CM146, Winter 2019
Problem Set 3: SVM and Kernels
Due March 2,2019, 11:59pm

1 Kernels [8 pts]

(a) **(2 pts)** For any two documents $\boldsymbol{x}$ and $\boldsymbol{z}$, define $K(\boldsymbol{x}, \boldsymbol{z})$ to equal the number of unique words that occur in both $\boldsymbol{x}$ and $\boldsymbol{z}$ (i.e., the size of the intersection of the sets of words in the two documents). Is this function a kernel? Give justification for your answer. **Solution:** We can show that $K(\boldsymbol{x}, \boldsymbol{z})$ is a kernel by explicitly constructing feature vectors $\phi(\boldsymbol{x})$ and $\phi(\boldsymbol{z})$ such that $K(\boldsymbol{x}, \boldsymbol{z}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{z})$. For any set of documents, we can always construct a vocabulary $\mathcal{V}$ with finite size for the words in the document set. Given the vocabulary $\mathcal{V}$, we can construct a feature mapping $\phi(\boldsymbol{x})$ for $\boldsymbol{x}$ by the following: For the k^{th} word w_k in $\mathcal{V}$, if w_k appears in document $\boldsymbol{x}$, assign $\phi(\boldsymbol{x})_k$ (the k^{th} element of $\phi(\boldsymbol{x})$) to be 1; otherwise set the element $\phi(\boldsymbol{x})_k$ to 0. Then the number of unique words common in $\boldsymbol{x}$ and $\boldsymbol{z}$ is $\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{z})$, giving us the kernel.

(b) **(3 pts)** One way to construct kernels is to build them from simpler ones. We have seen various "construction rules", including the following: Assuming $K_1(\boldsymbol{x}, \boldsymbol{z})$ and $K_2(\boldsymbol{x}, \boldsymbol{z})$ are kernels, then so are

- (scaling) $f(\boldsymbol{x})K_1(\boldsymbol{x}, \boldsymbol{z})f(\boldsymbol{z})$ for any function $f(\boldsymbol{x}) \in \mathbb{R}$
- (sum) $K(\boldsymbol{x}, \boldsymbol{z}) = K_1(\boldsymbol{x}, \boldsymbol{z}) + K_2(\boldsymbol{x}, \boldsymbol{z})$
- (product) $K(\boldsymbol{x}, \boldsymbol{z}) = K_1(\boldsymbol{x}, \boldsymbol{z})K_2(\boldsymbol{x}, \boldsymbol{z})$

Using the above rules and the fact that $K(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x} \cdot \boldsymbol{z}$ is (clearly) a kernel, show that the following is also a kernel:

$$\left(1 + \left(\frac{\boldsymbol{x}}{||\boldsymbol{x}||}\right) \cdot \left(\frac{\boldsymbol{z}}{||\boldsymbol{z}||}\right)\right)^3$$

Solution: We can construct the kernel in the following steps.

i. scaling. $K_1(\boldsymbol{x}, \boldsymbol{z}) = \frac{1}{||\boldsymbol{x}||}\frac{1}{||\boldsymbol{z}||}K(\boldsymbol{x}, \boldsymbol{z}) = \left(\frac{\boldsymbol{x}}{||\boldsymbol{x}||}\right) \cdot \left(\frac{\boldsymbol{z}}{||\boldsymbol{z}||}\right)$

ii. sum. $K_2(\boldsymbol{x}, \boldsymbol{z}) = 1 + K_1(\boldsymbol{x}, \boldsymbol{z}) = 1 + \left(\frac{\boldsymbol{x}}{||\boldsymbol{x}||}\right) \cdot \left(\frac{\boldsymbol{z}}{||\boldsymbol{z}||}\right)$

iii. product. $K_3(\boldsymbol{x}, \boldsymbol{z}) = K_2(\boldsymbol{x}, \boldsymbol{z})K_2(\boldsymbol{x}, \boldsymbol{z}) = \left(1 + \left(\frac{\boldsymbol{x}}{||\boldsymbol{x}||}\right) \cdot \left(\frac{\boldsymbol{z}}{||\boldsymbol{z}||}\right)\right)^2$

iv. product. $K_4(\boldsymbol{x}, \boldsymbol{z}) = K_3(\boldsymbol{x}, \boldsymbol{z})K_2(\boldsymbol{x}, \boldsymbol{z}) = \left(1 + \left(\frac{\boldsymbol{x}}{||\boldsymbol{x}||}\right) \cdot \left(\frac{\boldsymbol{z}}{||\boldsymbol{z}||}\right)\right)^3$

For step 2, note that $K(\boldsymbol{x}, \boldsymbol{z}) = 1$ is a valid kernel given by the constant feature mapping $\phi(\boldsymbol{x}) = 1$.

Parts of this assignment are adapted from course material by Tommi Jaakola (MIT), and Andrew Ng (Stanford), and Jenna Wiens (UMich).

(c) **(3 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,

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

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.

2 SVM [8 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 $\frac{1}{2}||\boldsymbol{\theta}||^2$ subject to $y_n \boldsymbol{\theta}^T \boldsymbol{x}_n \geq 1, n = 1, \ldots, N$.

(a) **(2 pts)** Given a single training vector $\boldsymbol{x} = (a, e)^T$ with label $y = -1$, what is the $\boldsymbol{\theta}^*$ that satisfies the above constrained minimization? **Solution:** The SVM with one negative data point orients $\boldsymbol{\theta}$ in the opposite direction of the single data point $\boldsymbol{x}$ in order to minimize the objective function while satisfying the constraint $y_n \boldsymbol{\theta}^T \boldsymbol{x}_n = 1$. The corresponding $\boldsymbol{\theta}$ is

$$\boldsymbol{\theta}^* = -\frac{\boldsymbol{x}}{||\boldsymbol{x}||^2}.$$

(b) **(2 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 $\boldsymbol{\theta}^*$ in this case, and what is the margin γ? **Solution:** In

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

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

$$\boldsymbol{\theta}^* = [-1, 2]^T, \gamma = \frac{1}{\sqrt{5}}.$$

(c) **(4 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 $(\boldsymbol{\theta}^*, b^*)$ and γ? Compare your solutions with and without offset. **Solution:** In this case, the SVM uses both data points as support vectors such that $y_1\boldsymbol{\theta}^T\boldsymbol{x}_1 + b = 1$ and $y_2\boldsymbol{\theta}^T\boldsymbol{x}_2 + b = 1$. The corresponding $\boldsymbol{\theta}$, b, and γ are

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

The margin for the classifier with offset is larger than the margin for the classifier without offset.

2.1 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, '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) **(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.

(c) **(0 pts)** In `main(...)`, we have provided code to read the tweets and labels into a $(630, d)$ feature matrix and $(630,)$ 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.**

2.2 Hyperparameter Selection for a Linear-Kernel SVM [10 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 two different kernels: linear and radial basis function (RBF). We will use the `sklearn.svm.SVC` class and explicitly set only three of the initialization parameters: `kernel`, `gamma`, 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 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.

(a) **(3 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**, **AUROC**, **precision**, **sensitivity**, and **specificity**.

Implement `performance(...)`. All measures, except sensitivity and specificity, are implemented in `sklearn.metrics` library. You can use `sklearn.metrics.confusion_matrix(...)` to calculate the other two. **Solution:**

(b) **(2 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) **(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. Your function should call `cv_performance(...)`, passing in instances of `SVC(kernel='linear', C=c)` with different values for `C`, e.g., $C = 10^{-3}, 10^{-2}, \ldots, 10^{2}$. **Solution:**

(d) **(3 pts)** Finally, using the training data from Section 2.1 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}	0.7089	0.8297	0.8105	0.7089	1	0
10^{-2}	0.7107	0.8306	0.8111	0.7102	1	0.0063
10^{-1}	0.8060	0.8755	0.8576	0.8357	0.9294	0.5081
10^{0}	0.8146	0.8749	0.8712	0.8562	0.9017	0.6045
10^{1}	0.8182	0.8766	0.8696	0.8595	0.9017	0.6167
10^{2}	0.8182	0.8766	0.8696	0.8595	0.9017	0.6167
best C	$10, 100$	$10, 100$	1	$10, 100$	$10^{-3}, 10^{-2}$	$10, 100$

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? **Solution:** Results may vary depending on cross-validation and on scale of search. Optimizing for sensitivity vs specificity should prefer different C's.

2.3 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.

(a) **(2 pts)** Describe the role of the additional hyperparameter γ for an RBF-kernel SVM. How does γ affect generalization error? **Solution:** The γ hyperparameter is related to the "spread" of the RBF kernel (inversely proportional to the variance). As γ increases, the variance decreases and the RBF kernel results in 'tighter' decision boundaries. In other words, as γ increases, the algorithm tries harder to avoid misclassifying training data. Thus, as γ continues to increase the classifier will start to overfit.

(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. **Solution:** Any logarithmic grid is fine. One could use for a coarser grid to search the parameter space and then a finer grid to narrow in on the answer.

(c) **(3 pts)** Finally, using the training data from Section 2.1 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	0.8165	100	10^{-2}
F1-score	0.8763	100	10^{-2}
AUROC	0.8746	100	10^{-2}
precision	0.8583	100	10^{-2}
sensitivity	1	–	–
specificity	0.6047	100	10^{-2}

Note: Multiple settings of C and γ result in a sensitivity of 1.

How does the CV performance vary with the hyperparameters of the RBF-kernel SVM? **Solution:** Results may vary depending on cross-validation and on scale of search.

2.4 Test Set Performance [6 pts]

In this section, you will apply the two classifiers learned in the previous sections to the test data from Section 2.1. Once you have predicted labels for the test data, you will measure performance.

(a) **(2 pts)** Based on the results you obtained in Section 2.2 and Section 2.3, 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 2.1 and `SVC.fit(...)`, train a linear- and an RBF-kernel SVM with your chosen settings. **Solution:**

(b) **(1 pts)** Implement `performance_test(...)` which returns the value of a performance measure, given the test data and a trained classifier. **Solution:**

(c) **(3 pts)** For each performance metric, use `performance_test(...)` 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, and the performance on the test data.

optimizing for	linear-kernel SVM			RBF-kernel SVM			
	C	test performance	95% CI	C	γ	test performance	95% CI
accuracy	10	0.7429	(0.6286 – 0.8429)	100	10^{-2}	0.7571	(0.6429 – 0.8571)
F1-score	10	0.4375	(0.1935 – 0.6316)	100	10^{-2}	0.4516	(0.2000 – 0.6452)
AUROC	1	0.7405	(0.5949 – 0.8750)	100	10^{-2}	0.7328	(0.5880 – 0.8707)
precision	10	0.6364	(0.3333 – 0.9231)	100	10^{-2}	0.7000	(0.3750 – 1.0000)
sensitivity	10^{-3}	1.0000	(1.0000 – 1.0000)	10^{-3}	10^{-5}	1.0000	(1.0000 – 1.0000)
specificity	10	0.9184	(0.8367 – 0.9811)	100	10^{-2}	0.9388	(0.8636 – 1.0000)

How do the test performance of your two classifiers compare? **Solution:** These results were obtained by determining the optimal hyperparameters for each performance metric, training the classifiers using these hyperparameters, then measuring the same performance metric on the test data.

The RBF-kernel SVM should do at least as well as the linear-kernel SVM. Depending on the performance measure chosen to optimize for when selecting hyperparameters, test performance will vary among the results. The 95% CI may vary slightly due to bootstrap samples.