CSM146, Fall 2018
Problem Set 3: Computational Learning Theory, Kernel, SVM
Due Nov 27, 2018 at 11:59 pm

Submission

- Submit your solutions electronically on the course Gradescope site as PDF files.
- If you plan to typeset your solutions, please use the LaTeX solution template. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app.

Parts of this assignment are adapted from course material by Andrew Ng (Stanford), Jenna Wiens (UMich) and Jessica Wu (Harvey Mudd).

1 VC Dimension [15 pts]

We define a set of concepts

$$H = \{sgn(ax^2 + bx + c); a, b, c, \in R\},$$

where $sgn(\cdot)$ is 1 when the argument $\cdot$ is positive, and 0 otherwise. What is the VC dimension of H? Prove your claim.

2 Kernels [15 pts]

Given vectors $\boldsymbol{x}$ and $\boldsymbol{z}$ in $\mathbb{R}^2$, define the kernel $K_\beta(\boldsymbol{x}, \boldsymbol{z}) = (1 + \beta \boldsymbol{x} \cdot \boldsymbol{z})^3$ for any value $\beta > 0$. Find the corresponding feature map $\phi_\beta(\cdot)$[1]. What are the similarities/differences from the kernel $K(\boldsymbol{x}, \boldsymbol{z}) = (1 + \boldsymbol{x} \cdot \boldsymbol{z})^3$, and what role does the parameter β play?

3 SVM [20 pts]

Suppose we are looking for a maximum-margin linear classifier *through the origin*, i.e. $b = 0$ (also hard margin, i.e., no slack variables). In other words, we minimize $\frac{1}{2}||\boldsymbol{w}||^2$ subject to $y_n \boldsymbol{w}^T \boldsymbol{x}_n \geq 1, n = 1, \ldots, N$.

(a) Suppose we have two training examples, $\boldsymbol{x}_1 = (1, 1)^T$ and $\boldsymbol{x}_2 = (1, 0)^T$ with labels $y_1 = 1$ and $y_2 = -1$. What is $\boldsymbol{w}^*$ in this case?

(b) Suppose we now allow the offset parameter b to be non-zero. How would the classifier and the margin change in the previous question? What are $(\boldsymbol{w}^*, b^*)$? Compare your solutions with and without offset.

4 Twitter analysis using SVMs [50 pts]

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[2],

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:

[1] You may use any external program to expand the cubic.

[2] 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. :-)

- `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`.

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.

4.1 Feature Extraction [10 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, '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).

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) 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) 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.

(c) In `main(...)`, we have provided code to read the tweets and labels into a $(630, d)$ feature matrix and $(630, 1)$ label array. 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.**

(d) Indicate that you have finished the feature extraction and generated the train/test splits in your write-up.

4.2 Hyper-parameter Selection for a Linear-Kernel SVM [30 pts]

Next, we will learn a classifier to separate the training data into positive and negative tweets. For the classifier, we will use SVMs with the linear kernel. We will use the `sklearn.svm.SVC` class[3] and explicitly set only three of the initialization parameters: `kernel`, and `C`. As usual, we will use `SVC.fit(X,y)` to train our SVM, but in lieu of using `SVC.predict(X)` to make predictions, we will use `SVC.decision_function(X)`, which returns the (signed) distance of the samples to the separating hyperplane.

SVMs have hyperparameters that must be set by the user. For both linear 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.

(a) 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**, and **AUROC**[4].

    Implement `performance(...)`. All measures are implemented in `sklearn.metrics` library.

(b) Next, implement `cv_performance(...)` to return the mean k-fold CV performance for the performance metric passed into the function. Here, you will make use of `SVC.fit(X,y)` and `SVC.decision_function(X)`, as well as your `performance(...)` function.

    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) Now, implement `select_param_linear(...)` to choose a setting for C for a linear SVM based on the training data and the specified metric. Your function should call `cv_performance(...)`, passing in instances of `SVC(kernel='linear', C=c)` with different values for `C`, e.g., $C = 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2$.

(d) Finally, using the training data from Section 4.1 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):

[3] Note that when using SVMs with the linear kernel, it is recommended to use sklearn.svm.LinearSVC instead of sklearn.svm.SVC because the backbone of sklearn.svm.LinearSVC is the LIBLINEAR library, which is specifically designed for the linear kernel. For the sake of the simplicity, in this problem set we use sklearn.svm.SVC.

[4] Read menu `http://scikit-learn.org/stable/modules/model_evaluation.html#roc-metrics` to understand the meaning of these evaluation metrics.

C	accuracy	F1-score	AUROC
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?

4.3 Test Set Performance [10 pts]

In this section, you will apply the classifier learned in the previous sections to the test data from Section 4.1. Once you have predicted labels for the test data, you will measure performance.

(a) Based on the results you obtained in Section 4.2, choose a hyperparameter setting for the linear-kernel SVM. Then, in `main(...)`, using the training data extracted in Section 4.1 and `SVC.fit(...)`, train a linear-kernel SVM with your chosen settings.

(b) Implement `performance_test(...)` which returns the value of a performance measure, given the test data and a trained classifier.

(c) For each performance metric, use `performance_test(...)` and the trained linear-kernel SVM classifier to measure performance on the test data. Report the results. Be sure to include the name of the performance metric employed, and the performance on the test data.