EECS 545 Machine Learning Homework 3

Tsung-Hung Yao

February 23, 2016

1) Support Vector Machine

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:

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

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

Proof. From the constraints of the original optimization problem, we can see

$$\begin{aligned} & t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \qquad (i = 1, \ldots, N) \end{aligned} \tag{3}$$

$\implies \xi_i \geq 0$ and $\xi_i \geq 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \implies \xi_i \geq \max(0, 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b))$
Since we want to min the objective function, we can just have $\xi_i = \max(0, 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b))$ □

Problem (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. Since $\xi_i^* > 0$, we can know $\xi_i^* = 1 - t^{(i)}(\mathbf{w}{*}^T\mathbf{x}^{(i)} + b^*)$
Also, distance for training data to hyperplane $(\mathbf{w}^*)^T\mathbf{x} + b^* - 1 = 0$ is $|r| = \frac{|(\mathbf{w}^*)^T\mathbf{x}^{(i)} + b^* - 1|}{||\mathbf{w}^*||} = \frac{|\xi_i^*|}{||\mathbf{w}^*||} \propto \xi_i^*$ □

Problem (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.

Proof. Let's denote the Indicator function $I(A) = \begin{cases} 1 & \text{if } A \text{ is true} \\ 0 & \text{if } A \text{ is NOT true} \end{cases}$

$\nabla_{\mathbf{w}} \max\left(0, 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)\right) = \begin{cases} 0 & \text{if } t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 \\ -t^{(i)}\mathbf{x}^{(i)} & \text{if } t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) < 1 \end{cases}$

$\implies \nabla_{\mathbf{w}} E(\mathbf{w}, b) = \mathbf{w} - C\sum_{i=1}^{N}(t^{(i)}\mathbf{x}^{(i)} I(t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) < 1))$

$\frac{\partial}{\partial b} \max\left(0, 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)\right) = \begin{cases} 0 & \text{if } t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 \\ -t^{(i)} & \text{if } t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) < 1 \end{cases}$

$\implies \frac{\partial}{\partial b} E(\mathbf{w}, b)$

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

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

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

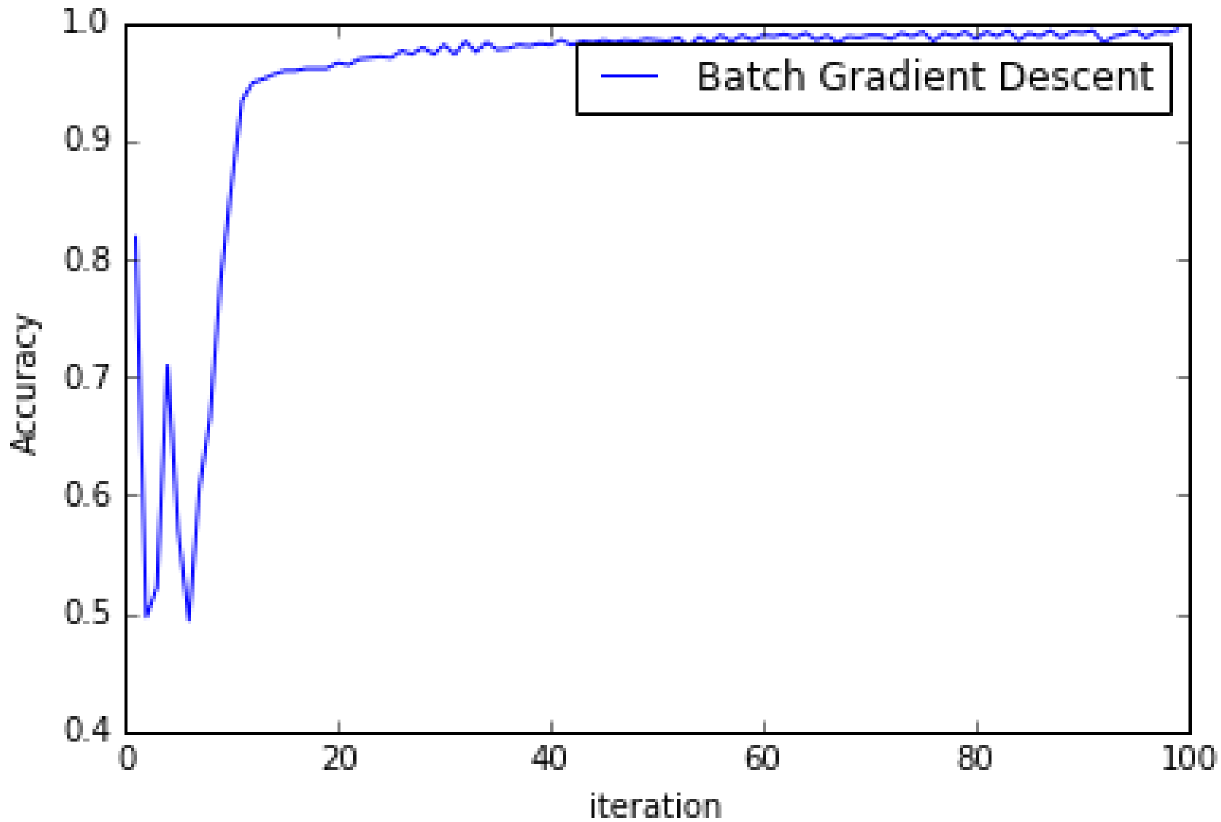
$\mathbf{w}^* \leftarrow \mathbf{0}$;
$b^* \leftarrow 0$;
for *j=1 to NumIterations* **do**
  $\mathbf{w}_{grad} \leftarrow \nabla_{\mathbf{w}} E(\mathbf{w}^*, b^*)$;
  $b_{grad} \leftarrow \frac{\partial}{\partial b} E(\mathbf{w}^*, b^*)$;
  $\mathbf{w}^* \leftarrow \mathbf{w}^* - \alpha(j)\ \mathbf{w}_{grad}$;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



Batch Gradient Descent

Problem (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.

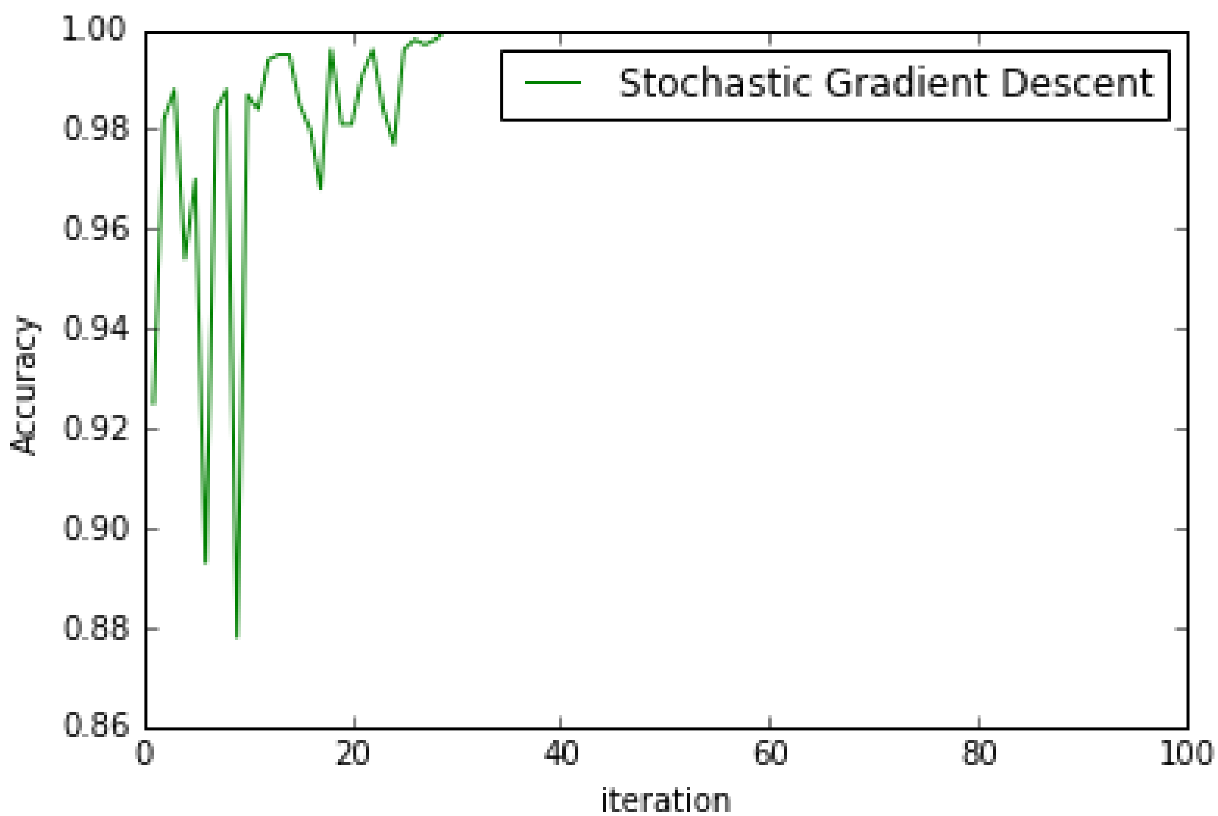
Proof. From (c), we have
$\nabla_{\mathbf{w}} \max\left(0, 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)\right) = -t^{(i)}\mathbf{x}^{(i)} I(t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) < 1)$ and
$\frac{\partial}{\partial b} \max\left(0, 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)\right) = -t^{(i)} I(t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) < 1) < 1)$
$\implies \nabla_{\mathbf{w}} E^{(i)}(\mathbf{w}, b) = \frac{1}{N}\mathbf{w} - Ct^{(i)}\mathbf{x}^{(i)} I(t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) < 1)$
and $\frac{\partial}{\partial b} E^{(i)}(\mathbf{w}, b) = -Ct^{(i)} I(t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) < 1)$ □

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

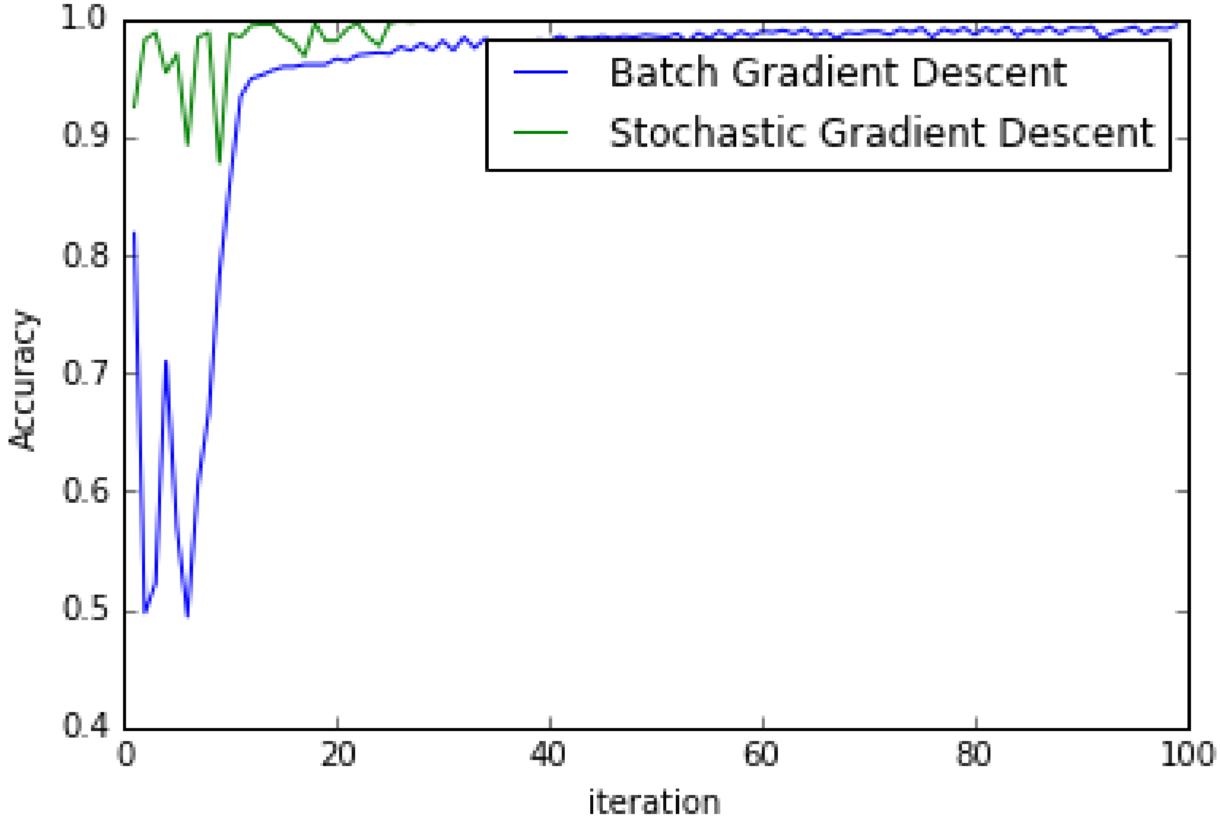
$\mathbf{w}^* \leftarrow \mathbf{0}$;
$b^* \leftarrow 0$;
for *j=1 to NumIterations* **do**
  for i = *Random Permutation of 1 to N* **do**
    $\mathbf{w}_{grad} \leftarrow \nabla_{\mathbf{w}} E^{(i)}(\mathbf{w}^*, b^*)$;
    $b_{grad} \leftarrow \frac{\partial}{\partial b} E^{(i)}(\mathbf{w}^*, b^*)$;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



Stochastic Gradient Descent



Two different Gradient Descent

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

Proof. From the graph above, we can see the SGD (stochastic gradient descent) converges way faster than BGD (batch gradient descent). SGD goes to the convergent point at the very beginning of the iteration. On the other hand, BGD doesn't converges till 20th iteration. □

Problem (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?

Proof.

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & t_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \qquad (i = 1, \ldots, N) \end{aligned} \tag{4}$$

$\implies L(\mathbf{w}, b, \xi, \alpha, \beta) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i(t_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i) - \sum_{i=1}^{N}\beta_i\xi_i$

$\implies$ We can have the dual problem:

$$\begin{aligned} \max_{\alpha,\beta>0} \quad & L_D(\alpha, \beta) \\ \text{where} \quad & L_D(\alpha, \beta) = \min_{\mathbf{w},b,\xi} L(\mathbf{w}, b, \xi, \alpha, \beta) \end{aligned} \tag{5}$$

Since this is unconstrained minimization with a convex differentiable objecti function. Therefore, for the fixed α, β, the minimizing $\mathbf{w}, b, \xi$ satisfy
$\frac{\partial}{\partial \mathbf{w}} L_D = \mathbf{w} - \sum_{i=1}^N \alpha_i t_i \mathbf{x}_i = 0 \implies \mathbf{w} = \sum_{i=1}^N \alpha_i t_i \mathbf{x}_i$
$\frac{\partial}{\partial b} L_D = \sum_{i=1}^N \alpha_i t_i = 0$
$\frac{\partial}{\partial \xi_i} L_D = C - \alpha_i - \beta_i = 0 \implies C = \alpha_i + \beta_i$
$\implies L_D = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^N \xi_i - \sum_{i=1}^N \alpha_i(t_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i) - \sum_{i=1}^N \beta_i\xi_i$
$= \frac{1}{2}(\sum_{i=1}^N \alpha_i t_i \mathbf{x}_i)^\top(\sum_{i=1}^N \alpha_i t_i \mathbf{x}_i) + (\sum_{i=1}^N C\xi_i) - \sum_{i=1}^N(\alpha_i t_i \mathbf{w}^T\mathbf{x}_i + \alpha_i t_i b - \alpha_i + \alpha_i\xi_i) - \sum_{i=1}^N \beta_i\xi_i$
$= \frac{1}{2}\sum_{i,j=1}^N \alpha_i\alpha_j t_i t_j \mathbf{x}_i^\top\mathbf{x}_j + (\sum_{i=1}^N(\alpha_i+\beta_i)\xi_i) - \sum_{i=1}^N(\alpha_i t_i(\sum_{j=1}^N \alpha_j t_j \mathbf{x}_j^\top)\mathbf{x}_i) + \sum_{i=1}^N \alpha_i - \sum_{i=1}^N \alpha_i\xi_i - \sum_{i=1}^N \beta_i\xi_i$
$= -\frac{1}{2}\sum_{i,j=1}^N \alpha_i\alpha_j t_i t_j \mathbf{x}_i^\top\mathbf{x}_j + \sum_{i=1}^N \alpha_i$
$\implies$ The dual problem is:

$$\begin{aligned}
\underset{\alpha,\beta}{\text{maximize}} \quad & -\frac{1}{2}\sum_{i,j=1}^{n} \alpha_i\alpha_j t_i t_j < \mathbf{x}_i, \mathbf{x}_j > + \sum_i \alpha_i \\
\text{subject to} \quad & \sum_i \alpha_i t_i = 0 \\
& \forall i, \alpha_i + \beta_i = \frac{C}{n} \\
& \forall i, \alpha_i \geq 0, \beta_i \geq 0
\end{aligned}$$

□

Problem (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.

Proof. Training accuracy: 1.0; Test accuracy: 0.977955911824 with parameter C=100 and gamma=1e-7 Since training accuracy is 1.0, there's no mis-classification. For the Misclassified in test data set, see the Fig 1 below. □

Problem (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.

Proof. Training accuracy: 0.998; Test accuracy: 0.9
For the misclassification image see the fig 2 for Training data set and fig 3 for test data set.
□

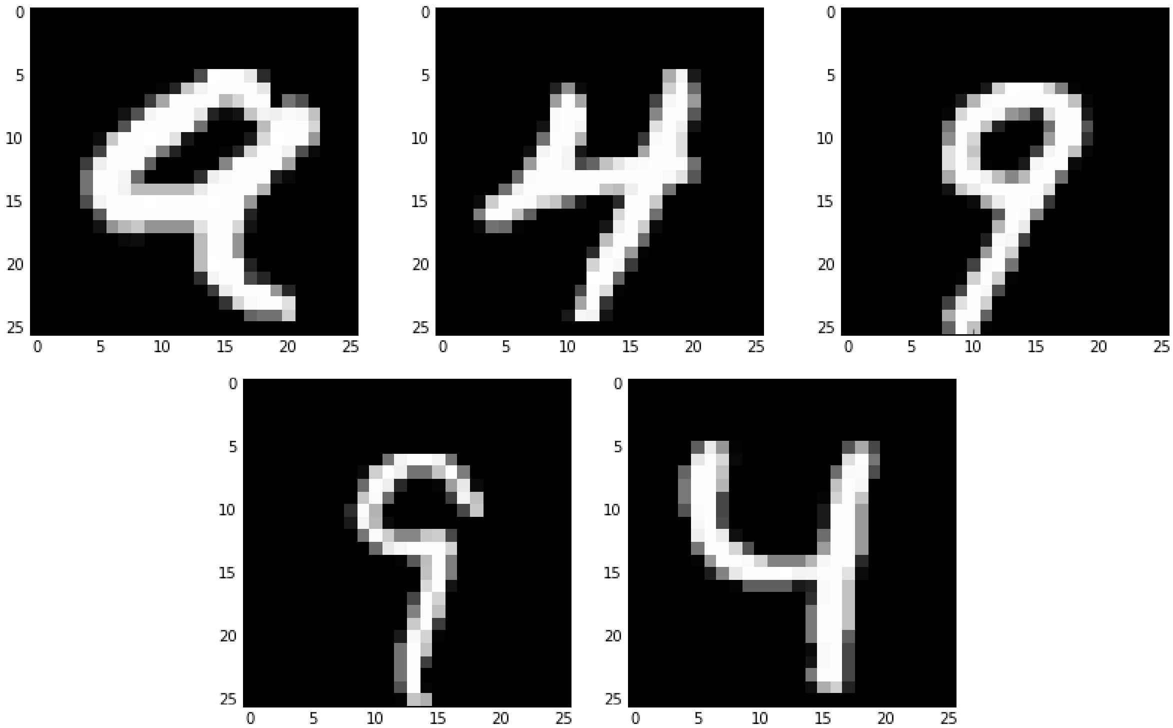


Figure 1: Pictures of SVM misclassification in test data set

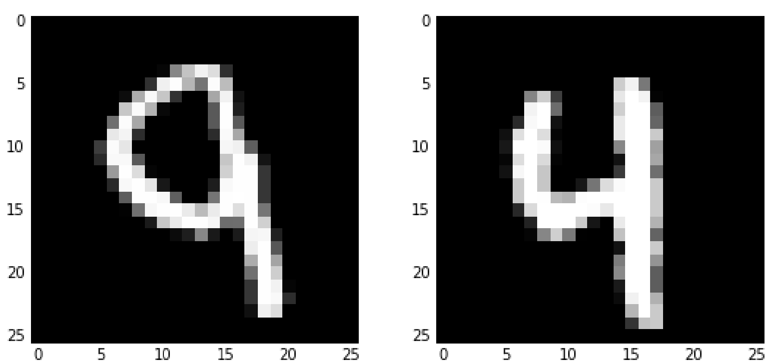


Figure 2: Pictures of LDA misclassification in train data set

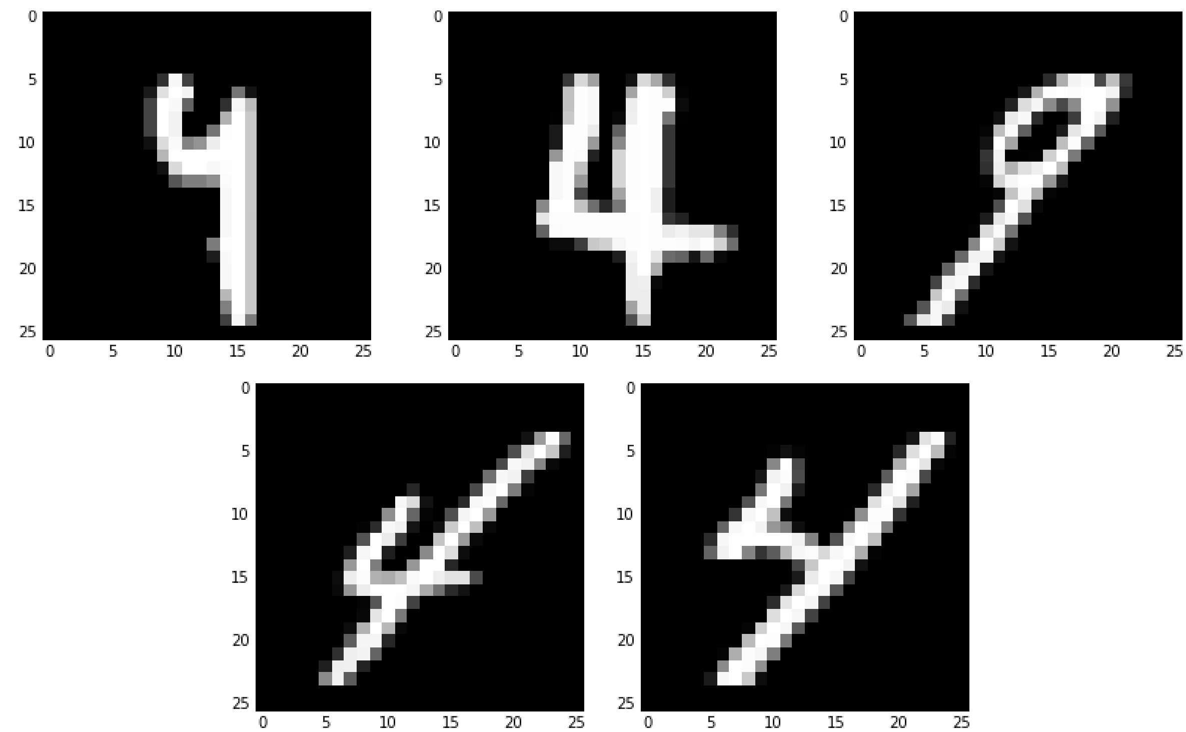


Figure 3: Pictures of LDA misclassification in test data set

2) Open Kaggle challenge

You can use any algorithm and design any features to perform classification on the hand-written 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.

uniqueame: yaots with score 0.9488

3) Constructing Kernels

Problem (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 $a = \langle \mathbf{u}, \mathbf{v} \rangle \implies k(\mathbf{u}, \mathbf{v}) = (a+1)^4 = a^4 + 4a^3 + 6a^2 + 4a + 1$
Here d=3, which means $a = u_1 v_1 + u_2 v_2 + u_3 v_3$
$a^4 = (u_1 v_1 + u_2 v_2 + u_3 v_3)^4 = k_4(\mathbf{u}, \mathbf{v}) = \phi_4(\mathbf{u})^\top \phi_4(\mathbf{v})$

, where $\phi_4(\mathbf{u}) = \begin{pmatrix} u_1^4 \\ u_2^4 \\ u_3^4 \\ 2u_1^3u_2 \\ 2u_1u_2^3 \\ 2u_1^3u_3 \\ 2u_1u_3^3 \\ 2u_2^3u_3 \\ 2u_2u_3^3 \\ \sqrt{6}u_1^2u_2^2 \\ \sqrt{6}u_2^2u_3^2 \\ \sqrt{6}u_1^2u_3^2 \\ 2\sqrt{3}u_1^2u_2u_3 \\ 2\sqrt{3}u_1u_2^2u_3 \\ 2\sqrt{3}u_1u_2u_3^2 \end{pmatrix}$

Similarly, we can have
$4a^3 = 4k_3(\mathbf{u},\mathbf{v}) = 2\phi_3(\mathbf{u})^\top 2\phi_3(\mathbf{v})$
$6a^2 = 6k_2(\mathbf{u},\mathbf{v}) = \sqrt{6}\phi_2(\mathbf{u})^\top \sqrt{6}\phi_2(\mathbf{v})$
$4a^1 = 4k_1(\mathbf{u},\mathbf{v}) = 2\phi_1(\mathbf{u})^\top 2\phi_1(\mathbf{v})$
where

$\phi_3(\mathbf{u}) = \begin{pmatrix} u_1^3 \\ u_2^3 \\ u_3^3 \\ \sqrt{3}u_1^2u_2 \\ \sqrt{3}u_1u_2^2 \\ \sqrt{3}u_1^2u_3 \\ \sqrt{3}u_1u_3^2 \\ \sqrt{3}u_2^2u_3 \\ \sqrt{3}u_2u_3^2 \\ \sqrt{6}u_1u_2u_3 \end{pmatrix}$, $\phi_2(\mathbf{u}) = \begin{pmatrix} u_1^2 \\ u_2^2 \\ u_3^2 \\ \sqrt{2}u_1u_2 \\ \sqrt{2}u_1u_3 \\ \sqrt{2}u_2u_3 \end{pmatrix}$ and $\phi_1(\mathbf{u}) = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$ Therefore, we can have

$k(\mathbf{u},\mathbf{v}) = (a+1)^4 = a^4 + 4a^3 + 6a^2 + 4a + 1$
$= \phi_4(\mathbf{u})^\top\phi_4(\mathbf{v}) + 2\phi_3(\mathbf{u})^\top 2\phi_3(\mathbf{v}) + \sqrt{6}\phi_2(\mathbf{u})^\top\sqrt{6}\phi_2(\mathbf{v}) + 2\phi_1(\mathbf{u})^\top 2\phi_1(\mathbf{v}) + 1 = \phi(\mathbf{u})^\top\phi(\mathbf{v})$
where $\phi(\mathbf{u})^\top = \begin{pmatrix} \phi_4(\mathbf{u})^\top & 2\phi_3(\mathbf{u})^\top & \sqrt{6}\phi_2(\mathbf{u})^\top & 2\phi_1(\mathbf{u})^\top & 1 \end{pmatrix}$

For arbitrary d, it still follows $k(\mathbf{u},\mathbf{v}) = \phi(\mathbf{u})^\top\phi(\mathbf{v})$, where
$\phi(\mathbf{u})^\top = \begin{pmatrix} \phi_4(\mathbf{u})^\top & 2\phi_3(\mathbf{u})^\top & \sqrt{6}\phi_2(\mathbf{u})^\top & 2\phi_1(\mathbf{u})^\top & 1 \end{pmatrix}$
$\phi_4(\mathbf{u}) = \begin{pmatrix} u_i^4, i = 1,2,...d \\ 2u_i^3u_j, i \neq j \\ \sqrt{6}u_i^2u_j^2, i \neq j \\ 2\sqrt{3}u_i^2u_ju_k, i \neq j \neq k \end{pmatrix}$, $\phi_3(\mathbf{u}) = \begin{pmatrix} u_i^3, i = 1,2,...d \\ \sqrt{3}u_i^2u_j^1, i \neq j \\ \sqrt{6}u_iu_ju_k, i \neq j \neq k \end{pmatrix}$ and $\phi_2(\mathbf{u}) = \begin{pmatrix} u_i^2, i = 1,2,...d \\ \sqrt{2}u_iu_j, i \neq j \end{pmatrix}$

□

Problem (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.

Before the following proof, let's denote the kernel matrix of each kernel function:

$$\mathbf{C} = \begin{pmatrix} k_1(\mathbf{x_1}, \mathbf{z_1}) & k_1(\mathbf{x_1}, \mathbf{z_2}) & \cdots & k_1(\mathbf{x_1}, \mathbf{z_n}) \\ k_1(\mathbf{x_2}, \mathbf{z_1}) & k_1(\mathbf{x_2}, \mathbf{z_2}) & \cdots & k_1(\mathbf{x_2}, \mathbf{z_n}) \\ \cdots & \cdots & \ddots & \cdots \\ k_1(\mathbf{x_n}, \mathbf{z_1}) & k_1(\mathbf{x_n}, \mathbf{z_2}) & \cdots & k_1(\mathbf{x_n}, \mathbf{z_n}) \end{pmatrix}$$

$$\mathbf{D} = \begin{pmatrix} k_2(\mathbf{x_1}, \mathbf{z_1}) & k_2(\mathbf{x_1}, \mathbf{z_2}) & \cdots & k_2(\mathbf{x_1}, \mathbf{z_n}) \\ k_2(\mathbf{x_2}, \mathbf{z_1}) & k_2(\mathbf{x_2}, \mathbf{z_2}) & \cdots & k_2(\mathbf{x_n}, \mathbf{z_n}) \\ \cdots & \cdots & \ddots & \cdots \\ k_2(\mathbf{x_n}, \mathbf{z_1}) & k_2(\mathbf{x_n}, \mathbf{z_2}) & \cdots & k_2(\mathbf{x_n}, \mathbf{z_n}) \end{pmatrix}$$

$$\mathbf{E} = \begin{pmatrix} k(\mathbf{x_1}, \mathbf{z_1}) & k(\mathbf{x_1}, \mathbf{z_2}) & \cdots & k(\mathbf{x_1}, \mathbf{z_n}) \\ k(\mathbf{x_2}, \mathbf{z_1}) & k(\mathbf{x_2}, \mathbf{z_2}) & \cdots & k(\mathbf{x_2}, \mathbf{z_n}) \\ \cdots & \cdots & \ddots & \cdots \\ k(\mathbf{x_n}, \mathbf{z_1}) & k(\mathbf{x_n}, \mathbf{z_2}) & \cdots & k(\mathbf{x_n}, \mathbf{z_n}) \end{pmatrix}$$

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

Proof. $k(\mathbf{x}, \mathbf{z})$ is a PD kernel
$\mathbf{u}^\top \mathbf{E} \mathbf{u} = \mathbf{u}^\top (\mathbf{C} + \mathbf{D}) \mathbf{u} = \mathbf{u}^\top \mathbf{C} \mathbf{u} + \mathbf{u}^\top \mathbf{D} \mathbf{u}$
$\because k_1$ and k_2 are PD kernel
$\therefore \mathbf{u}^\top \mathbf{C} \mathbf{u} \geq 0$ and $\mathbf{u}^\top \mathbf{D} \mathbf{u} \geq 0, \forall \mathbf{u} \in \mathbb{R}^n$
$\implies \mathbf{u}^\top \mathbf{E} \mathbf{u} \geq 0 \implies k(\mathbf{x}, \mathbf{z})$ is a PD kernel □

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

Proof. $k(\mathbf{x}, \mathbf{z})$ is NOT a PD kernel
Let $k_1(\mathbf{x}, \mathbf{z}) = 0 \implies \mathbf{u}^\top \mathbf{E} \mathbf{u} = -\mathbf{u}^\top \mathbf{D} \mathbf{u} \leq 0 \implies k(\mathbf{x}, \mathbf{z})$ is not a PD kernel □

(iii) $k(\mathbf{x}, \mathbf{z}) = a k_1(\mathbf{x}, \mathbf{z})$

Proof. $k(\mathbf{x}, \mathbf{z})$ is a PD kernel
$\mathbf{u}^\top \mathbf{E} \mathbf{u} = a \mathbf{u}^\top \mathbf{D} \mathbf{u} \geq 0 \implies k(\mathbf{x}, \mathbf{z})$ is PD □

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

Proof. $k(\mathbf{x}, \mathbf{z})$ is a PD kernel
$\mathbf{u}^\top \mathbf{E} \mathbf{u} = \sum_{i,j} u_i u_j e_{i,j} = \sum_{i,j} u_i (c_{i,j} d_{i,j}) u_j$
$\because \mathbf{C} = \mathbf{P} \mathbf{\Lambda} \mathbf{P}^\top = \sum_{k=1}^n (\lambda_k \mathbf{P}_k \mathbf{P}_k^\top) \implies c_{i,j} = \sum_{k=1}^n (\lambda_k P_{k,i} P_{k,j})$
$\therefore \sum_{i,j} u_i (c_{i,j} d_{i,j}) u_j = \sum_{i,j} u_i (\sum_{k=1}^n (\lambda_k P_{k,i} P_{k,j}) d_{i,j}) u_j = \sum_{i,j} (\sum_{k=1}^n (\lambda_k u_i P_{k,i} P_{k,j} u_j d_{i,j})) = \sum_{k=1}^n (\sum_{i,j} (\lambda_k u_i P_{k,i} d_{i,j} P_{k,j} u_j)) = \sum_{k=1}^n (\lambda_k \sum_{i,j} (u_i P_{k,i} d_{i,j} P_{k,j} u_j))$
$\because \mathbf{D}$ is PSD matrix $\implies \sum_{i,j} (u_i P_{k,i} d_{i,j} P_{k,j} u_j) \geq 0$

Also, since $\mathbf{C}$ is PSD, the eigenvalue of $\mathbf{C}$ is non-negative $\implies \lambda_k \geq 0 \implies \mathbf{u}^\top \mathbf{E}\mathbf{u} = \sum_{k=1}^{n}(\lambda_k \sum_{i,j}(u_i P_{k,i} d_{i,j} P_{k,j} u_j)) \geq 0 \implies k(\mathbf{x}, \mathbf{z})$ is a PD kernel □

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

Proof. $k(\mathbf{x}, \mathbf{z})$ is NOT a PD kernel
Consider $f(\mathbf{x}) = -1$ and $f(\mathbf{z}) = 1 \implies \mathbf{u}^\top \mathbf{E}\mathbf{u} = \mathbf{u}^\top \begin{pmatrix} -1_n & -1_n & \cdots & -1_n \end{pmatrix} \mathbf{u} = -(\sum_{i=1}^{n} u_i)^2 \leq 0 \implies k(\mathbf{x}, \mathbf{z})$ is NOT a PD kernel □

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

Proof. $k(\mathbf{x}, \mathbf{z})$ is a PD kernel

$$\mathbf{u}^\top \mathbf{E}\mathbf{u} = \mathbf{u}^\top \begin{pmatrix} p(k_1(\mathbf{x_1}, \mathbf{z_1})) & p(k_1(\mathbf{x_1}, \mathbf{z_2})) & \cdots & p(k_1(\mathbf{x_1}, \mathbf{z_n})) \\ p(k_1(\mathbf{x_2}, \mathbf{z_1})) & p(k_1(\mathbf{x_2}, \mathbf{z_2})) & \cdots & p(k_1(\mathbf{x_2}, \mathbf{z_n})) \\ \cdots & \cdots & \ddots & \cdots \\ p(k_1(\mathbf{x_n}, \mathbf{z_1})) & p(k_1(\mathbf{x_n}, \mathbf{z_2})) & \cdots & p(k_1(\mathbf{x_n}, \mathbf{z_n})) \end{pmatrix} \mathbf{u} = \sum_{i,j} u_i u_j p(c_{i,j})$$

Since the polynomial function is the linear combination of multiplication, addition, from the results from (i), (iii) and (iv), we can have $\mathbf{u}^\top \mathbf{E}\mathbf{u} = \sum_{i,j} u_i u_j p(c_{i,j}) \geq 0 \implies \mathbf{E}$ is a PD kernel □

(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. $k(\mathbf{x}, \mathbf{z})$ is a PD kernel

$exp\left(\frac{-\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma^2}\right) = exp\left(\frac{-\mathbf{x}^\top\mathbf{x}}{2\sigma^2}\right) exp\left(\frac{-\mathbf{z}^\top\mathbf{z}}{2\sigma^2}\right) exp\left(\frac{\mathbf{x}^\top\mathbf{z}}{\sigma^2}\right)$

Also, since $exp(x) = \sum_{i=1}^{\infty}(\frac{x^i}{i!}) \implies exp\left(\frac{\mathbf{x}^\top\mathbf{z}}{\sigma^2}\right) = \sum_{i=1}^{\infty}(\frac{(\mathbf{x}^\top\mathbf{z})^i}{\sigma^2 i!})$

$\implies \phi(\mathbf{x})^\top = exp\left(\frac{-\mathbf{x}^\top\mathbf{x}}{2\sigma^2}\right) \begin{pmatrix} 1 & (\frac{\mathbf{x}^\top}{\sigma})^1\frac{1}{\sqrt{1!}} & (\frac{\mathbf{x}^\top}{\sigma})^2\frac{1}{\sqrt{2!}} & (\frac{\mathbf{x}^\top}{\sigma})^3\frac{1}{\sqrt{3!}} & \cdots & (\frac{\mathbf{x}^\top}{\sigma})^j\frac{1}{\sqrt{j!}} & \cdots \end{pmatrix}$

$\because k(\mathbf{x}, \mathbf{z})$ can be expressed as $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^\top\phi(\mathbf{z})$, from Mercer Thm, we can know this is also a valid kernel function. Therefore, the Gram matrix derived from kernel is PSD $\implies k(\mathbf{x}, \mathbf{z})$ is PSD kernel □

4) Kernelized Ridge Regression

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.

Problem (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. $(P + QRS)^{-1} = P^{-1} - P^{-1}Q(R^{-1} + SP^{-1}Q)^{-1}SP^{-1}$
Let $P = \lambda I, Q = \Phi^T, R = IS = \Phi$
$\implies (\Phi^T\Phi + \lambda I)^{-1} = \frac{1}{\lambda}I - \frac{1}{\lambda}\Phi^\top(I + \Phi\frac{1}{\lambda}\Phi^\top)^{-1}\Phi\frac{1}{\lambda} = \frac{1}{\lambda}(I - \Phi^\top(\lambda I + \Phi\Phi^\top)^{-1}\Phi)$
$\implies \hat{\mathbf{w}} = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T\mathbf{t} = \frac{1}{\lambda}[I - \Phi^\top(\lambda I + \Phi\Phi^\top)^{-1}\Phi]\Phi^T\mathbf{t} = \frac{1}{\lambda}[\Phi^\top - \Phi^\top(\lambda I + \Phi\Phi^\top)^{-1}\Phi\Phi^\top]\mathbf{t} = \frac{\Phi^\top}{\lambda}[I - (\lambda I + K)^{-1}K]\mathbf{t}$
Also, $I - (\lambda I + K)^{-1}K = (\lambda I + K)^{-1}(\lambda I + K) - (\lambda I + K)^{-1}K = (\lambda I + K)^{-1}((\lambda I + K) - K) = \lambda(\lambda I + K)^{-1}$
$\implies \hat{\mathbf{w}} = \frac{\Phi^\top}{\lambda}[I - (\lambda I + K)^{-1}K]\mathbf{t} = \frac{\Phi^\top}{\lambda}[\lambda(\lambda I + K)^{-1}]\mathbf{t} = \Phi^\top(\lambda I + K)^{-1}\mathbf{t}$
$\implies \hat{\mathbf{w}}^\top\phi(\mathbf{x}) = \mathbf{t}^\top(\lambda I + K^\top)^{-1}\Phi(\mathbf{x})\phi(\mathbf{x}) = \mathbf{t}^\top(\lambda I + K)^{-1}k(\mathbf{x})$ ($\because K^\top = K$ and $k(\mathbf{x}) = \Phi(\mathbf{x})\phi(\mathbf{x})$) □

Problem (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_te` 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 $k(\mathbf{u}, \mathbf{v}) = (\langle \mathbf{u}, \mathbf{v} \rangle + 1)^2$
(ii) Polynomial kernel $k(\mathbf{u}, \mathbf{v}) = (\langle \mathbf{u}, \mathbf{v} \rangle + 1)^3$
(iii) Polynomial kernel $k(\mathbf{u}, \mathbf{v}) = (\langle \mathbf{u}, \mathbf{v} \rangle + 1)^4$
(iv) Gaussian kernel $k(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{2\sigma^2}\right)$ (set $\sigma = 1$)

Proof. RMSE for the kernel above are the following below:
RMSE of the kernel (i): 7.1430391254115539
RMSE of the kernel (ii): 4.4210302978118117
RMSE of the kernel (iii): 2.5477671261017258
RMSE of the kernel (iv):6.9406391493574136] □

5) Code Appendix

Problem 1) Support Vector Machine (d). Code

```
import numpy as np
from matplotlib import pyplot as plt
train_feature=np.genfromtxt("digits_training_data.csv",delimiter=",")
train_label=np.genfromtxt("digits_training_labels.csv",delimiter=",")
train_label=train_label.reshape((np.shape(train_label)[0],1))
for i in range(len(train_label)):
```

```
    if train_label[i]==9:
        train_label[i]=1
    else:
        train_label[i]=-1
train_data=np.hstack((train_feature, train_label))
test_feature=np.genfromtxt("digits_test_data.csv",delimiter=",")
test_label=np.genfromtxt("digits_test_labels.csv",delimiter=",")
test_label=test_label.reshape((np.shape(test_label)[0],1))
for i in range(len(test_label)):
    if test_label[i]==9:
        test_label[i]=1
    else:
        test_label[i]=-1
test_data=np.hstack((test_feature, test_label))

weight=np.zeros((np.shape(train_feature)[1],1))
bias=0

def get_Trainaccu(Weight, b, feature, label):
    predicted=np.dot(feature, Weight)
    b_ones=b*np.ones(np.shape(predicted))
    predicted+=b_ones
    count=0
    for i in range(len(predicted)):
        if predicted[i]>=0 and label[i]==1:
            count+=1
        if predicted[i]<0 and label[i]==-1:
            count+=1
    return float(count)/len(predicted)

print get_Trainaccu(weight, bias, train_feature, train_label)

def Indicator(Weight, Bias, feature_i, label_i):
    if label_i*(np.dot(Weight.T,feature_i)+Bias)<1:
        return 1
    else:
        return 0

print Indicator(weight, bias, train_feature[0,:],train_label[0])

def Gradient(Weight, Bias, Cost, feature, label):
    sumE=np.zeros(np.shape(Weight))
    for i in range(len(label)):
        feature_T=feature[i].reshape((np.shape(feature[i])[0],1))
        sumE+=(label[i]*Indicator(Weight, Bias, feature[i],label[i])*feature_T
    Wgrad=Weight-Cost*sumE
    sumB=0
```

```
    for i in range(len(label)):
        sumB+=(label[i]*Indicator(Weight, Bias, feature[i],label[i]))
    Bgrad=-Cost*sumB
    return (Wgrad,Bgrad)

def update(Weight, Bias, Iter_i, eta, gradient):
    alpha_i=float(eta)/(1+Iter_i*eta)
    Weight=Weight-alpha_i*gradient[0]
    Bias=Bias-alpha_i*gradient[1]
    return (Weight, Bias)
ite_lst=[]
Trainaccu_lstBGD=[]
for i in range(1,100):
    gradient=Gradient(weight, bias, 3, train_feature, train_label)
    weight=update(weight, bias, i, 0.001, gradient)[0]
    bias=update(weight, bias, i, 0.001, gradient)[1]
    Trainaccu=get_Trainaccu(weight, bias, train_feature, train_label)
    ite_lst.append(i)
    Trainaccu_lstBGD.append(Trainaccu)
print weight, np.shape(weight)
print bias, np.shape(bias)
print get_Trainaccu(weight,bias,test_feature, test_label)

plt.plot(ite_lst,Trainaccu_lstBGD,label="Batch Gradient Descent")
plt.xlabel("iteration")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

Problem 1) Support Vector Machine (f). Code

```
import numpy as np
from matplotlib import pyplot as plt
train_feature=np.genfromtxt("digits_training_data.csv",delimiter=",")
train_label=np.genfromtxt("digits_training_labels.csv",delimiter=",")
train_label=train_label.reshape((np.shape(train_label)[0],1))
for i in range(len(train_label)):
    if train_label[i]==9:
        train_label[i]=1
    else:
        train_label[i]=-1
train_data=np.hstack((train_feature, train_label))
test_feature=np.genfromtxt("digits_test_data.csv",delimiter=",")
test_label=np.genfromtxt("digits_test_labels.csv",delimiter=",")
test_label=test_label.reshape((np.shape(test_label)[0],1))
for i in range(len(test_label)):
    if test_label[i]==9:
        test_label[i]=1
```

```
    else:
        test_label[i]=-1
test_data=np.hstack((test_feature, test_label))

weight=np.zeros((np.shape(train_feature)[1],1))
bias=0

def get_Trainaccu(Weight, b, feature, label):
    predicted=np.dot(feature, Weight)
    b_ones=b*np.ones(np.shape(predicted))
    predicted+=b_ones
    count=0
    for i in range(len(predicted)):
        if predicted[i]>=0 and label[i]==1:
            count+=1
        if predicted[i]<0 and label[i]==-1:
            count+=1
    return float(count)/len(predicted)

print get_Trainaccu(weight, bias, train_feature, train_label)

def Indicator(Weight, Bias, feature_i, label_i):
    if label_i*(np.dot(Weight.T,feature_i)+Bias)<1:
        return 1
    else:
        return 0

print Indicator(weight, bias, train_feature[0,:],train_label[0])

def Gradient_i(Weight, Bias, Cost, feature_i, label_i, N):
    feature_T=feature_i.reshape((np.shape(feature_i)[0],1))
    I_i=Indicator(Weight, Bias, feature_i, label_i)
    Wgrad=Weight/float(N)-Cost*label_i*I_i*feature_T
    Bgrad=-Cost*label_i*I_i
    return (Wgrad,Bgrad)

def update(Weight, Bias, Iter_i, eta, gradient):
    alpha_i=float(eta)/(1+Iter_i*eta)
    Weight=Weight-alpha_i*gradient[0]
    Bias=Bias-alpha_i*gradient[1]
    return (Weight, Bias)
ite_lst=[]
Trainaccu_lstSGD=[]
for i in range(1,100):
    sample_Num=len(train_label)
    for j in np.random.choice(sample_Num,sample_Num,replace=False):
        gradient=Gradient_i(weight, bias, 3, train_feature[j], train_label[j],s
```

```
        weight=update(weight, bias, i, 0.001, gradient)[0]
        bias=update(weight, bias, i, 0.001, gradient)[1]
    Trainaccu=get_Trainaccu(weight, bias, train_feature, train_label)
    ite_lst.append(i)
    Trainaccu_lstSGD.append(Trainaccu)

print ite_lst
print Trainaccu_lstSGD
plt.plot(ite_lst,Trainaccu_lstSGD,label="Stochastic Gradient Descent")
plt.plot(ite_lst,Trainaccu_lstBGD,label="Batch Gradient Descent")
plt.xlabel("iteration")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

Problem 1) Support Vector Machine (i). Code

```
import pandas as pd
import numpy as np
from sklearn import svm
from sklearn.metrics import accuracy_score
import matplotlib.cm as cm
from matplotlib import pyplot as plt

# Read in csv file and conver to a numpy array
data = np.genfromtxt('./digits_training_data.csv', delimiter=',')

# plot a random training image (row)
k = int(np.random.random()*data.shape[0])
plt.imshow(data[k].reshape((26,26)), interpolation="nearest", cmap=cm.Greys_r)


X_train = pd.read_csv("digits_training_data.csv")
Y_train=pd.read_csv("digits_training_labels.csv")
X_test = pd.read_csv("digits_test_data.csv")
Y_test=pd.read_csv("digits_test_labels.csv")

clfTst_svm = svm.SVC(kernel="rbf", C=100, gamma=1e-7)
clfTst_svm.fit(X_train, Y_train)
y_predTst_svm = clfTst_svm.predict(X_test)
acc_svm = accuracy_score(Y_test, y_predTst_svm)
print "SVM accuracy: ",acc_svm

clfTr_svm = svm.SVC(kernel="rbf", C=100, gamma=1e-7)
clfTr_svm.fit(X_train, Y_train)
y_predTr_svm = clfTst_svm.predict(X_train)
acc_svm = accuracy_score(Y_train, y_predTr_svm)
print "SVM accuracy: ",acc_svm
```

```
test=np.array(Y_test.values)
test_lst=[]
for i in range(len(test)):
    test_lst.append(test[i][0])
print test_lst

count=0
error_lst=[]
for i in range(len(test_lst)):
    if test_lst[i]==y_predTst_svm[i]:
        count+=1
    else:
        error_lst.append(i)
print float(count)/len(test_lst)
print error_lst

drawX_test = np.genfromtxt('./digits_test_data.csv', delimiter=',')
for i in error_lst:
    plt.imshow(drawX_test[i].reshape((26,26)), interpolation="nearest", cmap=cm
    plt.show()
```

Problem 1) Support Vector Machine (j). Code

```
import numpy as np
import matplotlib.cm as cm
from matplotlib import pyplot as plt

X_train = np.genfromtxt("digits_training_data.csv",delimiter=",")
Y_train=np.genfromtxt("digits_training_labels.csv",delimiter=",")
Y_train=Y_train.reshape(np.shape(Y_train)[0],1)
trSampNum=np.shape(Y_train)[0]
featureNum=np.shape(X_train)[1]
X_test =np.genfromtxt("digits_test_data.csv",delimiter=",")
Y_test=np.genfromtxt("digits_test_labels.csv",delimiter=",")
Y_test=Y_test.reshape(np.shape(Y_test)[0],1)
tstSampNum=np.shape(Y_test)[0]
print trSampNum, featureNum
print np.shape(X_train), np.shape(Y_train)
print np.shape(X_test), np.shape(Y_test)

trainData=np.hstack((X_train,Y_train))
testData=np.hstack((X_test,Y_test))
print np.shape(trainData), np.shape(testData)

train9=trainData[trainData[:,featureNum]==9]
train4=trainData[trainData[:,featureNum]==4]
pi9=len(train9)
```

```
pi4=len(train4)
print pi9, pi4
train9_mean=np.mean(train9,axis=0)
train4_mean=np.mean(train4,axis=0)
train9_mean=train9_mean.reshape((np.shape(train9_mean)[0],1))
train4_mean=train4_mean.reshape((np.shape(train4_mean)[0],1))
print np.shape(train9_mean),np.shape(train4_mean)
train9_XMean=train9_mean[0:featureNum,:]
train4_XMean=train4_mean[0:featureNum,:]
print np.shape(train9_XMean), np.shape(train4_XMean)
CovMatr=np.cov(trainData[:,0:featureNum].T)
InvCov=np.linalg.pinv(CovMatr)
print np.shape(CovMatr)

y_predTr=[]
for i in range(trSampNum):
    value9=np.dot(np.dot(train9_XMean.T,InvCov),X_train[i,:])-0.5*np.dot(np.dot
    value4=np.dot(np.dot(train4_XMean.T,InvCov),X_train[i,:])-0.5*np.dot(np.dot
    if value9>=value4:
        y_predTr.append(9)
    else:
        y_predTr.append(4)

y_predTst=[]
for i in range(tstSampNum):
    value9=np.dot(np.dot(train9_XMean.T,InvCov),X_test[i,:])-0.5*np.dot(np.dot(
    value4=np.dot(np.dot(train4_XMean.T,InvCov),X_test[i,:])-0.5*np.dot(np.dot(
    if value9>=value4:
        y_predTst.append(9)
    else:
        y_predTst.append(4)

def get_accu(pred,target):
    error_lst=[]
    count=0
    for i in range(len(pred)):
        if pred[i]==target[i]:
            count+=1
        else:
            error_lst.append(i)
    return (float(count)/len(pred),error_lst)

Y_trainErrLst=get_accu(y_predTr,Y_train)[1]
Y_testErrLst=get_accu(y_predTst,Y_test)[1]
print get_accu(y_predTr,Y_train)
print get_accu(y_predTst,Y_test)
```

```
for i in Y_trainErrLst:
    plt.imshow(X_train[i].reshape((26,26)), interpolation="nearest", cmap=cm.Gr
    plt.show()

for i in Y_testErrLst:
    plt.imshow(X_test[i].reshape((26,26)), interpolation="nearest", cmap=cm.Gre
    plt.show()
```

Problem 2) Open Kaggle Challenge. Code

```
import numpy as np
from sklearn import cross_validation
from sklearn.ensemble import RandomForestClassifier
trainLabels = np.loadtxt('trainingLabels.gz', dtype=np.uint8, delimiter=',')
trainData = np.loadtxt('trainingData.gz', dtype=np.uint8, delimiter=',')
testData = np.loadtxt('testData.gz',dtype=np.uint8,delimiter=',')
clf_rf = RandomForestClassifier()
clf_rf.fit(trainData, trainLabels)
y_pred_rf = clf_rf.predict(testData)

f=open("HW3Kaggle_1.csv",'w')
writer=csv.writer(f)
writer.writerow(["id","category"])
for i in range(len(y_pred_rf)):
    writer.writerow([i+1,y_pred_rf[i]])
f.close()
```

Problem 4) Kernelized Ridge Regression (b). Code

```
import numpy as np
trD_raw=np.genfromtxt("steel_composition_train.csv",delimiter=",")
trD_raw=trD_raw[1:,1:]
featureNum=np.shape(trD_raw)[1]-1
sampleNum=np.shape(trD_raw)[0]
print featureNum
print sampleNum
trD_mean=np.mean(trD_raw,axis=0)
trD_var=np.var(trD_raw,axis=0)
trD=(trD_raw-trD_mean)/np.sqrt(trD_var)
trF=trD[:,0:featureNum]
trL=trD[:,featureNum]
trL_nonNorm=trD_raw[:,featureNum]

def poly_kernelFun(Samp1, Samp2, power):
    value=(np.dot(Samp1,Samp2)+1)**power
    return value

def Gauker(Samp1, Samp2):
```

```python
        dist=np.dot(Samp1-Samp2,Samp1-Samp2)
        value=np.exp(-dist/2)
        return value

def Poly_Gram_Mtr(Feature1, Feature2, power):
    SamNum=len(Feature1)
    GramMat=np.zeros((SamNum,SamNum))
    for i in range(SamNum):
        for j in range(SamNum):
            GramMat[i,j]=poly_kernelFun(Feature1[i,:],Feature2[j,:],power)
    return GramMat

def Gau_Gram_Mtr(Feature1, Feature2):
    SamNum=len(Feature1)
    GramMat=np.zeros((SamNum,SamNum))
    for i in range(SamNum):
        for j in range(SamNum):
            GramMat[i,j]=Gauker(Feature1[i,:],Feature2[j,:])
    return GramMat

def Gram_Vec(Feature, queryPt, kerPow):
    SampNum=len(Feature)
    GramVec=np.zeros((SampNum,1))
    if kerPow==0:
        for i in range(SampNum):
            GramVec[i,0]=Gauker(Feature[i],queryPt)
    else:
        for i in range(SampNum):
            GramVec[i,0]=poly_kernelFun(Feature[i],queryPt,kerPow)
    return GramVec

def get_pred(Feature,Label, kerPow):
    sampNum=len(Label)
    I_N=np.eye(sampNum)
    pred=np.zeros(sampNum)
    if kerPow==0:
        GramMtr=Gau_Gram_Mtr(Feature, Feature)
        for i in range(len(pred)):
            GramVec=Gram_Vec(Feature, Feature[i,:], 0)
            pred[i]=np.dot(Label.T,np.dot(np.linalg.inv(I_N+GramMtr),GramVec))
    else:
        GramMtr=Poly_Gram_Mtr(Feature, Feature, kerPow)
        for i in range(len(pred)):
            GramVec=Gram_Vec(Feature, Feature[i,:], kerPow)
            pred[i]=np.dot(Label.T,np.dot(np.linalg.inv(I_N+GramMtr),GramVec))
    return pred
```

```
def get_RMSE(pred, Label):
    return np.sqrt(((pred - Label) ** 2).mean())

RMSE_lst=[]
for i in range(2,5):
    pred=get_pred(trF,trL_nonNorm,i)
    RMSE=get_RMSE(pred, trL_nonNorm)
    RMSE_lst.append(RMSE)

pred=get_pred(trF,trL_nonNorm,0)
RMSE=get_RMSE(pred, trL_nonNorm)
RMSE_lst.append(RMSE)

print RMSE_lst
```