EECS 545 – Machine Learning - Homework #3

Meng Huang | Due: 11:00pm 02/22/2016

Homework Policy: Working in groups is fine, but each member must submit their own writeup. Please write the members of your group on your solutions. There is no strict limit to the size of the group but we may find it a bit suspicious if there are more than 4 to a team. **For coding problems, please include your code and report your results (values, plots, etc.)** in your PDF submission. You will lose points if your experimental results are only accessible through rerunning your code. Homework will be submitted via Gradescope (https://gradescope.com/).

1) Support Vector Machine (40 points).

Recall that maximizing the soft margin in SVM is equivalent to the following minimization problem:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b) \geq 1-\xi_i \\ & \xi_i \geq 0 \qquad (i=1,\ldots,N) \end{aligned} \tag{1}$$

Equivalently, we can solve the following unconstrained minimization problem:

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\max\left(0, 1-t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b)\right) \tag{2}$$

(a) Prove that minimization problem (1) and (2) are equivalent.

Proof: From (1), we know that we want to minimize the equation $\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i$ subjecting to $t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b) \geq 1-\xi_i$ and $\xi_i \geq 0$.

$$t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b) \geq 1-\xi_i \Leftrightarrow \xi_i \geq 1-t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b)$$

By adding the condition $\xi_i \geq 0$, we get:

$$\xi_i \geq \max\left(0, 1-t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b)\right)$$

So, it is obvious that, for given $\mathbf{w}$ and b, when $\xi_i = \max\left(0, 1-t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b)\right)$ for all i, it will reach the minimum. So,

$$\min_{\mathbf{w},b}\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \Leftrightarrow \min_{\mathbf{w},b}\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\max\left(0, 1-t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}+b)\right)$$

Therefore, equation (1) and (2) are equivalent. □

(b) Let $(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*)$ be the solution of minimization problem (1). Show that if $\xi_i^* > 0$, then the distance from the training data point $\mathbf{x}^{(i)}$ to the margin hyperplane $t^{(i)}((\mathbf{w}^*)^T\mathbf{x} + b^*) = 1$ is proportional to ξ_i^*.

Proof: From the problem (1), we know that $\xi_i = \max\left(0, 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)\right)$, so if $\xi_i > 0$, then $\xi_i = 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)$.

$$\Rightarrow t^{(i)}((\mathbf{w}^*)^T\mathbf{x}^{(i)} + b^*) = 1 - \xi_i^*$$
$$\Rightarrow t^{(i)}((\mathbf{w}^*)^T\mathbf{x} + b^*) - t^{(i)}((\mathbf{w}^*)^T\mathbf{x}^{(i)} + b^*) = \xi_i^*$$
$$\Rightarrow t^{(i)}(\mathbf{w}^*)^T(\mathbf{x} - \mathbf{x}^{(i)}) = \xi_i^*$$
$$\Rightarrow t^{(i)}(\mathbf{w}^*)^T(\mathbf{x} - \mathbf{x}^{(i)})\left(t^{(i)}(\mathbf{w}^*)^T(\mathbf{x} - \mathbf{x}^{(i)})\right)^T = (\xi_i^*)^2$$
$$\Rightarrow t^{(i)}(\mathbf{w}^*)^T(\mathbf{x} - \mathbf{x}^{(i)})(\mathbf{x} - \mathbf{x}^{(i)})^T(\mathbf{w}^*)t^{(i)} = (\xi_i^*)^2$$
$$\Rightarrow t^{(i)}(\mathbf{w}^*)^T\|(\mathbf{x} - \mathbf{x}^{(i)})\|^2(\mathbf{w}^*)t^{(i)} = (\xi_i^*)^2$$

Since $\mathbf{w}^*$ and $t^{(i)}$ are given, the distance between $\mathbf{x}$ and $\mathbf{x}^{(i)}$ is proportional to ξ_i^* for every $\mathbf{x}$ on the margin hyperplane $t^{(i)}((\mathbf{w}^*)^T\mathbf{x} + b^*) = 1$.

So the distance between $\mathbf{x}^{(i)}$ and its projection on the hyperplane $t^{(i)}((\mathbf{w}^*)^T\mathbf{x} + b^*) = 1$ is proportional to ξ_i^*.

In other words, the distance from the training data point $\mathbf{x}^{(i)}$ to the margin hyperplane $t^{(i)}((\mathbf{w}^*)^T\mathbf{x} + b^*) = 1$ is proportional to ξ_i^*.

□

(c) The error function in minimization problem (2) is

$$E(\mathbf{w}, b) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} \max\left(0, 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)\right)$$

Find its derivatives: $\nabla_{\mathbf{w}} E(\mathbf{w}, b)$ and $\frac{\partial}{\partial b}E(\mathbf{w}, b)$. Where the derivative is undefined, use a subderivative.

Answer:

$$\nabla_{\mathbf{w}} E(\mathbf{w}, b) = \mathbf{w} + C\sum_{i=1}^{N} \nabla_{\mathbf{w}} \max\left(0, 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)\right)$$

$$= \mathbf{w} + C\sum_{i=1}^{N} \nabla_{\mathbf{w}} \left[\mathbf{I}\left(1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) > 0\right)(1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b))\right]$$

$$= \mathbf{w} - C\sum_{i=1}^{N} \left[\mathbf{I}\left(t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) < 1\right) t^{(i)}\mathbf{x}^{(i)}\right]$$

$$\frac{\partial}{\partial b}E(\mathbf{w}, b) = C\sum_{i=1}^{N} \frac{\partial}{\partial b} \max\left(0, 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)\right)$$

$$= C\sum_{i=1}^{N} \frac{\partial}{\partial b} \left[\mathbf{I}\left(1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) > 0\right)(1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b))\right]$$

$$= -C\sum_{i=1}^{N} \left[\mathbf{I}\left(t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) < 1\right) t^{(i)}\right]$$

Note: in here, $\mathbf{I}()$ is the indicator function

(d) Implement the soft-margin SVM using batch gradient descent. Here is the pseudo code:

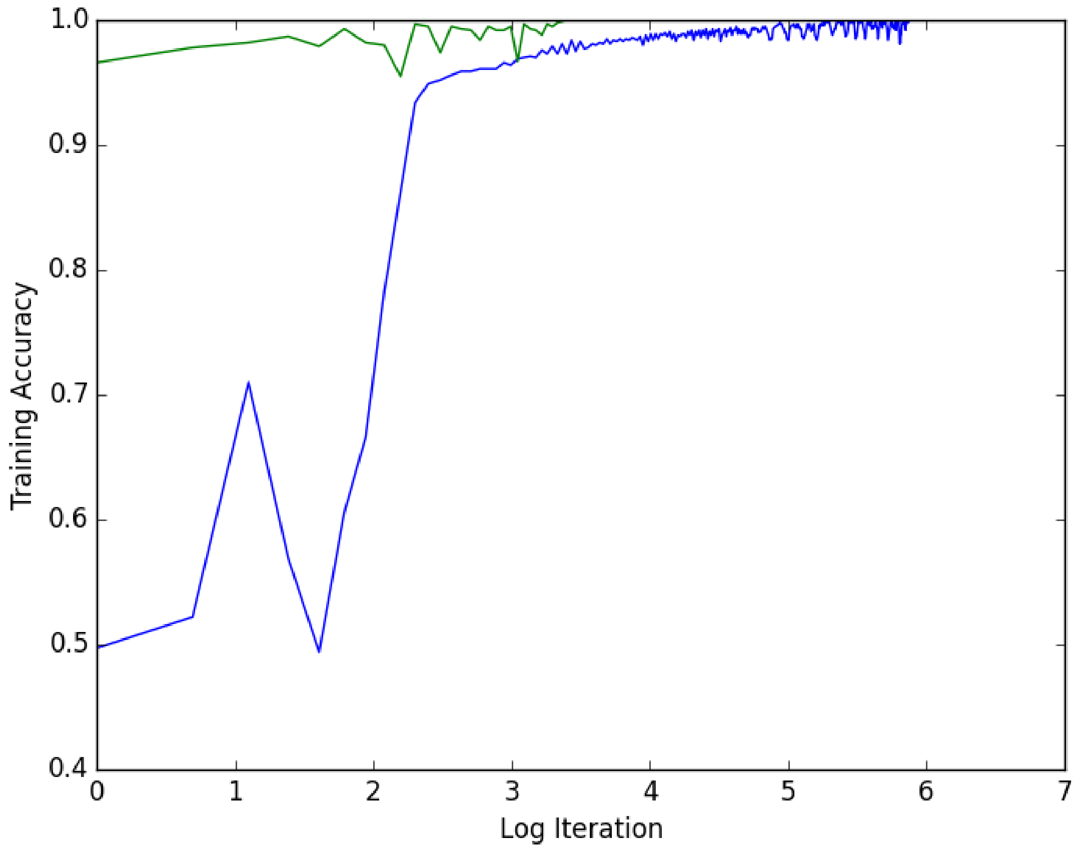
```
$\mathbf{w}^* \leftarrow \mathbf{0}$ ;
$b^* \leftarrow 0$ ;
for $j$=1 to NumIterations do
    $\mathbf{w}_{grad} \leftarrow \nabla_{\mathbf{w}} E(\mathbf{w}^*, b^*)$ ;
    $b_{grad} \leftarrow \frac{\partial}{\partial b} E(\mathbf{w}^*, b^*)$ ;
    $\mathbf{w}^* \leftarrow \mathbf{w}^* - \alpha(j)\ \mathbf{w}_{grad}$ ;
    $b^* \leftarrow b^* - \alpha(j)\ b_{grad}$ ;
end
return $\mathbf{w}^*$
```

Algorithm 1: SVM Batch Gradient Descent

The learning rate for the j-th iteration is defined as:

$$\alpha(j) = \frac{\eta_0}{1 + j \cdot \eta_0}$$

Set η_0 to 0.001 and the slack cost C to 3. Show the iteration-versus-accuracy (training accuracy) plot. The training and test data/labels are provided in `digits_training_data.csv`, `digits_training_labels.csv`, `digits_test_data.csv` and `digits_test_labels.csv`.



Note: the value of x-axis is log(j) which j is the iterator. The blue line is for Batch Gradient Descent, and the green line is for Stochastic Gradient Descent.

(e) Let

$$E^{(i)}(\mathbf{w}, b) = \frac{1}{2N}\|\mathbf{w}\|^2 + C \max\left(0, 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)\right)$$

then

$$E(\mathbf{w}, b) = \sum_{i=1}^{N} E^{(i)}(\mathbf{w}, b)$$

Find the derivatives $\nabla_{\mathbf{w}} E^{(i)}(\mathbf{w}, b)$ and $\frac{\partial}{\partial b} E^{(i)}(\mathbf{w}, b)$. Once again, use a subderivative if the derivative is undefined.

Answer:

$$\nabla_{\mathbf{w}} E^{(i)}(\mathbf{w}, b) = \frac{1}{N}\mathbf{w} + C\nabla_{\mathbf{w}} \max\left(0, 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)\right)$$
$$= \frac{1}{N}\mathbf{w} + C\nabla_{\mathbf{w}}\mathbf{I}\left(1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) > 0\right)(1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b))$$
$$= \frac{1}{N}\mathbf{w} - C\mathbf{I}\left(t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) < 1\right)t^{(i)}\mathbf{x}^{(i)}$$

$$\frac{\partial}{\partial b} E^{(i)}(\mathbf{w}, b) = C\frac{\partial}{\partial b} \max\left(0, 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)\right)$$
$$= C\frac{\partial}{\partial b}\mathbf{I}\left(1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) > 0\right)(1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b))$$
$$= -C\mathbf{I}\left(t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) < 1\right)t^{(i)}$$

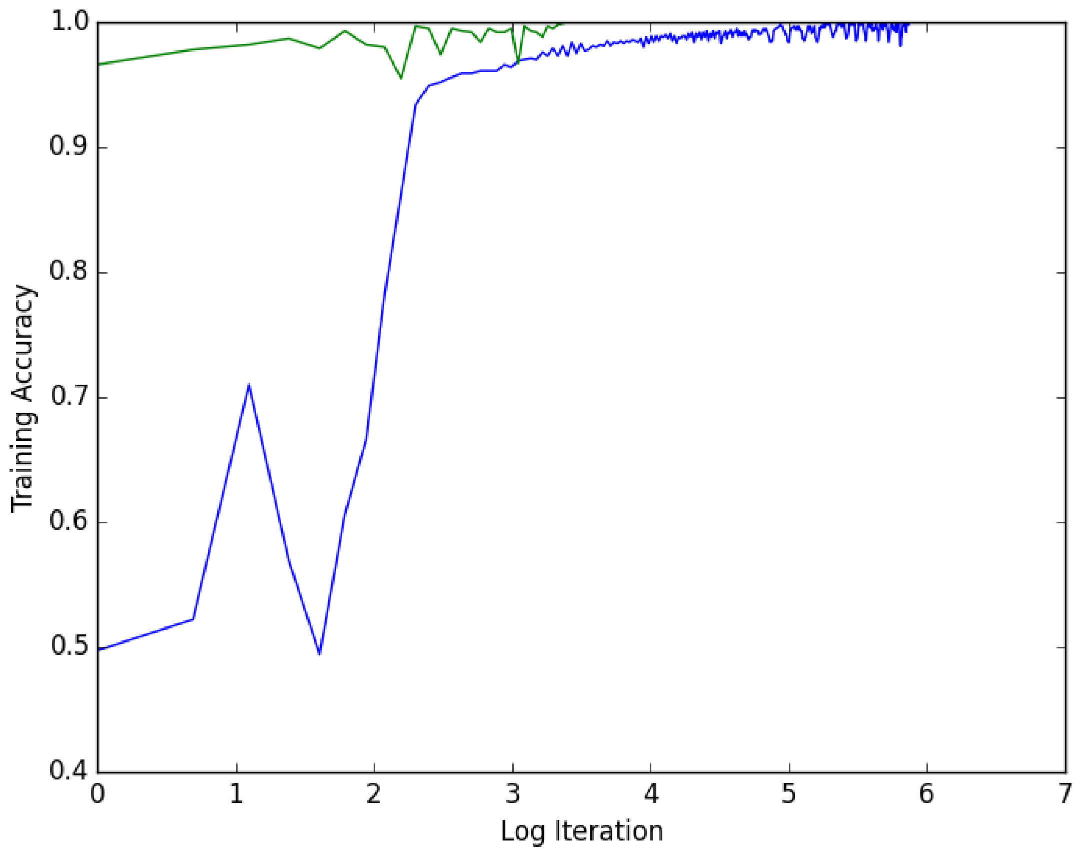
Note: in here, $\mathbf{I}()$ is the indicator function

(f) Implement the soft-margin SVM using stochastic gradient descent. Here is the pseudo-code:

```
$\mathbf{w}^* \leftarrow \mathbf{0}$ ;
$b^* \leftarrow 0$ ;
**for** *j=1 to NumIterations* **do**
    **for** *i = Random Permutation of 1 to N* **do**
        $\mathbf{w}_{grad} \leftarrow \nabla_{\mathbf{w}} E^{(i)}(\mathbf{w}^*, b^*)$ ;
        $b_{grad} \leftarrow \frac{\partial}{\partial b} E^{(i)}(\mathbf{w}^*, b^*)$ ;
        $\mathbf{w}^* \leftarrow \mathbf{w}^* - \alpha(j)\ \mathbf{w}_{grad}$ ;
        $b^* \leftarrow b^* - \alpha(j)\ b_{grad}$ ;
    **end**
**end**
**return** $\mathbf{w}^*$
```

Algorithm 2: SVM Stochastic Gradient Descent

Use the same $\alpha(\cdot)$, η_0 and C in (c). Be sure to use a new random permutation of the indices of the inner loop for each iteration of the outer loop. Show the iteration-versus-accuracy (outer iteration and training accuracy) curve **in the same plot** as that for batch gradient descent. The training and test data/labels are provided in `digits_training_data.csv`, `digits_training_labels.csv`, `digits_test_data.csv` and `digits_test_labels.csv`.



Note: the value of x-axis is log(j) which j is the iterator. The blue line is for Batch Gradient Descent, and the green line is for Stochastic Gradient Descent.

(g) What can you conclude about the convergence rate of stochastic gradient descent versus batch gradient descent? How did you make this conclusion?

Answer: It is obvious that the stochastic gradient descent converges much faster than batch gradient descent, since the number of iteration for stochastic gradient descent to reach 100% training accuracy is around 30 while the one for batch gradient descent is around 350.

(h) Show the Lagrangian function for minimization problem (1) and derive the dual problem. Your result should have only dual variables in the objective function as well as the constraints. How can you kernelize the soft-margin SVM based on this dual problem?

Answer: The Lagrangian function for minimization problem (1) is:

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N} a_i\{t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) - 1 + \xi_i\} - \sum_{i=1}^{N}\mu_i\xi_i$$

where $a_i \geq 0, \mu_i \geq 0, \xi_i \geq 0$ for all i
And the KKT conditions for the constraints are:

$$a_i \geq 0$$
$$t_i(\mathbf{w}^T\mathbf{x}^{(i)} + b) - 1 + \xi_i \geq 0$$
$$a_i\left(t_i(\mathbf{w}^T\mathbf{x}^{(i)} + b) - 1 + \xi_i\right) = 0$$
$$\mu_i \geq 0$$
$$\xi_i \geq 0$$
$$\mu_i\xi_i = 0$$

Deriving the dual problem:

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial}{\partial \mathbf{w}}\left[\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N} a_i\{t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) - 1 + \xi_i\} - \sum_{i=1}^{N}\mu_i\xi_i\right] = \mathbf{w} - \sum_{i=1}^{N} a_i t^{(i)}\mathbf{x}^{(i)}$$

$$\frac{\partial L}{\partial b} = \frac{\partial}{\partial b}\left[\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N} a_i\{t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) - 1 + \xi_i\} - \sum_{i=1}^{N}\mu_i\xi_i\right] = \sum_{i=1}^{N} a_i t^{(i)}$$

$$\frac{\partial L}{\partial \xi_i} = \frac{\partial}{\partial \xi_i}\left[\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N} a_i\{t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) - 1 + \xi_i\} - \sum_{i=1}^{N}\mu_i\xi_i\right] = C - a_i - \mu_i$$

So if $\frac{\partial L}{\partial \mathbf{w}} = 0$, $\frac{\partial L}{\partial b} = 0$, $\frac{\partial L}{\partial \xi_i} = 0$, then

$$\mathbf{w} = \sum_{i=1}^{N} a_i t^{(i)}\mathbf{x}^{(i)}$$

$$\sum_{i=1}^{N} a_i t_i = 0$$
$$a_i = C - \mu_i$$

Finally, we need to combine these equations with $L(\mathbf{w}, b, \mathbf{a})$

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N} a_i\{t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) - 1 + \xi_i\} - \sum_{i=1}^{N}\mu_i\xi_i$$

$$= \frac{1}{2}\left(\sum_{i=1}^{N} a_i t^{(i)} \mathbf{x}^{(i)}\right)^T \left(\sum_{j=1}^{N} a_j t^{(j)} \mathbf{x}^{(j)}\right) - \sum_{i=1}^{N}\left[a_i t^{(i)} \left(\sum_{j=1}^{N} a_j t^{(j)} \mathbf{x}^{(j)}\right)^T \mathbf{x}^{(i)} + a_i t^{(i)} b - a_i\right] + \sum_{i=1}^{N}(C - a_i - \mu_i)\xi_i$$

$$= \sum_{i=1}^{N} a_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j t^{(i)} t^{(j)} k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

Subject to

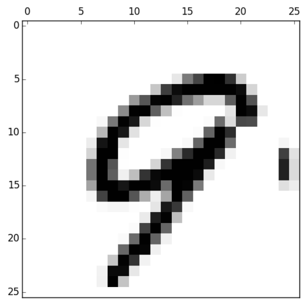
$$0 \le a_i \le C$$

$$\sum_{i=1}^{N} a_i t^{(i)} = 0$$

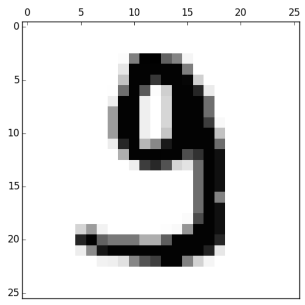
which $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$ (in problem (1), $\phi(\mathbf{x}) = \mathbf{x}$)

(i) Apply the soft-margin SVM (with RBF kernel) to handwritten digit classification. The training and test data/labels are provided in `digits_training_data.csv`, `digits_training_labels.csv`, `digits_test_data.csv` and `digits_test_labels.csv`. Report the training and test accuracy, and show 5 of the misclassified test images (if fewer than 5, show all; label them with your predictions). You can use the scikit-learn (or equivalent) implementation of the kernelized SVM in this question. You are free to select the parameters (for RBF kernel and regularization), and please report your parameters.

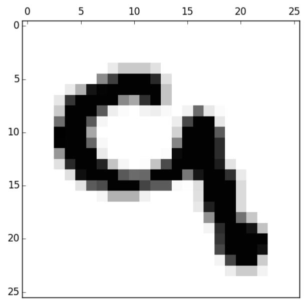
Answer: the training accuracy is 99.5 %, and the test accuracy is 95.4%. The followings are 5 misclassified images:
1. This image is predicted as "4".



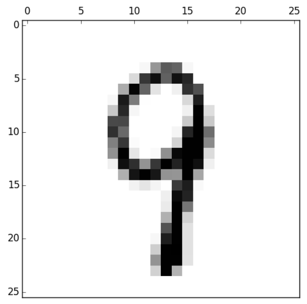
2. This image is predicted as "4".



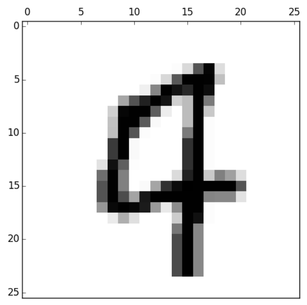
3. This image is predicted as "4".



4. This image is predicted as "4".



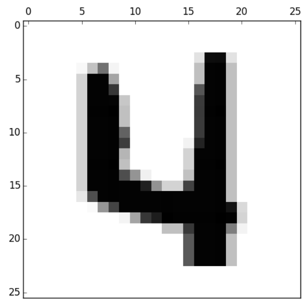
5. This image is predicted as "9".



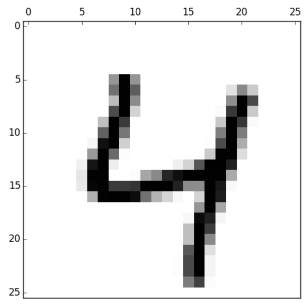
(j) Implement linear discriminant analysis (LDA) using the same data in (i). Report the training and test accuracy, and show 5 of the misclassified test images (if fewer than 5, show all; label them with your predictions). Is there any significant difference between LDA and SVM (with RBF kernel)? Using implementation from libraries is NOT allowed in this question. You can use pseudoinverse during computation.

Answer: the training accuracy is 99.7 %, and the test accuracy is 55.8%. The followings are 5 misclassified images:

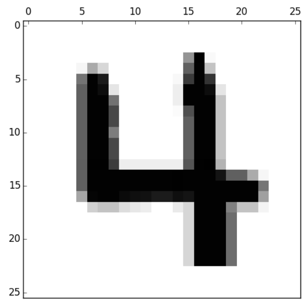
1. This image is predicted as ”9”.



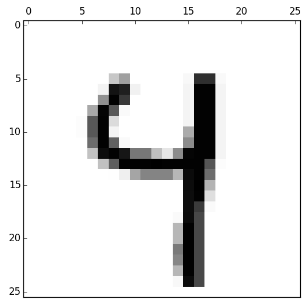
2. This image is predicted as ”9”.



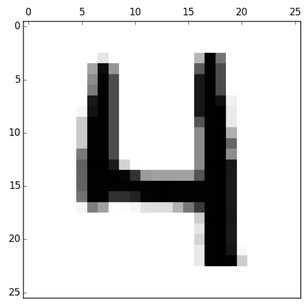
3. This image is predicted as ”9”.



4. This image is predicted as ”9”.



5. This image is predicted as ”9”.



It seems that LDA usually misclassifies some 4’s, which are open on the top, as 9, while the SVM with RBF usually misclassifies some 9’s, which are close on the top, as 4.

Code for problem 1:

```
import numpy as np
from numpy.linalg import pinv
from matplotlib import pyplot as plt
import matplotlib as mpl
from sklearn import preprocessing
from sklearn import svm
import math, csv

NumIteration = 1000
ita = 0.01
C = 3
a = lambda j : float(ita) / float(1.0 + j * ita)

def readFile(filename):
  data = np.genfromtxt('./%s' % filename, delimiter=',')
  return data

def getAccuracy(w, b, data, labels):
  data = data.T
  M, N = data.shape

  correctness = 0
  for i in xrange(0, N):
    x = data[:, i]
    t = labels[i]

    if (np.dot(w.T, x) + b) > 0:
      if t == 1:
        correctness += 1
    else:
      if t == -1:
        correctness += 1

  return float(correctness) / float(N)

def show_image(image, fig, show=False):
    """
    Render a given numpy.uint8 2D array of pixel data.
    """
    ax = fig.add_subplot(1,1,1)
    imgplot = ax.imshow(image, cmap=mpl.cm.Greys)
    imgplot.set_interpolation('nearest')
    ax.xaxis.set_ticks_position('top')
    ax.yaxis.set_ticks_position('left')
    if show:
      plt.show()
```

```
# Batch Gradient Descent
def trainBatchDescent(data, labels):
  data = data.T
  M, N = data.shape
  w = np.zeros(M)
  b = 0.0
  acc_list = []
  for j in xrange(NumIteration):
    w_grad = w
    b_grad = 0.0
    for i in xrange(N):
      x = data[:, i]
      t = labels[i]

      if (t * (np.dot(w.T, x) + b)) < 1:
        w_grad = w_grad - C * t * x
        b_grad = b_grad - C * t

    w = w - a(j+1) * w_grad
    b = b - a(j+1) * b_grad

    correctness = 0
    for i in xrange(0, N):
      x = data[:, i]
      t = labels[i]

      if (np.dot(w.T, x) + b) > 0:
        if t == 1:
          correctness += 1
      else:
        if t == -1:
          correctness += 1
    acc_list.append(float(correctness) / float(N))


  return w, b, acc_list

def Batch():
  print "Doing Batch Descent..."
  training_data = readFile('digits_training_data.csv')
  training_labels = readFile('digits_training_labels.csv')
  test_data = readFile('digits_test_data.csv')
  test_labels = readFile('digits_test_labels.csv')
  training_data_std = preprocessing.scale(training_data)
  test_data_std = preprocessing.scale(test_data)

  for i in xrange(0, training_labels.shape[0]):
    if training_labels[i] == 4:
      training_labels[i] = -1
    if training_labels[i] == 9:
      training_labels[i] = 1

  for i in xrange(0, test_labels.shape[0]):
    if test_labels[i] == 4:
      test_labels[i] = -1
    if test_labels[i] == 9:
      test_labels[i] = 1

  w, b, acc_list = trainBatchDescent(training_data, training_labels)
  acc = getAccuracy(w, b, training_data, training_labels)
  print "training accuracy: %f" % acc
  print "Finished Batch Descent"
  return acc_list
```

```
# Stochastic Gradient Descent
def trainStochasticDescent(data, labels):
  data = data.T
  M, N = data.shape
  w = np.zeros(M)
  b = 0
  acc_list = []

  for j in xrange(NumIteration):
    for i in np.random.permutation(N):
      x = data[:, i]
      t = labels[i]

      w_grad = 1.0 / float(N) * w
      b_grad = 0.0
      if (t * (np.dot(w.T, x) + b)) < 1:
        w_grad -= C * t * x
        b_grad -= C * t

      w = w - a(j+1) * w_grad
      b = b - a(j+1) * b_grad

    correctness = 0
    for i in xrange(0, N):
      x = data[:, i]
      t = labels[i]

      if (np.dot(w.T, x) + b) > 0:
        if t == 1:
          correctness += 1
      else:
        if t == -1:
          correctness += 1
    acc_list.append(float(correctness) / float(N))

  return w, b, acc_list

def Stochastic():
  print "Doing Stochastic Descent..."
  training_data = readFile('digits_training_data.csv')
  training_labels = readFile('digits_training_labels.csv')
  test_data = readFile('digits_test_data.csv')
  test_labels = readFile('digits_test_labels.csv')

  training_data_std = preprocessing.scale(training_data)
  test_data_std = preprocessing.scale(test_data)

  for i in xrange(0, training_labels.shape[0]):
    if training_labels[i] == 4:
      training_labels[i] = -1
    if training_labels[i] == 9:
      training_labels[i] = 1

  for i in xrange(0, test_labels.shape[0]):
    if test_labels[i] == 4:
      test_labels[i] = -1
    if test_labels[i] == 9:
      test_labels[i] = 1

  w, b, acc_list = trainStochasticDescent(training_data, training_labels)
  acc = getAccuracy(w, b, training_data, training_labels)
  print "training accuracy: %f" % acc
  print "Finished Stochastic Descent"
  return acc_list
```

```
# RBF Kernel for SVM
def SVM_RBF_Kernel():
  print "Doing RBF Kernel..."
  training_data = readFile('digits_training_data.csv')
  training_labels = readFile('digits_training_labels.csv')
  test_data = readFile('digits_test_data.csv')
  test_labels = readFile('digits_test_labels.csv')
  for i in xrange(0, training_labels.shape[0]):
    if training_labels[i] == 4:
      training_labels[i] = -1
    if training_labels[i] == 9:
      training_labels[i] = 1

  for i in xrange(0, test_labels.shape[0]):
    if test_labels[i] == 4:
      test_labels[i] = -1
    if test_labels[i] == 9:
      test_labels[i] = 1

  training_data_std = preprocessing.scale(training_data)
  test_data_std = preprocessing.scale(test_data)

  img_list = []

  rbf_svc = svm.SVC(kernel='rbf', C=1)
  rbf_svc.fit(training_data_std, training_labels)

  predictions = rbf_svc.predict(training_data_std)
  correctness = sum(((predictions + training_labels) / 2) ** 2)
  acc = float(correctness) / float(training_labels.size)
  print "training accuracy: %f" % acc

  predictions = rbf_svc.predict(test_data_std)
  correctness = sum(((predictions + test_labels) / 2) ** 2)
  acc = float(correctness) / float(test_labels.size)
  print "test accuracy: %f" % acc

  imgHeight = np.sqrt(test_data.shape[1])
  n = 0
  for i in xrange(predictions.size):
    if predictions[i] != test_labels[i]:
      if predictions[i] == -1:
        img_list.append((test_data[i].reshape((imgHeight, imgHeight)), 4))
      else:
        img_list.append((test_data[i].reshape((imgHeight, imgHeight)), 9))
      n += 1
    if n == 5:
      break


  print "Finished RBF Kernel"
  return img_list







# Linear Discriminant Analysis (LDA)
def trainLDA(data, labels):
  N4 = 0.0
  x4 = np.zeros(data.shape[1])
  N9 = 0.0
  x9 = np.zeros(data.shape[1])
  for i in xrange(labels.size):
    x = data[i, :]
    t = labels[i]

    if t == 1:
      N9 += 1.0
      x9 = x9 + x

    else:
      N4 += 1.0
      x4 = x4 + x

  mu4 = np.matrix(x4 / N4).T
  mu9 = np.matrix(x9 / N9).T

  sigma = np.matrix(np.zeros((data.shape[1], data.shape[1])))
  for i in xrange(labels.size):
    x = np.matrix(data[i, :]).T
    t = labels[i]

    if t == -1:
      sigma = sigma + (x - mu4) * (x - mu4).T
    else:
      sigma = sigma + (x - mu9) * (x - mu9).T
  sigma = sigma / (N4 + N9)

  return N9 / (N4 + N9), mu4, mu9, sigma



def predictLDA(test_data, model):
  phi9, mu4, mu9, sigma = model
  sigma_inv = pinv(sigma)

  predictions = []
  for i in xrange(test_data.shape[0]):
    x = np.matrix(test_data[i, :]).T
    P4 = -1 / 2 * (x - mu4).T * sigma_inv * (x - mu4) + np.log(1.0 - phi9)
    P9 = -1 / 2 * (x - mu9).T * sigma_inv * (x - mu9) + np.log(phi9)
    if P4 < P9:
      predictions.append(1)
    else:
      predictions.append(-1)

  return np.array(predictions)



def LDA():
  print "Doing LDA..."
  training_data = readFile('digits_training_data.csv')
  training_labels = readFile('digits_training_labels.csv')
  test_data = readFile('digits_test_data.csv')
  test_labels = readFile('digits_test_labels.csv')
  for i in xrange(0, training_labels.shape[0]):
    if training_labels[i] == 4:
      training_labels[i] = -1
    if training_labels[i] == 9:
      training_labels[i] = 1

  for i in xrange(0, test_labels.shape[0]):
    if test_labels[i] == 4:
      test_labels[i] = -1
    if test_labels[i] == 9:
      test_labels[i] = 1

  training_data_std = preprocessing.scale(training_data)
  test_data_std = preprocessing.scale(test_data)

  img_list = []

  model = trainLDA(training_data_std, training_labels)

  predictions = predictLDA(training_data_std, model)
  correctness = sum(((predictions + training_labels) / 2) ** 2)
  acc = float(correctness) / float(training_labels.size)
  print "training accuracy: %f" % acc

  predictions = predictLDA(test_data_std, model)
  correctness = sum(((predictions + test_labels) / 2) ** 2)
  acc = float(correctness) / float(test_labels.size)
  print "test accuracy: %f" % acc
  imgHeight = np.sqrt(test_data.shape[1])
  n = 0
  for i in xrange(predictions.size):
    if predictions[i] != test_labels[i]:
      if predictions[i] == -1:
        img_list.append((test_data[i].reshape((imgHeight, imgHeight)), 4))
      else:
        img_list.append((test_data[i].reshape((imgHeight, imgHeight)), 9))
      n += 1
    if n == 5:
      break


  print "Finished LDA"
  return img_list




def main():
  batch_acc_list = Batch()
  stoch_acc_list = Stochastic()

  fig = plt.figure()
  plt.plot(np.log(range(NumIteration)[1:]), batch_acc_list[1:], \
           np.log(range(NumIteration)[1:]), stoch_acc_list[1:])
  plt.xlabel('Log Iteration')
  plt.ylabel('Training Accuracy')
  fig.savefig('1.png')

  img_list = SVM_RBF_Kernel()
  n = 1
  for img, pred in img_list:
    fig = plt.figure()
    show_image(img, fig)
    fig.savefig('mis_img_%d_label_%d.png' % (n, pred))
    n += 1

  img_list = LDA()
  n = 6
  for img, pred in img_list:
    fig = plt.figure()
    show_image(img, fig)
    fig.savefig('mis_img_%d_label_%d.png' % (n, pred))
    n += 1

if __name__ == '__main__':
  main()
```

2) **Open Kaggle Challenge (20 points).** You can use any algorithm and design any features to perform classification on the handwritten digit dataset. Please refer to `https://inclass.kaggle.com/c/handwritten-digit-classification` for details. This problem will be graded separately based on your performance on the private leaderboard.

```
Code for problem 2:

from load_data import *
from sklearn.multiclass import OutputCodeClassifier
from sklearn import svm
from sklearn.linear_model import LogisticRegressionCV
from sklearn import preprocessing

def main():
  print "Fetching Data..."
  train_data, train_labels, test_data = load_data()
  train_data_std = preprocessing.scale(train_data)
  test_data_std = preprocessing.scale(test_data)

  print "Start Training..."
  svc = svm.SVC(decision_function_shape='ovo')
  trained_model = svc.fit(train_data_std, train_labels)
  # print "Predicting Train Data..."
  # predictions = trained_model.predict(train_data_std)

  # print "Calculating Train Accuracy..."
  # correctness = 0.0
  # for c in (train_labels - predictions):
  #   if c == 0:
  #     correctness += 1.0

  # print float(correctness) / float(train_labels.size)

  print "Predicting Test Data..."
  predictions = trained_model.predict(test_data_std)
  print "Outputing..."
  output_matrix = []
  for (i, t) in zip(range(1, len(predictions)+1), predictions):
    output_matrix.append([i, t])
  output_matrix = np.mat(output_matrix)
  np.savetxt('final_result.csv', output_matrix, fmt=('%i', '%i'), delimiter=',',\
             header='id,category', comments='')


if __name__ == "__main__":
  main()
```

3) **Constructing Kernels (20 points).**

(a) Let $\mathbf{u}, \mathbf{v}$ be vectors of dimension d. What feature map ϕ does the kernel

$$k(\mathbf{u}, \mathbf{v}) = (\langle \mathbf{u}, \mathbf{v} \rangle + 1)^4$$

correspond to? In other words, specify the function $\phi(\cdot)$ so that $k(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u})^\top \phi(\mathbf{v})$ for all $\mathbf{u}, \mathbf{v}$. Please show the expression for $d = 3$ and describe how to extend it to arbitrary dimension d.

Proof: Let $\mathbf{u} = (u_1, u_2, u_3)^T$ and $\mathbf{v} = (v_1, v_2, v_3)^T$, and $u_4 = v_4 = 1$
then

$$\begin{aligned} k(\mathbf{u}, \mathbf{v}) &= (\langle \mathbf{u}, \mathbf{v} \rangle + 1)^4 \\ &= (u_1 v_1 + u_2 v_2 + u_3 v_3 + u_4 v_4)^4 \\ &= \left(\sum_{i=1}^{4} u_i v_i \right)^4 \\ &= \sum_{i_1, i_2, i_3, i_4 \in \{1,2,3,4\}} u_{i_1} u_{i_2} u_{i_3} u_{i_4} v_{i_1} v_{i_2} v_{i_3} v_{i_4} \end{aligned}$$

So, if letting

$$\phi(\mathbf{u}) = \begin{pmatrix} u_1 u_1 u_1 u_1 \\ u_1 u_1 u_1 u_2 \\ u_1 u_1 u_1 u_3 \\ . \\ . \\ . \\ u_4 u_4 u_4 u_4 \end{pmatrix}$$

then, we can get:

$$k(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u})^T \phi(\mathbf{v})$$

To extending to arbitrary dimension d, we can just generally let:

$$\phi(\mathbf{u}) = \begin{pmatrix} u_1 u_1 u_1 u_1 \\ u_1 u_1 u_1 u_2 \\ u_1 u_1 u_1 u_3 \\ . \\ . \\ . \\ u_{d+1} u_{d+1} u_{d+1} u_{d+1} \end{pmatrix}$$

which the entries of this vector have the form $u_{i_1} u_{i_2} u_{i_3} u_{i_4}$ with $i_1, i_2, i_3 . i_4 \in \{1, 2, ..., d, d+1\}$
then

$$k(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u})^T \phi(\mathbf{v})$$

$\square$

(b) Let k_1, k_2 be positive-definite kernel functions over $\mathbb{R}^D \times \mathbb{R}^D$, let $a \in \mathbb{R}^+$ be a positive real number, let $f : \mathbb{R}^D \to \mathbb{R}$ be a real-valued function and let $p : \mathbb{R} \to \mathbb{R}$ be a polynomial with *positive* coefficients. For each of the functions k below, state whether it is necessarily a positive-definite kernel. If you think it is, prove it; if you think it is not, give a counterexample.

(i) $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) + k_2(\mathbf{x}, \mathbf{z})$

Answer: True
Proof: Suppose $k_1(\mathbf{x}, \mathbf{z}) = \phi_1(\mathbf{x})^T\phi_1(\mathbf{z})$ and $k_2(\mathbf{x}, \mathbf{z}) = \phi_2(\mathbf{x})^T\phi_2(\mathbf{z})$, then let $\phi(\mathbf{x}) = (\phi_1(\mathbf{x})^T, \phi_2(\mathbf{x})^T)^T$ ($\phi_1(\mathbf{x})$ vertically concatenates with $\phi_2(\mathbf{x})$), and it is obvious that $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T\phi(\mathbf{z})$

(ii) $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) - k_2(\mathbf{x}, \mathbf{z})$

Answer: False
Proof: $k(\mathbf{x}, \mathbf{x})$ should be greater than or equal to 0. But if $k_1(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T\mathbf{z}$ and $k_2(\mathbf{x}, \mathbf{z}) = (2\mathbf{x})^T(2\mathbf{z})$, which both of them are apparently a positive-definite kernel function, then $k(\mathbf{x}, \mathbf{x}) = -3\mathbf{x}^T\mathbf{x} \leq 0$ is not a positive-definite kernel function.

(iii) $k(\mathbf{x}, \mathbf{z}) = ak_1(\mathbf{x}, \mathbf{z})$

Answer: True
Proof: Suppose $k_1(\mathbf{x}, \mathbf{z}) = \phi_1(\mathbf{x})^T\phi_1(\mathbf{z})$, then let $\phi(\mathbf{x}) = \sqrt{a}\phi_1(\mathbf{x})$, and it is obvious that $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T\phi(\mathbf{z})$

(iv) $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z})k_2(\mathbf{x}, \mathbf{z})$

Answer: True
Proof: Suppose $k_1(\mathbf{x}, \mathbf{z}) = \phi_1(\mathbf{x})^T\phi_1(\mathbf{z})$ and $k_2(\mathbf{x}, \mathbf{z}) = \phi_2(\mathbf{x})^T\phi_2(\mathbf{z})$
then let

$$\phi(\mathbf{x}) = \begin{pmatrix} (\phi_1(\mathbf{x})_1)(\phi_2(\mathbf{x})_1) \\ (\phi_1(\mathbf{x})_1)(\phi_2(\mathbf{x})_2) \\ (\phi_1(\mathbf{x})_1)(\phi_2(\mathbf{x})_3) \\ . \\ . \\ . \\ (\phi_1(\mathbf{x})_m)(\phi_2(\mathbf{x})_n) \end{pmatrix}$$

which $\phi_i(\mathbf{x})_j$ means the j-th entry of vector $\phi_i(\mathbf{x})$, and m, n are the dimensions of $\phi_1(\mathbf{x})$ and $\phi_2(\mathbf{x})$, respectively.
and $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T\phi(\mathbf{z}) = \sum_{i=1}^{m}\sum_{j=1}^{n}(\phi_1(\mathbf{x})_i)(\phi_2(\mathbf{x})_j)(\phi_1(\mathbf{z})_i)(\phi_2(\mathbf{z})_j) = k_1(\mathbf{x}, \mathbf{z})k_2(\mathbf{x}, \mathbf{z})$

(v) $k(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$

Answer: False
Proof: if $f(\mathbf{x}) = 0$ for all $\mathbf{x}$, then it is a PSD kernel, but not a PD kernel

(vi) $k(\mathbf{x}, \mathbf{z}) = p(k_1(\mathbf{x}, \mathbf{z}))$

Answer: True
Proof: suppose $p(x) = \sum_{i=0}^{n} a_i x^i$ with $a_0, a_1, ..., a_n$ are positive
then, for each i, $a_i(k_1(\mathbf{x}, \mathbf{z}))^i$ is a positive-definite kernel function by (iii) and (iv) above.
Moreover, according to (i), we can show that $\sum_{i=0}^{n} a_i(k_1(\mathbf{x}, \mathbf{z}))^i$ is a positive-definite kernel function.
Therefore, $k(\mathbf{x}, \mathbf{z}) = p(k_1(\mathbf{x}, \mathbf{z}))$ is a positive-definite kernel function.

(vii) Prove that the Gaussian Kernel $k(\mathbf{x}, \mathbf{z}) = \exp\left(\frac{-\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma^2}\right)$ can be expressed as $\phi(\mathbf{x})^T\phi(\mathbf{z})$, where $\phi(\cdot)$ is an infinite-dimensional vector. (Hint: using power series)

Proof: As we know, the Taylor Expansion of $\exp(x)$ is $1 + x + \frac{x^2}{2!} + ...$
So

$$\begin{aligned}
\exp\left(\frac{-\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma^2}\right) &= \exp\left(\frac{-(\mathbf{x}-\mathbf{z})^T(\mathbf{x}-\mathbf{z})}{2\sigma^2}\right) \\
&= \exp\left(\frac{-\mathbf{x}^T\mathbf{x} - \mathbf{z}^T\mathbf{z} + 2\mathbf{x}^T\mathbf{z}}{2\sigma^2}\right) \\
&= \exp\left(-\frac{\|\mathbf{x}\|^2}{2\sigma^2}\right)\exp\left(-\frac{\|\mathbf{z}\|^2}{2\sigma^2}\right)\exp\left(\frac{\mathbf{x}^T\mathbf{z}}{\sigma^2}\right) \\
&= \exp\left(-\frac{\|\mathbf{x}\|^2}{2\sigma^2}\right)\exp\left(-\frac{\|\mathbf{z}\|^2}{2\sigma^2}\right)\left(1 + \frac{\mathbf{x}^T\mathbf{z}}{\sigma^2} + \frac{1}{2!}(\frac{\mathbf{x}^T\mathbf{z}}{\sigma^2})^2 + ...\right)
\end{aligned}$$

Since the Taylor Expansion of $\exp\left(\frac{\mathbf{x}^T\mathbf{z}}{\sigma^2}\right)$ is a positive coefficients polynomial, we can express it as $\phi(\mathbf{x})^T\phi(\mathbf{z})$ for some $\phi(.)$
Therefore,

$$\exp\left(\frac{-\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma^2}\right) = \left[\exp\left(-\frac{\|\mathbf{x}\|^2}{2\sigma^2}\right)\phi(\mathbf{x})\right]^T\left[\exp\left(-\frac{\|\mathbf{z}\|^2}{2\sigma^2}\right)\phi(\mathbf{z})\right]$$

is a positive-definite kernel function.

□

4) **Kernelized Ridge Regression (20 points).**

Recall that the error function for ridge regression (linear regression with L2 regularization) is:

$$E(\mathbf{w}) = (\Phi\mathbf{w} - \mathbf{t})^T(\Phi\mathbf{w} - \mathbf{t}) + \lambda\mathbf{w}^T\mathbf{w}$$

and its closed-form solution and model are:

$$\hat{\mathbf{w}} = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T\mathbf{t} \text{ and } \hat{f}(\mathbf{x}) = \hat{\mathbf{w}}^T\phi(\mathbf{x}) = \mathbf{t}^T\Phi(\Phi^T\Phi + \lambda I)^{-1}\phi(\mathbf{x})$$

Now we want to kernelize ridge regression and allow non-linear models.

(a) Use the following matrix inverse lemma to derive the closed-form solution and model for kernelized ridge regression:

$$(P + QRS)^{-1} = P^{-1} - P^{-1}Q(R^{-1} + SP^{-1}Q)^{-1}SP^{-1}$$

where P is an $n \times n$ invertible matrix, R is a $k \times k$ invertible matrix, Q is an $n \times k$ matrix and S is a $k \times n$ matrix. Make sure that your kernelized model only depends on the feature vectors $\phi(\mathbf{x})$ through inner products with other feature vectors.

Proof: Suppose $P = \lambda I$, $Q = \Phi^T$, $R = I$, and $S = \Phi$, then

$$\begin{aligned}
\hat{\mathbf{w}} &= (\Phi^T\Phi + \lambda I)^{-1}\Phi^T\mathbf{t} \\
&= \left(\frac{1}{\lambda}I - (\frac{1}{\lambda}I)\Phi^T(I + \Phi(\frac{1}{\lambda}I)\Phi^T)^{-1}\Phi(\frac{1}{\lambda}I)\right)\Phi^T\mathbf{t} \\
&= \frac{1}{\lambda}\left[I - \Phi^T(\lambda I + \Phi\Phi^T)^{-1}\Phi\right]\Phi^T\mathbf{t} \\
&= \frac{1}{\lambda}\Phi^T\left[I - (\lambda I + \Phi\Phi^T)^{-1}\Phi\Phi^T\right]\mathbf{t} \\
&= \frac{1}{\lambda}\Phi^T\left[(\lambda I + \Phi\Phi^T)^{-1}(\lambda I + \Phi\Phi^T) - (\lambda I + \Phi\Phi^T)^{-1}\Phi\Phi^T\right]\mathbf{t} \\
&= \Phi^T(\lambda I + \Phi\Phi^T)^{-1}\mathbf{t}
\end{aligned}$$

In the class, we know that $\mathbf{K} = \Phi\Phi^T$ which $\mathbf{K}_{nm} = \phi(\mathbf{x}_n)^T\phi(\mathbf{x}_m)$
So

$$\hat{\mathbf{w}} = \Phi^T(\lambda I + \mathbf{K})^{-1}\mathbf{t}$$

and

$$\hat{f}(\mathbf{x}) = \hat{\mathbf{w}}^T\phi(\mathbf{x}) = \phi(\mathbf{x})^T\hat{\mathbf{w}} = \phi(\mathbf{x})^T\Phi^T(\lambda I + \mathbf{K})^{-1}\mathbf{t} = \mathbf{k}(\mathbf{x})^T(\lambda I + \mathbf{K})^{-1}\mathbf{t}$$

where the n-th entry of $\mathbf{k}(\mathbf{x})$ is $k(\mathbf{x}, \mathbf{x}_n) = \phi(\mathbf{x})^T\phi(\mathbf{x}_n)$

□

(b) Apply kernelized ridge regression to the steel ultimate tensile strength dataset. The training data and test data are provided in `steel_composition_train.csv` and `steel_composition_test.csv`, respectively. We recommend you to normalize the data before applying the models. Report the RMSE (Root Mean Square Error) of the models on the training data. Try (set $\lambda = 1$)

(i) Polynomial kernel $k(\mathbf{u}, \mathbf{v}) = (\langle \mathbf{u}, \mathbf{v} \rangle + 1)^2$

Answer: the RMSE is 7.1430

(ii) Polynomial kernel $k(\mathbf{u}, \mathbf{v}) = (\langle \mathbf{u}, \mathbf{v} \rangle + 1)^3$

Answer: the RMSE is 4.4210

(iii) Polynomial kernel $k(\mathbf{u}, \mathbf{v}) = (\langle \mathbf{u}, \mathbf{v} \rangle + 1)^4$

Answer: the RMSE is 2.5478

(iv) Gaussian kernel $k(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{2\sigma^2}\right)$ (set $\sigma = 1$)

Answer: the RMSE is 6.9406

Code for problem 4:

```
from sklearn import preprocessing
from numpy.linalg import inv
import numpy as np
from matplotlib import pyplot as plt

def readData(filename, test=False):
  # read data from files
  cols = (1,2,3,4,5,6,7,8,9)
  if test:
    cols = (1,2,3,4,5,6,7,8)
  data = np.loadtxt(filename, delimiter=",", skiprows=1, usecols=cols)

  X = data[:, :-1]
  t = data[:, -1]

  if test:
    X = data[:, :]
    t = None

  return X, t

def rmse(predictions, targets):
  # calculate RMSE between predictions and targets
  predictions = np.array(predictions)
  targets = np.array(targets)
  return np.sqrt(((predictions - targets) ** 2).mean())

def getK(X, kernel):
  N, M = X.shape
  K = np.zeros((N, N))
  for i in xrange(N):
    for j in xrange(i, N):
      temp = kernel(X[i, :], X[j, :])
      K[i, j] = temp
      K[j, i] = temp

  return K

def predict(train_X, train_t, test_X, kernel):
  K = getK(train_X, kernel)

  temp = inv(np.identity(K.shape[0]) + K)

  predictions = []
  for i in xrange(test_X.shape[0]):
    k = np.zeros(train_X.shape[0])
    x = test_X[i, :]
    for j in xrange(train_X.shape[0]):
            k[j] = kernel(x, train_X[j, :])

        k = np.matrix(k)
        temp = np.matrix(temp)
        t = np.matrix(train_t).T
        predictions.append((k * temp * t)[0,0])

    return np.array(predictions)



# Part 1
def kernelA(u, v):
    return (u.dot(v) + 1) ** 2

# Part 2
def kernelB(u, v):
    return (u.dot(v) + 1) ** 3

# Part 3
def kernelC(u, v):
    return (u.dot(v) + 1) ** 4

# Part 4
def kernelD(u, v):
    return np.exp(- float(sum((u - v)**2)) / float(2) )



def main():
    train_X, train_t = readData('steel_composition_train.csv')

    scaler = preprocessing.StandardScaler().fit(train_X)
    train_X_std = scaler.transform(train_X)

    kernel_list = [kernelA, kernelB, kernelC, kernelD]
    name_list = ['(i)', '(ii)', '(iii)', '(iv)']

    for name, kernel in zip(name_list, kernel_list):
        predictions = predict(train_X_std, train_t, train_X_std, kernel)
        print "%s the training rmse is: %f" % (name, rmse(predictions, train_t))



if __name__ == "__main__":
    main()
```