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

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.

Solution: $VC(H) = 3$. We provide a geometric proof. See figure 1. The first 8 diagrams show that all possible dichotomies of three points can be shattered by the hypothesis.

To prove that there doesn't exist a set of 4 points that can be shattered by H, we should discuss two situations: 1) There are at least two points at the same position. In this case, assign one of them to be positive and another to be negative, and then they cannot be shattered. 2) The four points are all in different positions. Then, we can label them from left to right as + - + - as shown in the 9-th figure on the right bottom corner. In this case, no hypothesis in the set can separate these points.

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?

Solution: To show that K_β is a kernel, simply use the same feature mapping as used by the polynomial kernel of degree 3, but first scale $\boldsymbol{x}$ by $\sqrt{\beta}$. So, in effect they are both polynomial kernels of degree 3. If you look at the resulting feature vector, the offset term 1 is unchanged, the linear terms are scaled by $\sqrt{\beta}$, the quadratic terms are scaled by β, and the cubic terms are scaled by $\beta^{1.5}$. Although the model class remains unchanged, this changes how we penalize the features during learning (from the $||\boldsymbol{\theta}||^2$ in the objective). In particular, higher-order features will become more costly to use, so this will bias more towards a lower-order polynomial.

That is,

$$\begin{aligned} K_\beta(\boldsymbol{x}, \boldsymbol{z}) &= (1 + \beta \boldsymbol{x} \cdot \boldsymbol{z})^3 \\ &= (1 + \beta (x_1 z_1 + x_2 z_2))^3 \\ &= 1 + 3\beta (x_1 z_1 + x_2 z_2) + 3\beta^2 \left(x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2\right) \\ &\quad + \beta^3 \left(x_1^3 z_1^3 + 3 x_1^2 z_1^2 x_2 z_2 + 3 x_1 z_1 x_2^2 z_2^2 + x_2^3 z_2^3\right) \end{aligned}$$

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

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

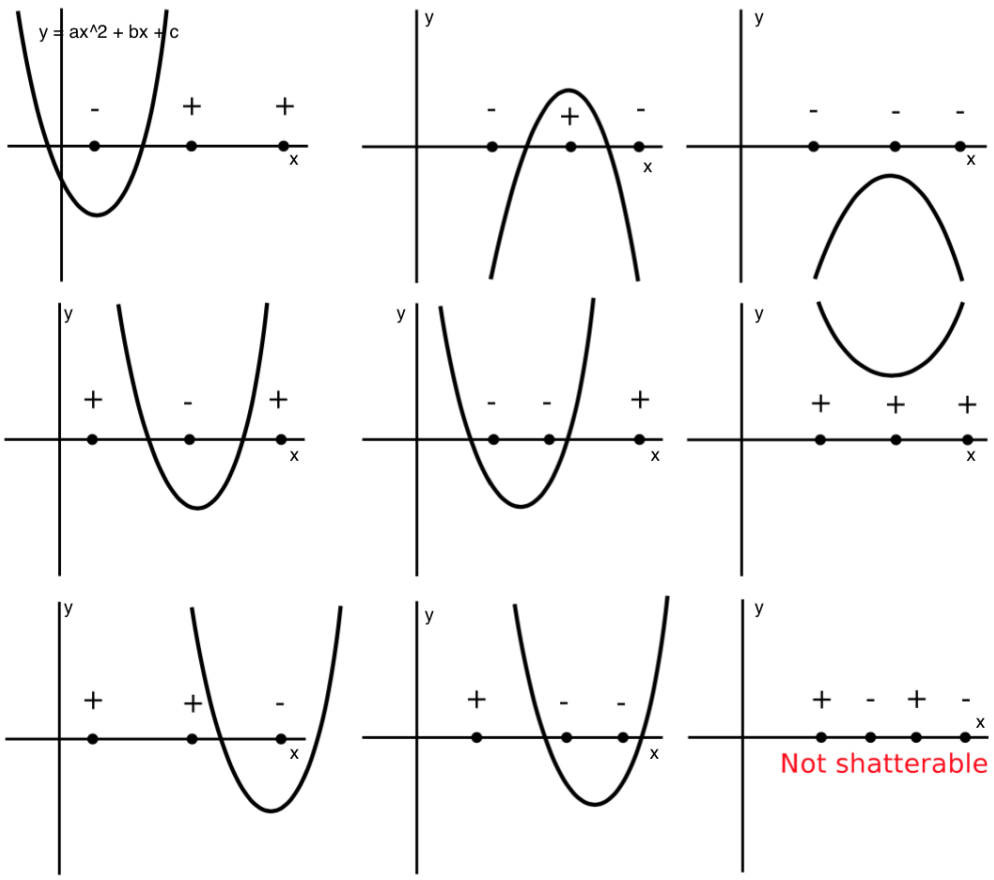

Figure 1: Geometric proof that VC(H) = 3. Note that the feature space only locates on x-axis, while the y-axis represents the value of $ax^2 + bx + c$.

so that

$$\phi_\beta(\boldsymbol{x}) = (1, \sqrt{3\beta}x_1, \sqrt{3\beta}x_2, \sqrt{3}\beta x_1^2, \sqrt{6}\beta x_1 x_2, \sqrt{3}\beta x_2^2, \sqrt{\beta^3}x_1^3, \sqrt{3\beta^3}x_1^2 x_2, \sqrt{3\beta^3}x_1 x_2^2, \sqrt{\beta^3}x_2^3)^T$$

The m^{th}-order terms in $\phi_\beta(\cdot)$ are scaled by $\beta^{m/2}$, so β trades off the influence of the higher-order versus lower-order terms in the polynomial. If $\beta = 1$, then $\beta^{1/2} = \beta = \beta^{3/2}$ so that $K_\beta = K$. If $0 < \beta < 1$, then $\beta^{1/2} > \beta > \beta^{3/2}$ so that lower-order terms have more weight and higher-order terms less weight; as $\beta \to 0$, K_β approaches $1 + 3\beta\boldsymbol{x} \cdot \boldsymbol{z}$ (a linear separator). If $\beta > 1$, the trade-off is reversed; as $\beta \to \infty$, only the constant and cubic terms in K_β remain.

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) **(10 pts)** 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?

Solution: In this case, the SVM uses both data points as support vectors such that $y_1 \boldsymbol{w}^T \boldsymbol{x}_1 = 1$ and $y_2 \boldsymbol{w}^T \boldsymbol{x}_2 = 1$. The corresponding $\boldsymbol{w}$

$$\boldsymbol{w}^* = [-1, 2]^T,$$

(b) **(10 pts)** 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.

Solution: In this case, the SVM uses both data points as support vectors such that $y_1 \boldsymbol{w}^T \boldsymbol{x}_1 + b = 1$ and $y_2 \boldsymbol{w}^T \boldsymbol{x}_2 + b = 1$. The corresponding $\boldsymbol{w}$, and b are

$$\boldsymbol{w}^* = [0, 2]^T, b^* = -1$$

The margin for the classifier with offset is larger than the margin for the classifier 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:

- `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

[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. :-)

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) **(0 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) **(0 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.

(c) **(0 pts)** 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) **(10 pts)** Report the dimensionality of the feature matrix in problem 4.1.(c) 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.

[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.

(a) **(0 pts)** 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.
Solution:

(b) **(10 pts)** 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.

Solution: Stratified splits are important, since the fundamental assumption of most ML algorithms is that the training set is a representative sample of the test set i.e., the training and test data are drawn from the same underlying distributions. If the ratio of positive to negative examples (the class balance) differs significantly between the training and test sets (across folds) this assumption will not hold.

(c) **(0 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. 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) **(20 pts)** 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):

C	accuracy	F1-score	AUROC
10^{-3}	0.7089	0.8297	0.8105
10^{-2}	0.7107	0.8306	0.8111
10^{-1}	0.8060	0.8755	0.8576
10^{0}	0.8146	0.8749	0.8712
10^{1}	0.8182	0.8766	0.8696
10^{2}	0.8182	0.8766	0.8696
best C	$10, 100$	$10, 100$	1

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.

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

(a) **(0 pts)** 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. **Solution:**

(b) **(0 pts)** Implement `performance_test(...)` which returns the value of a performance measure, given the test data and a trained classifier. **Solution:**

(c) **(10 pts)** 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.

	C	test performance
accuracy	10	0.7429
F1-score	10	0.4375
AUROC	1	0.7405