Stony Brook University
CSE512 – Machine Learning – Spring 17
Homework 2, Due: March, 7, 2017, 11:59PM

Instructions

- The homework is due on March 7, 2017. Anything that is received after the deadline will not be considered.
- The write-up **must** be prepared in Latex, including the Matlab code and figures in the report.
- We can use any Latex class you like, just report question number and your answer.
- If the question requires you to implement a Matlab function, the answer should be your code. Make sure it is sufficiently well documented that the TAs can understand what is happening.
- Each Question, regardless of how many sub-questions contains, is worth 10 points.

1 Ridge Regression

1.1 Question 1

In class, you learned about using k-folds cross validation as a way to estimate the true error of a learning algorithm and to tune parameters. The preferred solution is *Leave-One-Out Cross Validation* (LOOCV), which provides an almost unbiased estimate of this true error, but it can take a really long time to compute. In this problem, you will derive an algorithm for efficiently computing the LOOCV error for a particular regression algorithm.

We will now introduce a simple extension of the Least Square algorithm. Given a set of m data points and associated labels $\{\boldsymbol{x}_i, y_i | \boldsymbol{x}_i \in \Re^d, y_i \in \Re\}_{i=1}^m$. **Ridge Regression** finds the weight vector $\boldsymbol{w}$ and a bias term b to optimize the following:

$$\min_{\boldsymbol{w},b} \ \lambda\|\boldsymbol{w}\|^2 + \sum_{i=1}^{m} (\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)^2 \ . \tag{1}$$

Note that with $\lambda = 0$ Ridge Regression is exactly the Least Square formulation.

- Using the Matlab notation, let $\bar{\boldsymbol{w}} = [\boldsymbol{w}; b]$, $\bar{X} = [X; \mathbf{1}^\top]$, $\bar{I} = [I, \mathbf{0}; \mathbf{0}^\top, 0]$, $C = \bar{X}\bar{X}^\top + \lambda \bar{I}$, and $\boldsymbol{d} = \bar{X}\boldsymbol{y}$. Show that the solution of Ridge regression is:

$$\bar{\boldsymbol{w}} = C^{-1}\boldsymbol{d} \ . \tag{2}$$

We know Ridge regression finds weight vector as

$$\min_{\boldsymbol{w},b} \ \lambda\|\boldsymbol{w}\|^2 + \sum_{i=1}^{m} (\boldsymbol{w}^\top \boldsymbol{x}_i + b - y_i)^2$$

We also know $\bar{\boldsymbol{w}} = [\boldsymbol{w}; b], \bar{X} = [X; \mathbf{1}^\top], \bar{I} = [I, \mathbf{0}; \mathbf{0}^\top, 0]$, $C = \bar{X}\bar{X}^\top + \lambda\bar{I}$, and $\boldsymbol{d} = \bar{X}\boldsymbol{y}$
Therefore, $\|\bar{\boldsymbol{w}}\|^2 = \|\boldsymbol{w}\|^2 + b^2$ and we get ridge expression in vector form as follow:

$$\min_{\bar{\boldsymbol{w}}} \ \lambda(\|\bar{\boldsymbol{w}}\|^2 + b^2) + \|\bar{\boldsymbol{w}}^\top \bar{X} - Y\|^2$$

But we know $\bar{\boldsymbol{w}}^\top \bar{X} = \bar{X}^\top \bar{\boldsymbol{w}}$, thus we have:

$$\min_{\bar{\boldsymbol{w}}} \ \lambda(\|\bar{\boldsymbol{w}}\|^2 + b^2) + \|\bar{X}^\top \bar{\boldsymbol{w}} - Y\|^2$$

Using property of the norm we have:

$$\min_{\bar{\boldsymbol{w}}} \ \lambda(\|\bar{\boldsymbol{w}}\|^2 + b^2) + \|Y - \bar{X}^\top \bar{\boldsymbol{w}}\|^2$$

As ridge expression is convex function of $\bar{\boldsymbol{w}}$, so it suffices to find $\bar{\boldsymbol{w}}$ where gradient is zero. Therefore, on taking first moment w.r.t. $\bar{\boldsymbol{w}}$, we get:

$$2\lambda\bar{\boldsymbol{w}} + 2\bar{X}(\bar{X}^\top \bar{\boldsymbol{w}} - Y) = 0$$

Thus, we have:

$$\lambda\bar{I}\bar{\boldsymbol{w}} + \bar{X}\bar{X}^\top \bar{\boldsymbol{w}} = \bar{X}Y$$

$$\bar{\boldsymbol{w}}(\bar{X}\bar{X}^\top + \lambda\bar{I}) = \bar{X}Y$$

$$\bar{\boldsymbol{w}}C = \boldsymbol{d}$$

Hence $\bar{\boldsymbol{w}} = C^{-1}\boldsymbol{d}$ proved

- Now suppose we remove $\boldsymbol{x}_i$ from the training data, let $C_{(i)}, \boldsymbol{d}_{(i)}, \bar{\boldsymbol{w}}_{(i)}$ be the corresponding matrices for removing $\boldsymbol{x}_i$. Express $C_{(i)}$ in terms of C and $\boldsymbol{x}_i$. Express $\boldsymbol{d}_{(i)}$ in terms of $\boldsymbol{d}$ and $\boldsymbol{x}_i$.

  We know $C = \bar{X}\bar{X}^\top + \lambda\bar{I}$, But $\bar{X}\bar{X}^\top$ represents summation of dot product of $x_{(i)}.x_{(i)}^\top$ for every feature combination where feature is represented by row number and column number. $\lambda\bar{I}$ adds a constant to the overall value to particular feature and doesn't depend on samples.

  Thus, we can say that,

  $$C_{(i)} = C - x_{(i)}.x_{(i)}^\top$$

  Similarly we know that $\boldsymbol{d} = \bar{X}\boldsymbol{y}$. Thus, d_i is summation of $x.y$ for all samples. So by removing of sample, we just don't add the $x.y$ for the particular sample for that particular feature. Therefore, we can say that:

  $$d_{(i)} = d - x_{(i)}.y_{(i)}$$

- Express $C_{(i)}^{-1}$ in terms of C^{-1} and $\boldsymbol{x}_i$. *Hint:* use the Sherman-Morrison formula

  $$(A + \boldsymbol{u}\boldsymbol{v}^\top)^{-1} = A^{-1} - \frac{A^{-1}\boldsymbol{u}\boldsymbol{v}^\top A^{-1}}{1 + \boldsymbol{v}^\top A^{-1}\boldsymbol{u}} \ .$$

  From 1)a and 1)b, we have,

  $$(A + \boldsymbol{u}\boldsymbol{v}^\top)^{-1} = (C - x_i x_i^T)^{-1}$$

Using Sherman-Morrison formula:

$$(A + \boldsymbol{u}\boldsymbol{v}^\top)^{-1} = A^{-1} - \frac{A^{-1}\boldsymbol{u}\boldsymbol{v}^\top A^{-1}}{1 + \boldsymbol{v}^\top A^{-1}\boldsymbol{u}} \ .$$

Replacing A by C, u by $-x_i, v^T$ *by* x_i^T*, we get,*

$$(C - x_i x_i^T)^{-1} = C^{-1} + \frac{C^{-1}x_i x_i^T C^{-1}}{1 - x_i^T C^{-1} x_i}$$

$$C_i^{-1} = C^{-1} + \frac{C^{-1}x_i x_i^T C^{-1}}{1 - x_i^T C^{-1} x_i}$$

$$C_i^{-1} = C^{-1} + \frac{C^{-1}x_i x_i^T C^{-1}}{1 - x_i^T C^{-1} x_i}$$

- Show that

$$\bar{\boldsymbol{w}}_{(i)} = \bar{\boldsymbol{w}} + (C^{-1}\bar{\boldsymbol{x}}_i)\frac{\bar{\boldsymbol{x}}_i^\top \bar{\boldsymbol{w}} - y_i}{1 - \bar{\boldsymbol{x}}_i^\top C^{-1}\bar{\boldsymbol{x}}_i} \ .$$

$$\bar{\boldsymbol{w}}_{(i)} = C_i^{-1} d_i$$

$$\bar{\boldsymbol{w}}_{(i)} = \left(C^{-1} + \frac{C^{-1}\boldsymbol{x}_i\boldsymbol{x}_i^\top C^{-1}}{1 - \boldsymbol{x}_i^\top C^{-1}\boldsymbol{x}_i} \right)(d - x_i y_i)$$

$$\bar{\boldsymbol{w}}_{(i)} = C^{-1}d - C^{-1}x_i y_i + \frac{C^{-1}\boldsymbol{x}_i\boldsymbol{x}_i^\top C^{-1}d}{1 - \boldsymbol{x}_i^\top C^{-1}\boldsymbol{x}_i} - \frac{C^{-1}\boldsymbol{x}_i\boldsymbol{x}_i^\top C^{-1}x_i y_i}{1 - \boldsymbol{x}_i^\top C^{-1}\boldsymbol{x}_i}$$

$$\bar{\boldsymbol{w}}_{(i)} = w + \frac{C^{-1}\boldsymbol{x}_i\boldsymbol{x}_i^\top \bar{w} - C^{-1}x_i y_i - C^{-1}\boldsymbol{x}_i\boldsymbol{x}_i^\top C^{-1}x_i y_i + C^{-1}\boldsymbol{x}_i y_i \boldsymbol{x}_i^\top C^{-1}x_i}{1 - \boldsymbol{x}_i^\top C^{-1}\boldsymbol{x}_i}$$

$$\bar{\boldsymbol{w}}_{(i)} = w + (C^{-1}\boldsymbol{x}_i)\frac{\bar{\boldsymbol{x}}_i^\top \bar{\boldsymbol{w}} - y_i}{1 - \boldsymbol{x}_i^\top C^{-1}\boldsymbol{x}_i}$$

Since y_i is scalar order of placement doesn't matter

- Show that the leave-one-out error for removing the i^{th} training sample is

$$\bar{\boldsymbol{w}}_{(i)}^\top \bar{\boldsymbol{x}}_i - y_i = \frac{\bar{\boldsymbol{w}}^\top \bar{\boldsymbol{x}}_i - y_i}{1 - \bar{\boldsymbol{x}}_i^\top C^{-1}\bar{\boldsymbol{x}}_i} \ . \tag{3}$$

$$\bar{w_i}^T = w^T + \frac{({x_i}^T w - y_i)^T {x_i}^T (C^{-1})^T}{1 - {x_i}^T C^{-1} x_i}$$

$$= w^T + \frac{(w^T x_i - y_i){x_i}^T {C^{-1}}^T}{1 - {x_i}^T C^{-1} x_i}$$

[Here, y_i is scalar so won't have any transpose effect]

$$= \frac{(w^T - w^T {x_i}^T C^{-1} x_i + w^T {x_i}^T C^{-1^T} - y_i {x_i}^T C^{-1^T}) x_i - y_i}{1 - {x_i}^T C^{-1} x_i}$$

As C^{-1} is symmetric matrix, therefore ${C^{-1}}^\top$ is equal to C^{-1}.Now we will just solve for the numerator N,

$$N = w^T x_i - w^T {x_i}^T C^{-1} x_i x_i + w^T x_i {x_i}^T C^{-1} x_i - y_i {x_i}^T C^{-1} x_i - y_i + {x_i}^T C^{-1} x_i y_i$$

After cancelling the ${x_i}^T C^{-1} x_i y_i$ terms, we get,

$$N = -w^T {x_i}^T C^{-1} x_i x_i + w^T x_i {x_i}^T C^{-1} x_i + w^T x_i - y_i$$

Now, ${x_i}^T C^{-1} x_i$ is a scalar quantity which cancels the terms $-w^T {x_i}^T C^{-1} x_i x_i$ and $w^T x_i {x_i}^T C^{-1} x_i$, thus we are left with,

$$N = w^T x_i - y_i$$

Putting back this numerator in our original equation we get,

$${\bar{w}_i}^T = \frac{w^T x_i - y_i}{1 - {x_i}^T C^{-1} x_i}$$

Hence, proved.

- The LOOCV is defined as: $\sum_{i=1}^{m} (\bar{\boldsymbol{w}}_{(i)}^\top \bar{\boldsymbol{x}}_i - y_i)^2$. What is the algorithmic complexity of computing LOOCV error using the formula given in the previous point? How is it compared with the usual way of computing LOOCV? Note that the complexity of inverting a $k \times k$ matrix is $O(k^3)$.

  Here $w^\top$, C^{-1} calculation is done only once which takes O(n^3) and after that calculating error for each left out point using precalculated matrices takes O(n^2). we have to calculate this error for n points so asymptotic complexity of the approach is O(n^3)

  $$\frac{\bar{\boldsymbol{w}}^\top \bar{\boldsymbol{x}}_i - y_i}{1 - \bar{\boldsymbol{x}}_i^\top C^{-1} \bar{\boldsymbol{x}}_i}$$

  Usual Approach: we have to calculate $w_i^\top$ for each sample which takes O(n^3) time and then we have to repeat this for n points so time complexity becomes O(n^4)

  $$\sum_{i=1}^{m} (\bar{\boldsymbol{w}}_{(i)}^\top \bar{\boldsymbol{x}}_i - y_i)^2$$

1.2 Question 2

Implement the Ridge Regression algorithm that finds the solution of (**??**) through (**??**). The prototype of the function must be

```
[w,w_0] = train_rr(X, y, lambda)
```

where $X \in \mathbb{R}^{m \times d}$ is the matrix of m input vectors in d dimension, i.e. each input $\boldsymbol{x}_i$ is a row of X, y is a column vector of m columns containing the labels associated to the training samples. lambda is the value of λ in (**??**). You can assume $\lambda > 0$.

Solution We use the closed form of ridge regression here as mentioned in the above problem. Also, we know that the Matrix $\bar{X}X^T + \lambda I$ is always invertible and hence we can directly use the inverse function. Below is the matlab code for the same

```
function [w, w_0] = train_rr(X, y, lambda)
    [m,n] = size(X);
    Z = ones(m,1);
    X = [X Z];
    C = transpose(X)*X + lambda*eye(n+1);
    d = transpose(X)*y;
    res = inv(C) * d;
    w_0 = res(n+1);
    w= res(1:n);
end
```

2 Perceptron

Given a sequence of m samples $(\boldsymbol{x}_i, y_i)$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ and the labels $y_i \in \{-1, 1\}$, the Perceptron algorithm runs as follows:

Algorithm 1 Perceptron pseudocode

```
Initialize: w = 0
for i = 1, ··· , T do
  if y_i⟨w, x_i⟩ < 0 then
    w ← w + y_i x_i
  end if
end for
```

For simplicity, we have put the bias b into $\boldsymbol{w}$, that is $\boldsymbol{w} \leftarrow [\boldsymbol{w}; b]$ and $\boldsymbol{x}_i \leftarrow [\boldsymbol{x}_i; 1]$. So it is no necessary to consider the bias term in next two questions. Assume $\|\boldsymbol{x}_i\| \leq R$ for all i. We have shown that if there exists some w^* such that $\|w^*\| = 1$ and for all i $y_i \langle w^*, \boldsymbol{x}_i \rangle \geq \gamma$, then the number of mistakes is upper bounded by

$$\frac{R^2}{\gamma^2} \ . \tag{4}$$

2.1 Question 3

Consider the generalized Perceptron updates $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta y_i \boldsymbol{x}_i$ with *learning rate* $\eta > 0$. The Perceptron algorithm is the special case $\eta = 1$. Prove a bound on the number of mistakes similar to (**??**). How does η affect this bound?

$$
\begin{aligned}
\|w_i + 1\|^2 &= \|w_i\|^2 + 2\eta y_i \langle w_i, x_i \rangle + (\eta \|x\|)^2 \\
&\leq \|w_i\|^2 + (\eta R)^2 \,, \text{ since } \; 2y_i \langle w_i, x_i \rangle < 0 \text{ and } \|x\|^2 \leq R^2 \\
\langle w^*, w_T + 1 \rangle &\leq \eta R \sqrt{M}
\end{aligned}
$$

$$
\begin{aligned}
\langle w^*, w_T + 1 \rangle &= \langle w^*, \sum_t y_t x_t \rangle \\
&= \sum_t y_t \langle w^*, x_t \rangle \\
&\geq \sum_{i=1}^{M} \eta\gamma \\
&\geq M\eta\gamma \\
\eta R\sqrt{M} &\geq M\eta\gamma \\
M &\leq \frac{R^2}{\gamma^2}
\end{aligned}
$$

Number of mistakes doesn't depend on learning rate.

2.2 Question 4

Now let's drop the assumption that samples are linear separable. Let $\boldsymbol{u}$ be any vector with $\|\boldsymbol{u}\| = 1$ and let $\gamma > 0$. Define $\ell_i(\boldsymbol{u}) = \max(0, \gamma - y_i \langle \boldsymbol{u}, \boldsymbol{x}_i \rangle)$ and and $L_q(\boldsymbol{u}) = \sum_{i=1}^m \ell_i(\boldsymbol{u})^q$ for $1 \leq q \leq 2$.

- Building on the previous proof, use the fact that you can express the updates *on all rounds* as $\boldsymbol{w} \leftarrow \boldsymbol{w} + \tau_i y_i \boldsymbol{x}_i$ for $\tau_i \in \{0, 1\}$, and show that the number of mistakes M of the Perceptron algorithm satisfies the *implicit* (i.e. M appears on both sides) inequality

$$
\gamma M \leq M^{1-\frac{1}{q}} L_q^{\frac{1}{q}}(\boldsymbol{u}) + R\sqrt{M}, \; \forall 1 \leq q \leq 2 \tag{5}
$$

  Hint: Start from the proof seen in class, and use the Hölder's inequality:

$$
\sum_{i=1}^{m} a_i b_i \leq \left(\sum_{i=1}^{m} |a_i|^q \right)^{\frac{1}{q}} \left(\sum_{i=1}^{m} |b_i|^p \right)^{\frac{1}{p}} \,, \; \forall p, q \in [1, +\infty] \text{ such that } \frac{1}{p} + \frac{1}{q} = 1 \,.
$$

  Note that the Cauchy-Schwarz inequality is a special case of Hölder's inequality with $p = q = 2$.

$$
\begin{aligned}
\|w_i + 1\|^2 &= \|w_i\|^2 + 2y_i\langle w_i, x_i\rangle + \|x\|^2 \\
&\leq \|w_i\|^2 + R^2, \text{ since } \ 2y_i\langle w_i, x_i\rangle \leq 0 \text{ and } \|x\|^2 \leq R^2 \\
\langle w^*, w_T + 1\rangle &\leq R\sqrt{M}
\end{aligned}
$$

$$
\begin{aligned}
\langle w^*, w_T + 1\rangle &= \langle w^*, \sum_t y_t x_i\rangle \\
&= \sum_t y_t \langle w^*, x_i\rangle \\
&= \sum_{i=1}^{M} \gamma - \ell_i(\boldsymbol{u}) \\
&= M\gamma - \sum_{i=1}^{M} \ell_i(\boldsymbol{u}) \\
&= M\gamma - \sum_{i=1}^{M} \ell_i(\boldsymbol{u}) * 1 \\
&\geq M\gamma - \left(\sum_{i=1}^{m} \ell_i(\boldsymbol{u})^q\right)^{\frac{1}{q}} \left(\sum_{i=1}^{m} |1|^p\right)^{\frac{1}{p}}, \ \forall p, q \in [1, +\infty] \text{ such that } \frac{1}{p} + \frac{1}{q} = 1 \ . \\
&\geq M\gamma - \left(\sum_{i=1}^{m} \ell_i(\boldsymbol{u})^q\right)^{\frac{1}{q}} \left(\sum_{i=1}^{m} |1|^{1-\frac{1}{q}}\right)^{1-\frac{1}{q}}, \ \forall 1 \leq q \leq 2 \\
&\geq M\gamma - L_q^{\frac{1}{q}}(\boldsymbol{u}) M^{1-\frac{1}{q}}, \ \forall 1 \leq q \leq 2 \\
R\sqrt{M} &\geq M\gamma - L_q^{\frac{1}{q}}(\boldsymbol{u}) M^{1-\frac{1}{q}}, \ \forall 1 \leq q \leq 2 \\
M\gamma &\leq R\sqrt{M} + L_q^{\frac{1}{q}}(\boldsymbol{u}) M^{1-\frac{1}{q}}, \ \forall 1 \leq q \leq 2
\end{aligned}
$$

- Setting $q = 2$ in (**??**), find an *explicit* upper bound on M, that is M must be only on the left hand side of the inequality.

$$
\begin{aligned}
M\gamma &\leq R\sqrt{M} + L_q^{\frac{1}{q}}(\boldsymbol{u}) M^{1-\frac{1}{q}}, \ \forall 1 \leq q \leq 2 \\
M\gamma &\leq R\sqrt{M} + \sqrt{L_2(\boldsymbol{u})}\sqrt{M}, \text{ when q = 2} \\
M\gamma &\leq \sqrt{M}(R + \sqrt{L_2(\boldsymbol{u})}) \\
\sqrt{M} &\leq \frac{R + \sqrt{L_2(\boldsymbol{u})}}{\gamma} \\
M &\leq \left(\frac{R + \sqrt{L_2(\boldsymbol{u})}}{\gamma}\right)^2
\end{aligned}
$$

2.3 Question 5

Implement the Perceptron algorithm in Algorithm 1. The prototype of the function must be

`[w,w_0] = train_perceptron(X, y)`

where $X \in \mathbb{R}^{m \times d}$ is the matrix of m input vectors in d dimension, i.e. each input $\boldsymbol{x}_i$ is a row of X, y is a column vector of m columns containing the labels associated to the training samples. Learn `w_0` using the usual trick of augmenting the input data with a constant feature equal to 1.

Solution:

```
function [ w, w_0 ] = train_perceptron(X, y)
    [m, n] = size(X);
    X = [X ones(m,1)];
    w = zeros(n+1,1);
    for i = 1:m
        if y(i)*(X(i,:)* w) <= 0
            w = w + y(i)*X(i,:)';
        end
    end
    w_0 = w(n+1);
    w = w(1:n);
end
```

3 Support Vector Machines

3.1 Question 6

Consider training a SVM on a linearly separable dataset consisting of m points. Let n be the number of support vectors obtained by training on the entire set. Show that the LOOCV error is upper bounded by $\frac{n}{m}$. *Hint:* Consider two cases: (1) removing a support vector datapoint and (2) removing a non-support vector datapoint.

Solution:

(1) Remove support vector :
When we remove a support vector datapoint from the training set, the resulting hyperplane might change and we might get a non-zero chance of error on the left out support vector. Since the number of support vectors is n, the maximum number of LOOCV errors is n.
Therefore total number of errors is upper bounded by $\frac{n}{m}$

(2) Remove non-support vector:
As the non-support data point doesn't change the hyper plane,there will not be any change in the LOOCV error of the SVM.

3.2 Question 7

Here you have to implement the soft SVM classification problem in the primal form using Matlab quadratic solver.

Quadratic programs refer to optimization problems in which the objective function is quadratic and the constraints are linear. Many Machine Learning algorithms are reduced to solving quadratic programs. In this question, we will use the quadratic program solver of Matlab ('quadprog') to optimize the primal objective of a linear SVM.

For the primal objective function

$$\begin{aligned} \min_{\boldsymbol{w}} \ & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t. } & y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 - \xi_i, \ i = 1, \ldots, m \\ & \xi_i \geq 0, \ i = 1, \ldots, m \ . \end{aligned} \tag{6}$$

1. Cast the SVM primal objective as a quadratic program: Look at the `quadprog` function of Matlab, and write down what `H, f, A, b, Aeq, beq, lb, ub` are.

2. Using the above answer, implement a function

   ```
   [w,w_0] = train_svm_primal(X, y, C)
   ```

   that solves the primal problem for SVMs, where $X \in \mathbb{R}^{m\times d}$ is the matrix of m input vectors in d dimension, i.e. each input $\boldsymbol{x}_i$ is a row of X, y is a column vector of m columns containing the labels associated to the training samples. C is the constant in the primal formulation (**??**).

   Solution:

   (a) H is the matrix containing Identity matrix and appended by zeros to accomodate for the dimensions of W and X. It is of the for [I, 0; 0,0]. It is a symmetric matrix.

   f contains elements that are of data type double. It is initialized to be vector of ones. It represents the linear term in the expansion $1/2 * x' * H * x + f' * x$

   b is the constant in the given expression Ax ¿= b. In case of the SVM the constraints are given as -Ax ¿= -b. Thus it is vector of constants

   A contains elements that are of data type double. It is obtained by concatenating X samples and ones which is then multiplied with the diagonal entries of the y samples.

   Aeq is the matrix of doubles representing the constants in the constraint $Aeq * x = beq$

   beq is the vector of doubles and it represents the constant vectors in the constraint $Aeq * x = beq$

   (b) Below is the code for *train_svm_primal*

   ```
   function [ w, w_0,fval] = train_svm_primal(X, y, C)
       [m,d] = size(X)

       semi_identity = eye(d+1);
       semi_identity(d+1,d+1) = 0;

       Z = [X ones(m,1)];
       A = -diag(y)*Z;
       c = -C*ones(m,1);

       H = semi_identity;
       f = zeros(d+1,1);
   ```

```
        [w, fval] = quadprog(H,f,A,c);
        w_0 = w(d+1);
        w = w(1:d);
    end
```

4 Empirical Results

4.1 Question 8

In the following we will use the synthetic data in synthdata.mat. It is a 2-d classification dataset composed by 2 classes.

We will use the implementations of Rigde Regression, SVM, and the Perceptron to train linear classifiers. Note that, while Ridge Regression is typically used for regression, nothing prevents us to use it as a classification algorithm. In this case, the training procedure goes on as usual, and in the testing phase we predict with the sign of the prediction.

- Set $\lambda = 1$, train Ridge Regression using the training data in synthdata.mat. Test the obtained solution on test data in synthdata.mat, and report the accuracy.
- Set $C = 0.01$ in (**??**), use your implementation of the SVM and the training data in synthdata.mat. Test the obtained solution on test data in synthdata.mat, and report the accuracy, the objective value of the SVM, and the number of support vectors.
- Repeat the above question with $C = 100$.
- Run the Perceptron with one pass over the training data. Test the obtained solution on test data in synthdata.mat, and report the accuracy.
- Plot in 2d the separating hyperplanes obtained in all the above cases and the training points.

Solution

We will use the below function to calculate the accuracy of our trained model

```
function [ accuracy ] = accuracy( X, y, w, w_0 )
    [m,n] = size(X);
     w = [w_0;w];
     X = [ones(m,1) X];
     errors = 0;
     for i = 1: m
         if y(i)*(X(i,:)*w)<=0
             errors = errors+1;
         end
     end
     accuracy = (m-errors) / m * 100;
end
```

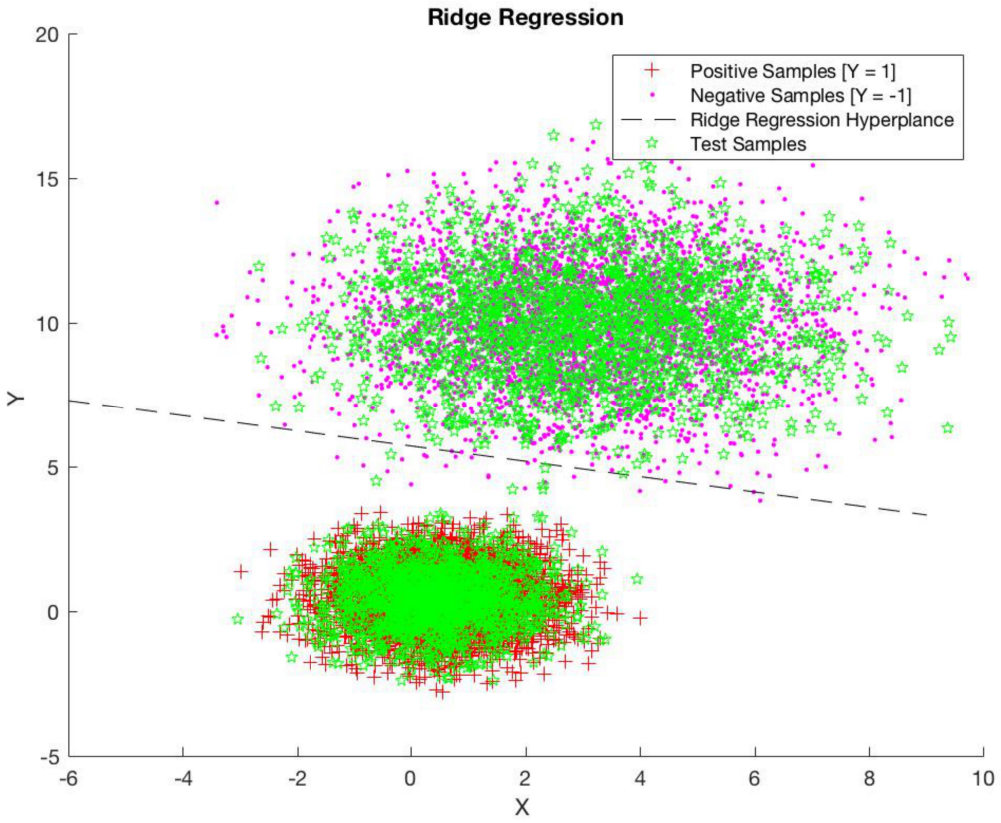
- Ridge Regression We train our model on the Xtrain and ytrain given in synthdata.mat and we get the w, using this w, we use the below code(train_rr function is same as in Question 2):

```
function [ ] = rr(Xtrain, ytrain, Xtest, ytest)

    [w,w_0 ]= train_rr(Xtrain, ytrain,1);
    err =  test_rr(Xtest, ytest, w, w_0);
```

```
    X_Pos = Xtrain(ytrain == 1, :);
    X_Neg = Xtrain(ytrain == -1, :);

    hold on
    plot(X_Pos(:,1),X_Pos(:,2),'+r')
    plot(X_Neg(:,1),X_Neg(:,2),'.b')
    plot_hyperplane(w, w_0);
    plot(Xtest(:,1),Xtest(:,2),'gp')

    % Adding titles
    title('Ridge Regression')
    xlabel('X Lables')
    ylabel('Y Labels')
    legend('Positive Samples [Y = 1]','Negative Samples [Y = -1]','Ridge Regression Hyperplance',
end
```

Accuracy: 99.798%

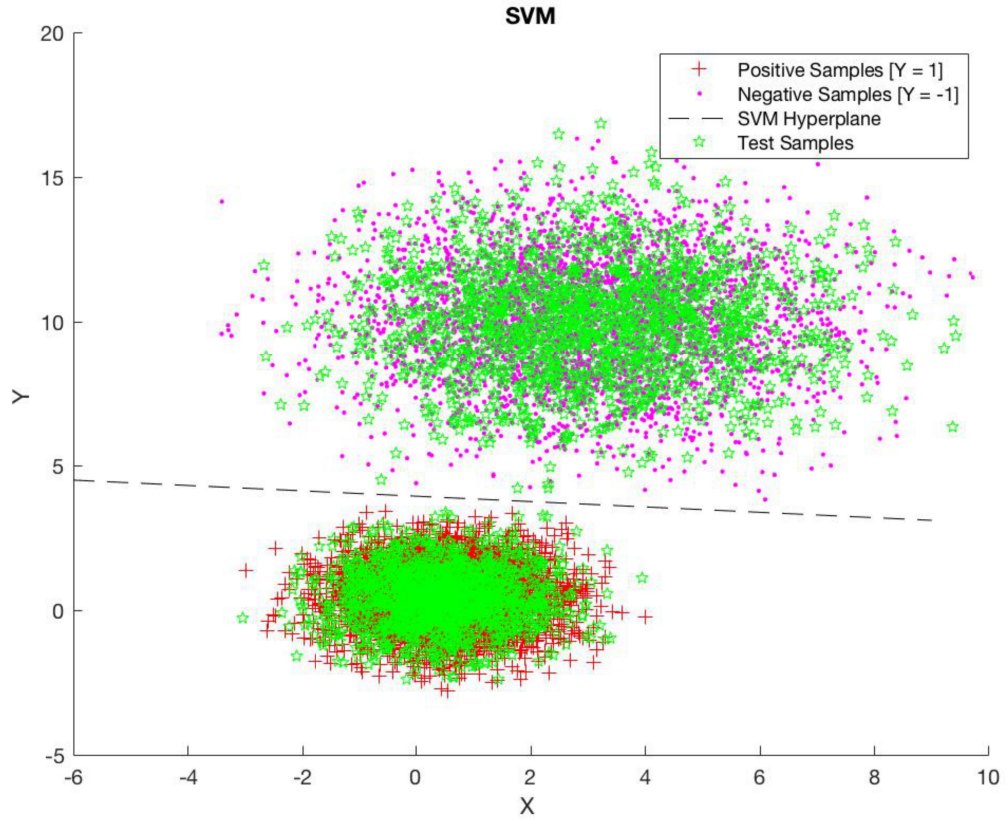
Below is the plot of the Ridge Regression hyperplane.



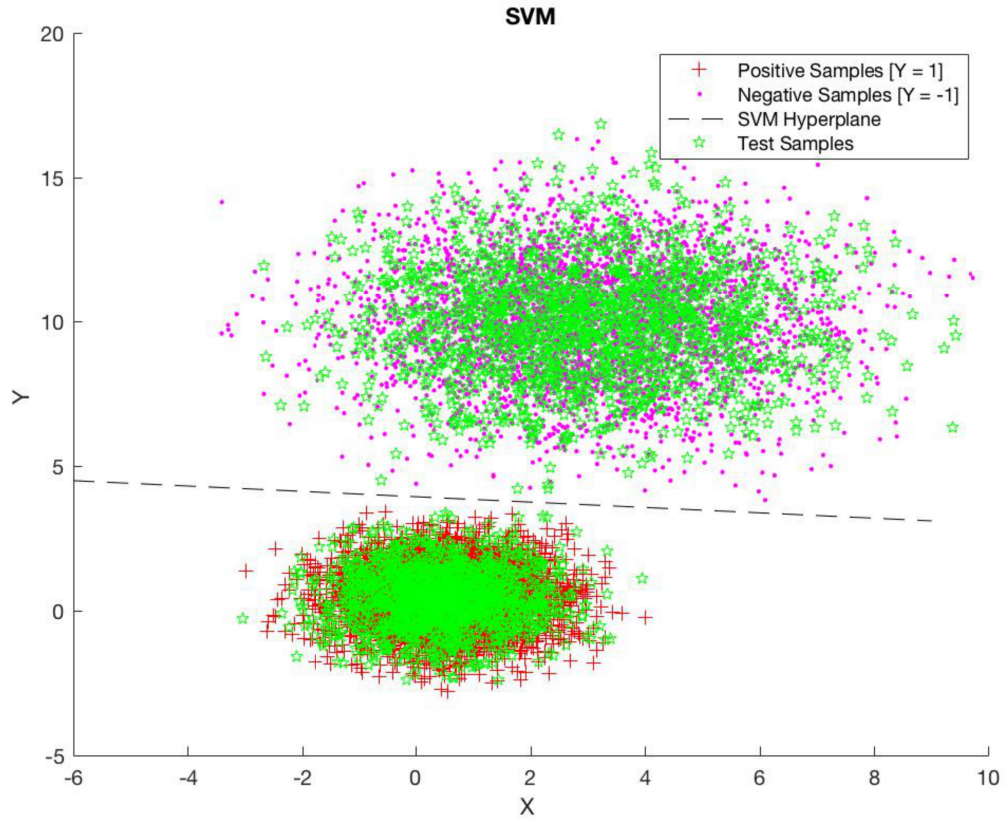
- SVM Below is the code to train and test using SVM. We use the $train_svm_primal$ function from Question 7 to calculate the weight vector. We first run the code using C = 0.01

```
function [ ] = svm(Xtrain, ytrain, Xtest, ytest, C)
    [w,w_0] = train_svm_primal(Xtrain, ytrain, C);
    a = accuracy(Xtest, ytest, w, w_0);
    X_Pos = Xtrain(ytrain == 1, :);
    X_Neg = Xtrain(ytrain == -1, :);

    hold on
    plot(X_Pos(:,1),X_Pos(:,2),'+r')
    plot(X_Neg(:,1),X_Neg(:,2),'.b')
    plot_hyperplane(w, w_0);
    plot(Xtest(:,1),Xtest(:,2),'gp')

    % Adding titles
    title('SVM')
    xlabel('X Lables')
    ylabel('Y Labels')
    legend('Positive Samples [Y = 1]','Negative Samples [Y = -1]','Perceptron Hyperplane', 'Test Sa

end
```

Accuracy: 100%
Objective Value: $3.199e^{-04}$
Below is the graph showing the SVM hyperplane with C = 0.01.



We now update C = 100 and run the above code again, we get the
Accuracy: 100%
Objective Value: $2.489e^{04}$



As observed, there is no change in the plot. This is because, even though we are using a Soft margin, since the our data is linearly separable, the contribution of misclassified samples is 0 and thus the any value of C will have no impact on our solution.

- Perceptron Below is the code to train and test using Perceptron. We use the $train_svm_primal$ function from Question 5 to calculate the weight vector.

```
function [ output_args ] = perceptron(Xtrain,ytrain, Xtest, ytest)
    [w,w_0] = train_perceptron(Xtrain, ytrain);
    acc = accuracy(Xtest, ytest, w, w_0);
    disp(acc);
    X_Pos = Xtrain(ytrain == 1, :);
    X_Neg = Xtrain(ytrain == -1, :);

    hold on
    plot(X_Pos(:,1),X_Pos(:,2),'+r')
    plot(X_Neg(:,1),X_Neg(:,2),'.b')
    plot_hyperplane(w, w_0);
    plot(Xtest(:,1),Xtest(:,2),'gp')
```

```
    % Adding titles
    title('Ridge Regression')
    xlabel('X Lables')
    ylabel('Y Labels')
    legend('Positive Samples [Y = 1]','Negative Samples [Y = -1]','Perceptron Hyperplane', 'Test Sa
end
```

Accuracy: 100%

Below is the plot for the perceptron