with different values for `C` and `gamma`. Explain what kind of grid you used and why.

(c) **(3 pts)** Finally, using the training data from Section 1 and the function implemented here, find the best setting for C and γ for each performance measure mentioned above. Report your findings in tabular format. This time, because we have a two-dimensional grid search, report only the best score for each metric, along with the accompanying C and γ setting:

metric	score	C	γ
accuracy			
F1-score			
AUROC			
precision			
sensitivity			
specificity			

How does the CV performance vary with the hyperparameters of the RBF-kernel SVM?

4 Test Set Performance – Bootstrap Confidence Intervals [5 pts]

In this section, you will apply the two classifiers learned in the previous sections to the test data from Section 1. Once you have predicted 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 works as follows: 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 (trained) classifier to the bootstrap sample and calculate performance as before. Repeat this process t times, generating t bootstrap samples and t estimates of performance (for a given metric). The bootstrap 95% confidence interval is then given by the 2.5 and 97.5 percentiles of these t values. For this problem, use $t = 1000$.

(a) **(1 pts)** Based on the results you obtained in Section 2 and Section 3, choose a hyperparameter setting for the linear-kernel SVM and a hyperparameter setting for the RBF-kernel SVM. Explain your choice.

Then, in `main(...)`, using the training data extracted in Section 1 and `SVC.fit(...)`, train a linear- and an RBF-kernel SVM with your chosen settings.

(b) **(2 pts)** Implement `performance_CI(...)` which returns the value of a performance measure along with its 95% confidence interval (calculated using bootstrapping), given the test data and a trained classifier.

(c) **(2 pts)** For each performance metric, use `performance_CI(...)` and the two trained linear- and RBF-kernel SVM classifiers 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 do the test performance of your two classifiers compare?

5 Feature Importances [3 pts]

(a) **(1 pts)** Now let us consider your trained linear-kernel SVM. How would you use the coefficients of the learned classifier to find the most important features?

(b) **(1 pts)** In order of rank, e.g. from most important to least important, list the 10 most important features for tweets with positive sentiment and the 10 most important features for tweets with negative sentiment. Comment on the resulting features.

(c) **(1 pts)** List one limitation of this approach for determining feature importance.

6 Explanation [4 pts]

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 use technical jargon or describe the classifier's performance; rather, describe what your model has *learned* about tweets.

7 Extra Credit: Contest [+3 pts]

Now let us experiment (and push yourselves) a bit more by holding a Twitter competition.

For this contest, we have provided a held-out test data set, `held_out_tweets.txt`. Staff has the true labels for these tweets, and your goal is to minimize the prediction error on this data set.

You should submit your predictions in a file called `<username1>_<username2>_twitter.txt` in your submission. This file should include your predictions on the test data set, which can be written using `write_label_answer(...)`. Additionally, **include a short report in your writeup that describes what you did, including any rationale**. Staff will compare your predictions to the ground truth and award extra credit points based on the performance of your classifier.

Ground rules:

- *Your predictions must be auto-generated from your code and use only the provided data sets.* In particular, no predicting manually (which defeats the entire purpose of this exercise).
- *You must use a SVM classifier.* Now that you are familiar with SVM classifiers applied to this dataset, try your hand at improving your model. Feel free to experiment with feature engineering or model tweaking. You must try at least one variation on the exercises in this assignment. Personally, for this dataset, I find feature engineering to be more fun, but you can try tinkering with the existing features or the model.
- 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).

Held-out Data:

- As with `tweets.txt`, build a feature matrix from `held_out_tweets.txt`. Use the dictionary you built when reading `tweets.txt`. Note that `held_out_tweets.txt` may contain words that are not in `tweets.txt` – I suggest simply ignoring such words.
- As with the rest of the assignment, instead of writing binary labels, you should write *continuous* scores using `SVC.decision_function(X)`.

Note: It is OK if it turns out that your new method performs worse than the original one – just explain and document your results. (You probably will not win the contest however...)