Introduction to Machine Learning
Homework 5: Kernel Methods

Due Tuesday November 26 at 11:59PM

Instructions: Your answers to the questions below, including plots and mathematical work, should be submitted as a single PDF file. It's preferred that you write your answers using software that typesets mathematics (e.g. LaTeX, LyX, or MathJax via iPython), though if you need to you may scan handwritten work. You may find the minted package convenient for including source code in your LaTeX document. If you are using LyX, then the listings package tends to work better. For formatting pseudo-code, you could try algorithm2e.

1 Kernels

(a) For any two documents x and z, define $k(x, z)$ to equal the number of unique words that occur in both x and z (i.e., the size of the intersection of the sets of words in the two documents). Is this function a kernel? Justify your answer. (Hint: $k(x, z)$ is a kernel if there exists $\phi(x)$ such that $k(x, z) = \phi(x)^T \phi(z)$).

(b) Assuming that $\vec{x} = [x_1, x_2], \vec{z} = [z_1, z_2]$ (i.e., both vectors are two-dimensional) and $\beta > 0$, show that the following is a kernel:

$$k_\beta(\vec{x}, \vec{z}) = (1 + \beta \vec{x} \cdot \vec{z})^2 - 1$$

Do so by demonstrating a feature mapping $\Phi(\vec{x})$ such that $k_\beta(\vec{x}, \vec{z}) = \Phi(\vec{x}) \cdot \Phi(\vec{z})$.

(c) One way to construct kernels is to build them from simpler ones. Assuming $k_1(x, z)$ and $k_2(x, z)$ are kernels, then one can show that so are these:

i. (scaling) $f(x)f(z)k_1(x, z)$ for any function $f(x) \in \mathcal{R}$,
ii. (sum) $k(x, z) = k_1(x, z) + k_2(x, z)$,
iii. (product) $k(x, z) = k_1(x, z)k_2(x, z)$.

Using the above rules and the fact that $k(x, z) = x^T z$ is a kernel, show that the following is also a kernel:

$$\left(1 + \left(\frac{x}{||x||_2}\right)^T \left(\frac{z}{||z||_2}\right)\right)^3.$$

2 Kernelized SVM

Recall the SVM objective function

$$\min_{w \in \mathbf{R}^n} \frac{\lambda}{2}\|w\|^2 + \frac{1}{m}\sum_{i=1}^{m} \max\left(0, 1 - y_i w^T x_i\right)$$

and the Pegasos algorithm on the training set $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbf{R}^d \times \{-1, 1\}$ (Algorithm 1).

Algorithm 1: Pegasos Algorithm

input: Training set $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbf{R}^d \times \{-1, 1\}$ and $\lambda > 0$.
$w^{(1)} = (0, \ldots, 0) \in \mathbf{R}^d$
$t = 0$ # step number
repeat
  $t = t + 1$
  $\eta^{(t)} = 1/(t\lambda)$ # step multiplier
  randomly choose j in $1, \ldots, n$
  if $y_j \left\langle w^{(t)}, x_j \right\rangle < 1$
    $w^{(t+1)} = (1 - \eta^{(t)}\lambda) w^{(t)} + \eta^{(t)} y_j x_j$
  else
    $w^{(t+1)} = (1 - \eta^{(t)}\lambda) w^{(t)}$
until bored
return $w^{(t)}$

Note that in every step of Pegasos, we rescale $w^{(t)}$ by $\left(1 - \eta^{(t)}\lambda\right) = \left(1 - \frac{1}{t}\right) \in (0, 1)$. This "shrinks" the entries of $w^{(t)}$ towards 0, and it's due to the regularization term $\frac{\lambda}{2}\|w\|_2^2$ in the SVM objective function. Also note that if the example in a particular step, say (x_j, y_j), is not classified with the required margin (i.e. if we don't have margin $y_j w_t^T x_j \geq 1$), then we also add a multiple of x_j to $w^{(t)}$ to end up with $w^{(t+1)}$. This part of the adjustment comes from the empirical risk. Since we initialize with $w^{(1)} = 0$, we are guaranteed that we can always write

$$w^{(t)} = \sum_{i=1}^{n} \alpha_i^{(t)} x_i$$

after any number of steps t. When we kernelize Pegasos, we'll be tracking $\alpha^{(t)} = (\alpha_1^{(t)}, \ldots, \alpha_n^{(t)})^T$ directly, rather than w.

1. Kernelize the expression for the margin. That is, show that $y_j \left\langle w^{(t)}, x_j \right\rangle = y_j K_{j\cdot} \alpha^{(t)}$, where $k(x_i, x_j) = \langle x_i, x_j \rangle$ and $K_{j\cdot}$ denotes the jth row of the kernel matrix K corresponding to kernel k.

2. Suppose that $w^{(t)} = \sum_{i=1}^{n} \alpha_i^{(t)} x_i$ and for the next step we have selected a point (x_j, y_j) that does not have a margin violation. Give an update expression for $\alpha^{(t+1)}$ so that $w^{(t+1)} =$

$\sum_{i=1}^{n} \alpha_i^{(t+1)} x_i$.

3. Repeat the previous problem, but for the case that (x_j, y_j) has a margin violation. Then give the full pseudocode for kernelized Pegasos. You may assume that you receive the kernel matrix K as input, along with the labels $y_1, \ldots, y_n \in \{-1, 1\}$

3 Image Classification

The MNIST dataset is a database of handwritten digits. This problem will apply SVMs to automatically classify digits; the US postal service uses a similar optical character recognition (OCR) of zip codes to automatically route letters to their destination. The original dataset can be downloaded at http://yann.lecun.com/exdb/mnist/. For this problem, we randomly chose a subset of the original dataset. We have provided you with two data files, `mnist train.txt`, `mnist test.txt`. The training set contains 2000 digits, and the test set contains 1000 digits. Each line represents an image of size 28×28 by a vector of length 784, with each feature specifying a grayscale pixel value. The first column contains the labels of the digits, 0â9, the next 28 columns respesent the first row of the image, and so on. We also provide a scipt to show `show_img.py` to show a single image; using these will help you have a better understanding of what the data looks like and how it is represented.

Your linear classifier will obtain less than 15% test error, and using a Gaussian kernel you will obtain less than 7% test error! Had you used more training data, SVM with Gaussian kernel can get down to 1.4% test error (degree 4 polynomial obtains 1.1% test error). With further fine-tuning (e.g., augmenting the training set by adding deformed versions of the existing training images), a SVM-based approach can obtain 0.56% test error. The state-of-the-art, which uses a convolutional neural network, obtains 0.23% test error.

(a) Read in `mnist_train.txt, mnist_test.txt` and transform them into feature vectors. Normalize the feature vectors so that each feature is in the range [-1, 1]. Since in this dataset each feature has minimum value 0 and maximum value 255, you can do this normalization by transforming each column $\vec{v}$ to $2\vec{v}/255 - 1$. The normalization step can be crucial when you incorporate higher-order features. It also helps prevent numerical difficulties while solving the SVM optimization problem.

(b) Implement multi-class prediction using one-versus-all classification. Train 10 binary classifiers using the Pegasos algorithm from the previous question with the usual linear kernel. For each classifier, you relabel one of the labels to 1, and the other 9 labels to -1. Following learning, you will have 10 distinct weight vectors. To predict the label of an example x, compute the dot product of x with each weight vector, giving you 10 scores, and predict the label with the maximum score.

(c) Instead of holding out a specific portion of your training data as a validation set, there is another approach to estimate the test error: k-fold cross-validation. Cross validation is particularly useful when you have a small amount of training data. Cross validation divides the training data into k parts of equal size. Then, for $i = 1, \ldots, k$ we fit a model using all of

the data except for the kth part, and use the remaining part to compute the validation error. Finally, we report the averaged validation error. Use Scikit-learn's k-fold cross-validation[1] with $k = 5$, and find a model having the smallest cross-validation error from $\lambda = 2^5, 2^4, \ldots, 2^1$. Plot the cross-validation error vs. λ. What is the best λ? For this value of λ, re-train the classifier now using all of the training data. What is the test error?

(d) We now explore the use of non-linear kernels within Support Vector Machines. We will make use of a Scikit-Learn's SVM classifier (SVC)[2] , which provides a nice interface to a powerful library called libSVM. This library implements the SMO algorithm, which is an alternative optimization algorithm for the SVM objective with kernels from what we saw in class. However, SVC by default trains a one-vs-one classifier, so use Scikit-Learn's OneVsRestClassifier [3] to train your classifier. Try the default setting of the SVC, which uses the Gaussian kernel with $\gamma = 1/\text{num\_features}$, and $C = 1$. Make sure that each feature is scaled to $[-1, 1]$ as in problem (b) which could also be done by using Scikit-Learn's MinMaxScaler. Note that in the library, the Gaussian kernel is of the form $K(\vec{u}, \vec{v}) = e^{-\gamma|\vec{u}-\vec{v}|^2}$ (equivalent to RBF when $\gamma = 1/2\sigma^2$) and the optimization problem is of the form

$$\begin{aligned} \min_{\boldsymbol{w},b,\xi} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{j=1}^{m}\xi_j \\ \text{subject to} \quad & y_j(\boldsymbol{w}\cdot\boldsymbol{x}_j + b) \geq 1 - \xi_j, \\ & \xi_j \geq 0. \end{aligned}$$

This can be seen to be equivalent to the SVM optimization problem solved by Pegasos when $C = 1/m\lambda$. Train on the full training set. What is the test error?

(e) Rather than using the default settings, we can choose the two parameters to be tuned (C and γ) using cross-validation. Report the 10-fold cross-validation error when γ and C are at their default settings.

(f) Finally, try different γ and C values to find a model with small cross-validation error. What were the best values that you found? What is the cross-validation error? What is the test error for this setting?

[1] http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.cross_val_score.html#sklearn.cross_validation.cross_val_score

[2] http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[3] http://scikit-learn.org/stable/modules/multiclass.html#one-vs-the-rest