GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 6254 Spring 2018
Problem Set #3

Assigned: 14 Feb 18
Due Date: 26 Feb 18

Suggested reading:

- *Elements of Statistical Learning* (by Hastie, Tibshirani, and Friedman): Section 4.5 (pages 129–135) discusses optimal separating hyperplanes; Sections 12.1–12.3 (pages 417–438) discuss support vector machines and kernels in more detail.

Homework is due in class at the beginning of the class period on the due date.

For **distance learning students**, the homework should be scanned and uploaded to the t-square dropbox **one week after the regular due date**.

PROBLEM 3.1:
Apply the perceptron algorithm to the following pattern classes:

$$\omega_1 : \{(0,0,0)^T, (1,0,0)^T, (1,0,1)^T, (1,1,0)^T\}$$
$$\omega_2 : \{(0,0,1)^T, (0,1,1)^T, (0,1,0)^T, (1,1,1)^T\}.$$

Let $\boldsymbol{\theta}(1) = (-1,-2,-2,0)^T$ where $\boldsymbol{\theta} = (w_1, w_2, w_3, b)^T$.

PROBLEM 3.2:
Consider a support vector machine and the following training data from two categories:

$$\omega_1 : [1,1]^T, [2,2]^T, [2,0]^T$$
$$\omega_2 : [0,0]^T, [1,0]^T, [0,1]^T$$

(a) Plot these six training points, and construct by inspection the weight vector for the optimal hyperplane and the optimal margin.

(b) How many support vectors are there? Identify them.

(c) Construct the solution in dual space by finding the Lagrange undetermined multipliers, α_i. Compare your result to part (a).

PROBLEM 3.3:

In the notes I provided on kernels, we proved that if the function $k(\boldsymbol{u}, \boldsymbol{v})$ is a symmetric and positive semi-definite kernel, then it is an "inner product kernel" (i.e., there is a Hilbert space $\mathcal{H}$ with inner product $\langle \cdot, \cdot \rangle$ and a map $\boldsymbol{\Phi} : \mathbb{R}^d \to \mathcal{H}$ such that $k(\boldsymbol{u}, \boldsymbol{v}) = \langle \boldsymbol{\Phi}(\boldsymbol{u}), \boldsymbol{\Phi}(\boldsymbol{v}) \rangle$). Show that the converse is also true: an inner product kernel must be positive semi-definite. (This direction is much easier.)

PROBLEM 3.4:

1. Show that if $k_1(\boldsymbol{u}, \boldsymbol{v})$ and $k_2(\boldsymbol{u}, \boldsymbol{v})$ are PSD kernels, then

$$k'(\boldsymbol{u}, \boldsymbol{v}) = \gamma_1 k_1(\boldsymbol{u}, \boldsymbol{v}) + \gamma_2 k_2(\boldsymbol{u}, \boldsymbol{v})$$

   is a PSD kernel for $\gamma_1, \gamma_2 \geq 0$.

2. Let $f(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}$ be an arbitrary function on $\mathbb{R}^d$. Show that if $k(\boldsymbol{u}, \boldsymbol{v})$ is a PSD kernel, then

$$k'(\boldsymbol{u}, \boldsymbol{v}) = f(\boldsymbol{u}) k(\boldsymbol{u}, \boldsymbol{v}) f(\boldsymbol{v})$$

   is also a PSD kernel.

3. Show that if $k_1(\boldsymbol{u}, \boldsymbol{v})$ and $k_2(\boldsymbol{u}, \boldsymbol{v})$ are PSD kernels, then

$$k'(\boldsymbol{u}, \boldsymbol{v}) = k_1(\boldsymbol{u}, \boldsymbol{v}) \cdot k_2(\boldsymbol{u}, \boldsymbol{v})$$

   is a PSD kernel. (Hint: any $N \times N$ symmetric positive semi-definite matrix can be written as $\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathrm{T}} = \sum_{n=1}^{N} \lambda_n \boldsymbol{v}_n \boldsymbol{v}_n^{\mathrm{T}}$, and if you did part (b) correctly, you have already shown that if $\boldsymbol{K}$ is a symmetric PSD matrix, then $K'(i,j) = v(i)K(i,j)v(j)$ is also symmetric PSD ...)

4. Suppose that $k_1(\boldsymbol{u}, \boldsymbol{v}), k_2(\boldsymbol{u}, \boldsymbol{v}), \ldots$ is an infinite sequence of PSD kernels that converge pointwise:

$$\lim_{i \to \infty} k_i(\boldsymbol{u}, \boldsymbol{v}) = k(\boldsymbol{u}, \boldsymbol{v}), \quad \text{for all } \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d.$$

   Argue that $k(\boldsymbol{u}, \boldsymbol{v})$ is a PSD kernel.

5. Argue that if $k(\boldsymbol{u}, \boldsymbol{v})$ is a PSD kernel, then

$$k'(\boldsymbol{u}, \boldsymbol{v}) = \exp(k(\boldsymbol{u}, \boldsymbol{v}))$$

   is a PSD kernel. (Hint: Taylor, plus all the properties you proved above.)

6. Show that

$$k(\boldsymbol{u}, \boldsymbol{v}) = \exp\left(-\frac{\|\boldsymbol{u} - \boldsymbol{v}\|_2^2}{2\sigma^2}\right)$$

   is a PSD kernel for all $\sigma^2 > 0$. (Hint: write the k above as $k(\boldsymbol{u}, \boldsymbol{v}) = f(\boldsymbol{u})k'(\boldsymbol{u}, \boldsymbol{v})f(\boldsymbol{v})$.)

PROBLEM 3.5:
In this problem you will use SVMs to build a simple text classification system. To begin, you need to get the MNIST dataset. You can do this in Python using the commands

```
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
```

This downloads the dataset and stores it in a default location (~/scikit_learn_data/). You can also use the optional parameter `data_home=path` in `fetch_mldata` to direct the dataset to a custom path if you wish. The first time you run this command, the data will be downloaded from `mldata.org`, but once it has been downloaded this will simply load the dataset into the variable `mnist`.

This data set contains 70 000 handwritten digits, each of size 28×28 pixels.[1] You should begin by putting this data into a more convenient form by setting

```
X = mnist.data
y = mnist.target
```

Each row of `X` corresponds to one image. You can use the following to plot the $\texttt{j}^{\text{th}}$ image:

```
import matplotlib.pyplot as plt
plt.title('The jth image is a {label}'.format(label=int(y[j])))
plt.imshow(X[j].reshape((28,28)), cmap='gray')
plt.show()
```

In this problem you will build a classifier to classify between the (sometimes very similar) images of the digits "4" and "9". To get just this data, you can use

```
X4 = X[y==4,:]
X9 = X[y==9,:]
```

There are a little under 7 000 examples from each class. You should begin by using this data to form three distinct datasets: the first 4 000 points from each class will be used for designing the classifier (this is the *training set*), and the remainder will be used to evaluate the performance of our classifier (this is the *testing set*).

Since SVMs involve tuning a parameter C, you will also need to set this parameter. We will do this in a principled way using the so-called "holdout method". This means that you take the training set and divide it into two parts: you use the first to fit the classifier, and the second – which we call the *holdout set* – to gauge performance for a given value of C. You will want to decide on a finite set of "grid points" on which to test C (I would suggest a logarithmic grid of values to test both $C \ll 1$ and $C \gg 1$). For each value of C, you will train an SVM on the first part of the set, and then compute the error rate on the holdout set. In the end, you can then choose the value of C that results in a classifier that gives the smallest error on the holdout set

Note: Once you have selected C, you may then retrain on all the entire training set (including the holdout set), but you should **never** use the testing set until every parameter in your algorithm is set. These are used exclusively for computing the final test error.

To train the SVM , you can use the built in solver from scikit-learn. As an example, to train a linear SVM on the full dataset with a value of $C = 1$, we would use

```
from sklearn import svm
clf = svm.SVC(C=1.0,kernel='linear')
clf.fit(X,y)
```

[1] You can learn more about this dataset at `http://yann.lecun.com/exdb/mnist/`.

Once you have trained the classifier, you can calculate the probability of error for the resulting classifier on the full dataset via

```
Pe = 1 - clf.score(X,y)
```

1. Train an SVM using the kernels $k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^{\mathrm{T}}\boldsymbol{v} + 1)^p$, $p = 1, 2$, that is, the inhomogeneous linear and quadratic kernel. To do this, you will want to set the parameters `kernel='poly'` and `degree=1` or `degree=2`.

   For each kernel, report the best value of C, the test error, and the number of data points that are support vectors (returned via `clf.support_vectors_`. Turn in your code.

2. Repeat the above using the radial basis function kernel $k(\boldsymbol{u}, \boldsymbol{v}) = e^{-\gamma\|\boldsymbol{u}-\boldsymbol{v}\|^2}$ (by setting the parameters `kernel='rbf', gamma=gamma`. You will now need to determine the best value for both C and γ. Report the best value of C and γ, the test error, and the number of support vectors.

3. For each kernel, turn in a 4×4 subplot showing images of the 16 support vectors that violate the margin by the greatest amount (these are, in a sense, the "hardest" examples to classify), and explain how these are determined. Above each subplot, indicate the true label (4 or 9).

   To help get started with this, you can use the following code:

   ```
   f, axarr = plt.subplots(4, 4)
   axarr[0, 0].imshow(X[j].reshape((28,28)), cmap='gray')
   axarr[0, 0].axis('off')
   axarr[0, 0].set_title('{label}'.format(label=int(y[j])))
   ...
   plt.show()
   ```

   You should consider using the `clf.decision_function` output of the classifier here.