ECE 595: Homework 4

Rahul Deshmukh, Class ID: 58

(Spring 2019)

In collaboration with: Murali Krishnan & Anmol Singh Guram

Exercise 1

a Convergence of Logistic Regression:

(i) Given: The two datasets are linearly separable.

To Prove: The magnitude of slope and intercept parameters $\boldsymbol{w}$ and w_0 would tend to ∞

Proof: If the data is linearly separable then we would want the Loss function to be zero. The Loss function for Logistic regression is given by:

$$
\begin{aligned}
J(\boldsymbol{\theta}) &= -\sum_{j}(y_j log(h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)) + (1-y_j)log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_j))) \\
&= -\sum_{\boldsymbol{x}_j \in C_1} log(h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)) - \sum_{\boldsymbol{x}_j \in C_0} log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)) \\
\text{where } h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) &= \frac{1}{1+e^{\boldsymbol{\theta}^T\boldsymbol{x}_j}}
\end{aligned}
$$

We can observe that:

$$
\begin{aligned}
h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) &< 1 \text{ for finite value of } \boldsymbol{\theta} \\
\Rightarrow log(h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)) &< 0 \;\&\; log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_j)) < 0 \\
\Rightarrow J(\boldsymbol{\theta}) &> 0
\end{aligned}
$$

Also, we can say the following:

$$
\begin{aligned}
\boldsymbol{x}_j \in C_1 \lim_{\boldsymbol{\theta}\to\infty} h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) &= 1 \\
\boldsymbol{x}_j \in C_0 \lim_{\boldsymbol{\theta}\to\infty} h_{\boldsymbol{\theta}}(\boldsymbol{x}_j) &= 0 \\
\Rightarrow \lim_{\boldsymbol{\theta}\to\infty} J(\boldsymbol{\theta}) &= 0
\end{aligned}
$$

Therefore, we will attain the minimum zero loss only when $\boldsymbol{\theta} \to \infty$. This means that for any solution $\boldsymbol{\theta}_k$ we can obtain a better solution $\alpha\boldsymbol{\theta}_k$ where $\alpha > 0$ with a smaller loss value. This indicates that the logistic regression will suffer from nonconvergence in case of linearly separable data.

To Prove: Gradient descent iterates would not converge in a finite number of steps if we have the stopping criteria as $||\boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^{(k)}|| = 0$

Proof:

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \alpha_k (\sum_{n=1}^{N} (h_{\boldsymbol{\theta}^{(k)}}(\boldsymbol{x}_n) - y_n)\boldsymbol{x}_n)$$

$$\boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^{(k)} = -\alpha_k (\sum_{n=1}^{N} (h_{\boldsymbol{\theta}^{(k)}}(\boldsymbol{x}_n) - y_n)\boldsymbol{x}_n)$$

$$\text{if } ||\boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^{(k)}|| = 0 \Rightarrow h_{\boldsymbol{\theta}^{(k)}}(\boldsymbol{x}_n) = y_n$$

$$\Rightarrow \text{only when } \boldsymbol{\theta} \to \boldsymbol{\infty}$$

$$\Rightarrow k \to \infty$$

$$\text{ie never converges}$$

(ii) When we have $||\boldsymbol{w}|| < c_1$ and $|w_0| < c_2$ for some $c_1, c_2 > 0$, then we can say that $\boldsymbol{\theta}$ is no longer unbounded and cannot reach to ∞. Then these limits will be used as a termination criteria and thus we will have a converged solution.

Some other ways to counter the non-convergence issue can be to add a regularization term $(\lambda||\theta||^2)$ to the loss function and solve the unconstrained problem. **Or** To Rather minimize the function $||\theta||^2$ subject to the constraint $J(\boldsymbol{\theta}) < \epsilon$.

(iii) We don't face the issue of non-convergence of other linear classifiers when the data is linearly separable this is because for the other methods we can attain a zero loss for linearly separable data within finite number of iterations (which we have proved in class for several methods)

b Proof of convergence of online mode of perceptron algorithm:

Given: $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \alpha_k y_j \boldsymbol{x_j}$ with $\alpha_k = 1$for $j \in \mathcal{M}_k$

(i) To prove: The move from $\boldsymbol{w^{(k)}}$ to $\boldsymbol{w^{(k+1)}}$ is in the right direction i.e. decreases the misclassification error.
We need to show that $y_j(\boldsymbol{w^{(k+1)}})^T\boldsymbol{x_j} > y_j(\boldsymbol{w^{(k)}})^T\boldsymbol{x_j}$

Proof:

$$
\begin{aligned}
y_j(\boldsymbol{w}^{(k+1)})^T\boldsymbol{x}_j &= y_j(\boldsymbol{w}^{(k)} + \alpha_k y_j \boldsymbol{x}_j)^T\boldsymbol{x}_j \\
&= y_j(\boldsymbol{w}^{(k)})^T\boldsymbol{x}_j + y_j(\alpha_k y_j \boldsymbol{x}_j)^T\boldsymbol{x}_j \\
&= y_j(\boldsymbol{w}^{(k)})^T\boldsymbol{x}_j + (\alpha_k y_j^2 ||\boldsymbol{x}_j||^2) \\
&\geq y_j(\boldsymbol{w}^{(k)})^T\boldsymbol{x}_j \text{ ....as } \alpha_k y_j^2 ||\boldsymbol{x}_j||^2 \geq 0 \\
\Rightarrow y_j(\boldsymbol{w}^{(k+1)})^T\boldsymbol{x}_j &\geq y_j(\boldsymbol{w}^{(k)})^T\boldsymbol{x}_j \\
&\text{hence proved}
\end{aligned}
$$

(ii) Proof of convergence:

Given: $\rho = \min_j y_j(\boldsymbol{w^*})^T\boldsymbol{x}_j$

(1) To prove: $(\boldsymbol{w^{(k)}})^T\boldsymbol{w}^* \geq k\rho$ (Please note: this is a tighter bound than the homework prompt)

Proof:

$$
\begin{aligned}
(\boldsymbol{w^{(k)}})^T\boldsymbol{w}^* &= (\boldsymbol{w^{(k-1)}} + y_j\boldsymbol{x}_j)^T\boldsymbol{w}^* \\
&= (\boldsymbol{w^{(k-1)}})^T\boldsymbol{w}^* + (y_j\boldsymbol{x}_j)^T\boldsymbol{w}^* \\
&= (\boldsymbol{w^{(k-1)}})^T\boldsymbol{w}^* + y_j(\boldsymbol{w}^*)^T\boldsymbol{x}_j \\
&\text{ by definition: } y_j(\boldsymbol{w}^*)^T\boldsymbol{x}_j \geq \rho \\
&\geq (\boldsymbol{w^{(k-1)}})^T\boldsymbol{w}^* + \rho \\
&\text{By induction we can say:} \\
(\boldsymbol{w^{(k)}})^T\boldsymbol{w}^* &\geq (\boldsymbol{w^{(0)}})^T\boldsymbol{w}^* + k\rho \\
&\text{where } \boldsymbol{w^{(0)}} = \boldsymbol{0} \\
\Rightarrow (\boldsymbol{w^{(k)}})^T\boldsymbol{w}^* &\geq k\rho \\
&\text{hence proved}
\end{aligned}
$$

(2) Given: $R = max_j||\boldsymbol{x}_j||_2$, To prove: $||\boldsymbol{w^{(k)}}||_2^2 \leq k^2R^2$

Proof:

$$
\begin{aligned}
||\boldsymbol{w}^{\boldsymbol{(k)}}|| &= ||\boldsymbol{w}^{\boldsymbol{(k-1)}} + y_j\boldsymbol{x}_j|| \\
&\text{using Triangle inequality we get:} \\
&\leq ||\boldsymbol{w}^{\boldsymbol{(k-1)}}|| + ||y_j\boldsymbol{x}_j|| \\
&\leq ||\boldsymbol{w}^{\boldsymbol{(k-1)}}|| + ||\boldsymbol{x}_j|| \text{ .... as } y_j = \pm 1 \\
&\text{By induction we have:} \\
&\leq ||\boldsymbol{w}^{\boldsymbol{(0)}}|| + k||\boldsymbol{x}_j|| \\
&\text{where } \boldsymbol{w}^{\boldsymbol{(0)}} = \boldsymbol{0} \\
&\leq k||\boldsymbol{x}_j|| \\
&\leq kR \\
\Rightarrow ||\boldsymbol{w}^{\boldsymbol{(k)}}||^2 &\leq k^2R^2 \\
&\text{hence proved}
\end{aligned}
$$

(3) To Prove: $\frac{(\boldsymbol{w}^{\boldsymbol{(k)}})^T\boldsymbol{w}^*}{||\boldsymbol{w}^(k)||_2} \geq \frac{\rho}{R}$ and $k \leq \frac{R^2||\boldsymbol{w}^*||}{\rho^2}$ (Please note: this is a tighter bound than the homework prompt)

$$
\begin{aligned}
(\boldsymbol{w}^{\boldsymbol{(k)}})^T\boldsymbol{w}^* &\geq k\rho \text{ (proved in part (1))} \\
\boldsymbol{w}^{(k)} &\leq kR \text{ (proved in part (2))} \\
\Rightarrow \frac{(\boldsymbol{w}^{\boldsymbol{(k)}})^T\boldsymbol{w}^*}{||\boldsymbol{w}^(k)||_2} &\geq \frac{k\rho}{kR} = \frac{\rho}{R} \\
&\text{hence proved}
\end{aligned}
$$

Now let us try to prove that the sequence $\boldsymbol{w}^{(k)}$ converges to $\boldsymbol{w}^*$. Therefore we need to prove the following:

$$\exists\ \epsilon > 0 \,\boldsymbol{|}\, ||w^{(k)} - w^*|| < \epsilon$$

Proof:

$$\begin{aligned}
||\boldsymbol{w}^{(k)} - \boldsymbol{w}^*||^2 &= ||\boldsymbol{w}^{(k-1)} + \alpha_k y_j \boldsymbol{x}_j - \boldsymbol{w}^*||^2 \\
&= ||\boldsymbol{w}^{(k-1)} - \boldsymbol{w}^* + \alpha_k y_j \boldsymbol{x}_j||^2 \\
&= ||\boldsymbol{w}^{(k-1)} - \boldsymbol{w}^*||^2 + ||\alpha_k y_j \boldsymbol{x}_j||^2 + 2\alpha_k (\boldsymbol{w}^{(k-1)} - \boldsymbol{w}^*)^T y_j \boldsymbol{x}_j
\end{aligned}$$

we know that at k-1 iteration the misclassification error is greater that 0 ie $(\boldsymbol{w}^{(k-1)})^T y_j \boldsymbol{x}_j \leq 0$

$$\begin{aligned}
&\leq ||\boldsymbol{w}^{(k-1)} - \boldsymbol{w}^*||^2 + \alpha_k^2 ||y_j \boldsymbol{x}_j||^2 - 2\alpha_k (\boldsymbol{w}^*)^T y_j \boldsymbol{x}_j \\
&\leq ||\boldsymbol{w}^{(k-1)} - \boldsymbol{w}^*||^2 + \alpha_k(\alpha_k ||y_j \boldsymbol{x}_j||^2 - 2(\boldsymbol{w}^*)^T y_j \boldsymbol{x}_j)
\end{aligned}$$

if $\alpha_k ||y_j \boldsymbol{x}_j||^2 - 2(\boldsymbol{w}^*)^T y_j \boldsymbol{x}_j \leq 0$

$$\text{ie } \alpha_k \leq \frac{2(\boldsymbol{w}^*)^T y_j \boldsymbol{x}_j}{||y_j \boldsymbol{x}_j||^2}$$

$$\text{choosing } \alpha_k = \frac{(\boldsymbol{w}^*)^T y_j \boldsymbol{x}_j}{||y_j \boldsymbol{x}_j||^2} \text{ we get:}$$

$$||\boldsymbol{w}^{(k)} - \boldsymbol{w}^*||^2 \leq ||\boldsymbol{w}^{(k-1)} - \boldsymbol{w}^*||^2 - \frac{((\boldsymbol{w}^*)^T y_j \boldsymbol{x}_j)^2}{||y_j \boldsymbol{x}_j||^2}$$

By induction we get:

$$||\boldsymbol{w}^{(k)} - \boldsymbol{w}^*||^2 \leq ||\boldsymbol{w}^{(0)} - \boldsymbol{w}^*||^2 - k\frac{((\boldsymbol{w}^*)^T y_j \boldsymbol{x}_j)^2}{||y_j \boldsymbol{x}_j||^2}$$

where $\boldsymbol{w}^{(0)} = \mathbf{0}$

$$||\boldsymbol{w}^{(k)} - \boldsymbol{w}^*||^2 \leq ||\boldsymbol{w}^*||^2 - k\frac{((\boldsymbol{w}^*)^T y_j \boldsymbol{x}_j)^2}{||y_j \boldsymbol{x}_j||^2} = \epsilon^2$$

Therefore, The sequence the sequence $\boldsymbol{w}^{(k)}$ converges to $\boldsymbol{w}^*$ if $\epsilon^2 > 0$:

$$\Rightarrow ||\boldsymbol{w}^*||^2 - k\frac{((\boldsymbol{w}^*)^T y_j \boldsymbol{x}_j)^2}{||y_j \boldsymbol{x}_j||^2} > 0$$

$$||\boldsymbol{w}^*||^2 > k\frac{((\boldsymbol{w}^*)^T y_j \boldsymbol{x}_j)^2}{||y_j \boldsymbol{x}_j||^2}$$

$$k < \frac{||\boldsymbol{w}^*||^2 ||y_j \boldsymbol{x}_j||^2}{((\boldsymbol{w}^*)^T y_j \boldsymbol{x}_j)^2}$$

$$k < \frac{||\boldsymbol{w}^*||^2 ||\boldsymbol{x}_j||^2}{((\boldsymbol{w}^*)^T y_j \boldsymbol{x}_j)^2}$$

using earlier definitions $||\boldsymbol{x}_j|| \leq R$ and $(\boldsymbol{w}^*)^T y_j \boldsymbol{x}_j \geq \rho$

$$k \leq \frac{||\boldsymbol{w}^*||^2 R^2}{\rho^2}$$

hence proved

c Soft-Margin SVM:

$$\begin{gathered}\underset{\boldsymbol{\theta},\boldsymbol{\xi}}{\text{argmin}}\frac{1}{2}||\boldsymbol{w}||_2^2 + \frac{C}{2}||\boldsymbol{\xi}||_2^2 \\ \text{subject to } y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j) \geq 1 - \xi_j \\ \xi_j \geq 0, j = 1, ..., N\end{gathered}$$

We need to find the convex dual problem for it

(i) The constraint $\xi_j \geq 0$ can be removed without affecting the solution to the optimization problem, why? lets write ou the lagrangian of the function:

$$L(\theta, \lambda, \xi, \mu) = \frac{1}{2}||\boldsymbol{w}||_2^2 + \frac{C}{2}||\boldsymbol{\xi}||_2^2 + \sum_j(\lambda_j(1 - \xi_j - y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0)) + \mu_j(-\xi_j))$$

Form complementary slackness condition we have:

$$\begin{gathered}\mu_j(\xi_j) = 0 \\ \Rightarrow \mu_j = 0, \xi_j \geq 0 \\ \text{or } \mu_j \geq 0, \xi_j = 0\end{gathered}$$

Therefore the constraint $\xi_j \geq 0$ is always satisfied and using the above fact we can simplify the lagrangian to:

$$L(\theta, \lambda, \xi) = \frac{1}{2}||\boldsymbol{w}||_2^2 + \frac{C}{2}||\boldsymbol{\xi}||_2^2 + \sum_j(\lambda_j(1 - \xi_j - y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0)))$$

(ii) Finding the optimal solutions:

$$\begin{aligned}L(\theta, \lambda, \xi) &= \frac{1}{2}||\boldsymbol{w}||_2^2 + \frac{C}{2}||\boldsymbol{\xi}||_2^2 + \sum_j(\lambda_j(1 - \xi_j - y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0))) \\ \nabla_{\boldsymbol{w}}L = 0 &\Rightarrow \boldsymbol{w}^* - \sum_j \lambda_j y_j \boldsymbol{x}_j = 0 \\ &\Rightarrow \boldsymbol{w}^* = \sum_j \lambda_j y_j \boldsymbol{x}_j \\ \nabla_{w_0}L = 0 &\Rightarrow \sum_j \lambda_j y_j = 0 \\ \nabla_{\boldsymbol{\xi}}L = 0 &\Rightarrow C\boldsymbol{\xi} - \boldsymbol{\lambda} = 0 \\ &\Rightarrow C\boldsymbol{\xi} = \boldsymbol{\lambda} \Rightarrow \boldsymbol{\xi}^* = \frac{\boldsymbol{\lambda}}{C}\end{aligned}$$

hence proved

(iii) Finding the convex dual of the problem:

$$
\begin{aligned}
&\underset{\boldsymbol{w},w_0}{\text{minimize}}\, \mathcal{L}(\boldsymbol{w}, w_0, \lambda) \\
=&\underset{\boldsymbol{w},w_0}{\text{minimize}} \left\{ \frac{1}{2}||\boldsymbol{w}||_2^2 + \frac{C}{2}||\boldsymbol{\xi}||_2^2 + \max_{\lambda \geq 0} \left\{ \sum_j (\lambda_j (1 - \xi_j - y_j(\boldsymbol{w}^T \boldsymbol{x}_j + w_0))) \right\} \right\} \\
=&\underset{\boldsymbol{w},w_0}{\text{minimize}} \{ \underset{\lambda}{\text{maximize}}\, \mathcal{L}(\boldsymbol{w}, w_0, \lambda) \} \\
=&\underset{\lambda}{\text{maximize}} \{ \underset{\boldsymbol{w},w_0}{\text{minimize}}\, \mathcal{L}(\boldsymbol{w}, w_0, \lambda) \} \\
=&\underset{\lambda}{\text{maximize}} \{ \mathcal{L}(\boldsymbol{w}^*, w_0^*, \lambda) \} \\
=&\underset{\lambda}{\text{maximize}} \{ \frac{1}{2}||\boldsymbol{w}^*||_2^2 + \frac{C}{2}||\boldsymbol{\xi}^*||_2^2 + \sum_j (\lambda_j (1 - \xi_j^* - y_j(\boldsymbol{w}^{*T} \boldsymbol{x}_j + w_0^*))) \Big\} \\
=&\underset{\lambda}{\text{maximize}} \Big\{ + \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j + \frac{C}{2} \sum_k \frac{\lambda_k}{C}^2 + \sum_j \lambda_j \\
&- \sum_k \frac{\lambda_k^2}{C} - \sum_i \sum_j \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j - w_0^* \sum_j (\lambda_j y_j) \Big\} \\
=&\underset{\lambda}{\text{maximize}} \Big\{ - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j - \frac{1}{2} \sum_k \frac{\lambda_k^2}{C} + \sum_j \lambda_j \Big\} \\
&\text{subject to } \sum_j \lambda_j y_j = 0
\end{aligned}
$$

hence proved

d Hard- Margin SVM:

$$\underset{\boldsymbol{\theta}}{\text{argmin}} \frac{1}{2}||\boldsymbol{w}||_2^2$$
$$\text{st } y_j g_{\boldsymbol{\theta}}(\boldsymbol{x_j}) \geq 0, j = 1, ..., N$$

(i) To prove: if the constraint is changed to $y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j) \geq \gamma$ such that $\gamma \geq 0$ then the problem remains the same.

Proof:

$$\underset{\boldsymbol{\theta}}{\text{argmin}} \frac{1}{2}||\boldsymbol{w}||_2^2$$
$$y_j g_{\boldsymbol{\theta}}(\boldsymbol{x}_j) \geq \gamma$$
$$\Rightarrow y_j(\boldsymbol{w}^T\boldsymbol{x}_j + w_0)/\gamma \geq 1$$
$$\Rightarrow y_j(\frac{\boldsymbol{w}^T}{\gamma}\boldsymbol{x}_j + \frac{w_0}{\gamma}) \geq 1$$

We can scale the objective function by a constant $\frac{1}{\gamma^2}$ without changing the problem

$$\Rightarrow \underset{\boldsymbol{\theta}}{\text{argmin}} \frac{1}{2}\frac{||\boldsymbol{w}||_2^2}{\gamma^2}$$
$$y_j(\frac{\boldsymbol{w}^T}{\gamma}\boldsymbol{x}_j + \frac{w_0}{\gamma}) \geq 1$$
$$\Rightarrow \underset{\hat{\boldsymbol{\theta}}=\frac{\boldsymbol{\theta}}{\gamma}}{\text{argmin}} \frac{1}{2}||\hat{\boldsymbol{w}}||_2^2$$
$$y_j(\hat{\boldsymbol{w}}^T\boldsymbol{x}_j + \hat{w}_0) \geq 1$$

This is the same problem as earlier. Hence proved.

(ii) To prove : SVM for two points $\boldsymbol{x}_1 \in C_+$ and $\boldsymbol{x}_2 \in C_-$ is solvable.

Proof:

The Dual Problem is:

$$\max_{\lambda \geq 0} \Big\{ \sum_{j=1}^{2} \lambda_j - \frac{1}{2} \sum_{i=1}^{2} \sum_{j=1}^{2} (\lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j) \Big\}$$

$$\text{st} \sum_{j=1}^{2} \lambda_j y_j = 0$$

$$\Rightarrow \max_{\lambda \geq 0} \Big\{ \sum_{j=1}^{2} \lambda_j - \frac{1}{2} \sum_{i=1}^{2} \sum_{j=1}^{2} (\lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j) \Big\}$$

$$\text{st} \sum_{j=1}^{2} \lambda_j y_j = 0$$

$$\Rightarrow \lambda_1 - \lambda_2 = 0$$

$$\Rightarrow \lambda_1 = \lambda_2 = \lambda$$

$$\Rightarrow \max_{\lambda \geq 0} \Big\{ 2\lambda - \frac{\lambda^2}{2} (\boldsymbol{x}_1^T \boldsymbol{x}_1 + \boldsymbol{x}_2^T \boldsymbol{x}_2 - 2\boldsymbol{x}_1^T \boldsymbol{x}_2) \Big\}$$

taking derivative and equating to zero we get

$$2 - \lambda(||\boldsymbol{x}_1 - \boldsymbol{x}_2||^2) = 0$$

$$\lambda = \frac{2}{||\boldsymbol{x}_1 - \boldsymbol{x}_2||^2} \geq 0$$

$$\boldsymbol{w}^* = \sum_{j=1}^{2} \lambda_j y_j \boldsymbol{x}_j$$

$$\Rightarrow \boldsymbol{w}^* = \lambda(\boldsymbol{x}_1 - \boldsymbol{x}_2)$$

$$= \frac{2(\boldsymbol{x}_1 - \boldsymbol{x}_2)}{||\boldsymbol{x}_1 - \boldsymbol{x}_2||^2}$$

$$w_0 = -\frac{\boldsymbol{w}^{*T}(\boldsymbol{x}_1 + \boldsymbol{x}_2)}{2}$$

$$\Rightarrow w_0 = -\frac{1}{2}\lambda(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T(\boldsymbol{x}_1 + \boldsymbol{x}_2)$$

$$w_0 = -\frac{(||\boldsymbol{x}_1||^2 - ||\boldsymbol{x}_2||^2)}{||\boldsymbol{x}_1 - \boldsymbol{x}_2||^2}$$

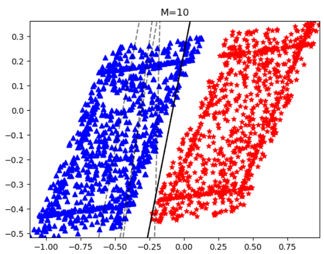
Hence proved.

Exercise 2

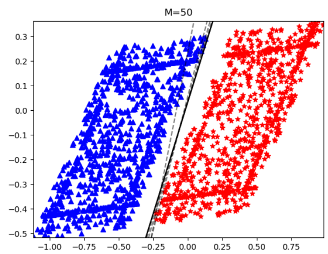
(a) Please find the code at page 15

(b) Please find the code at page 15

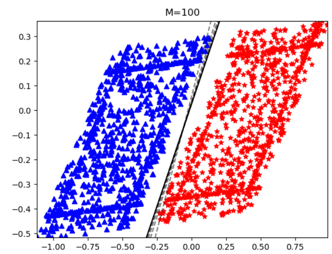
The Learning rate was chosen as 10^{-1}



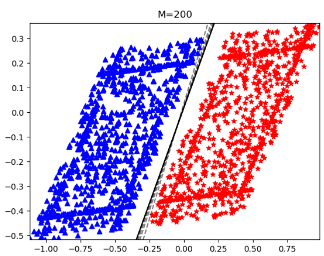
(a) M=10



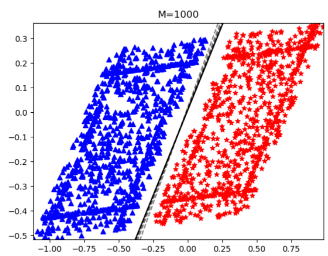
(b) M=50



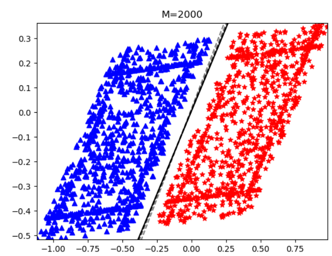
(c) M=100



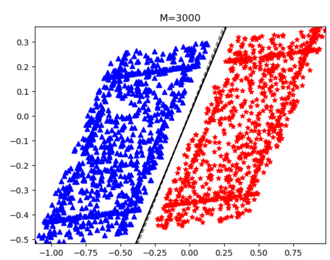
(d) M=200



(e) M=1000

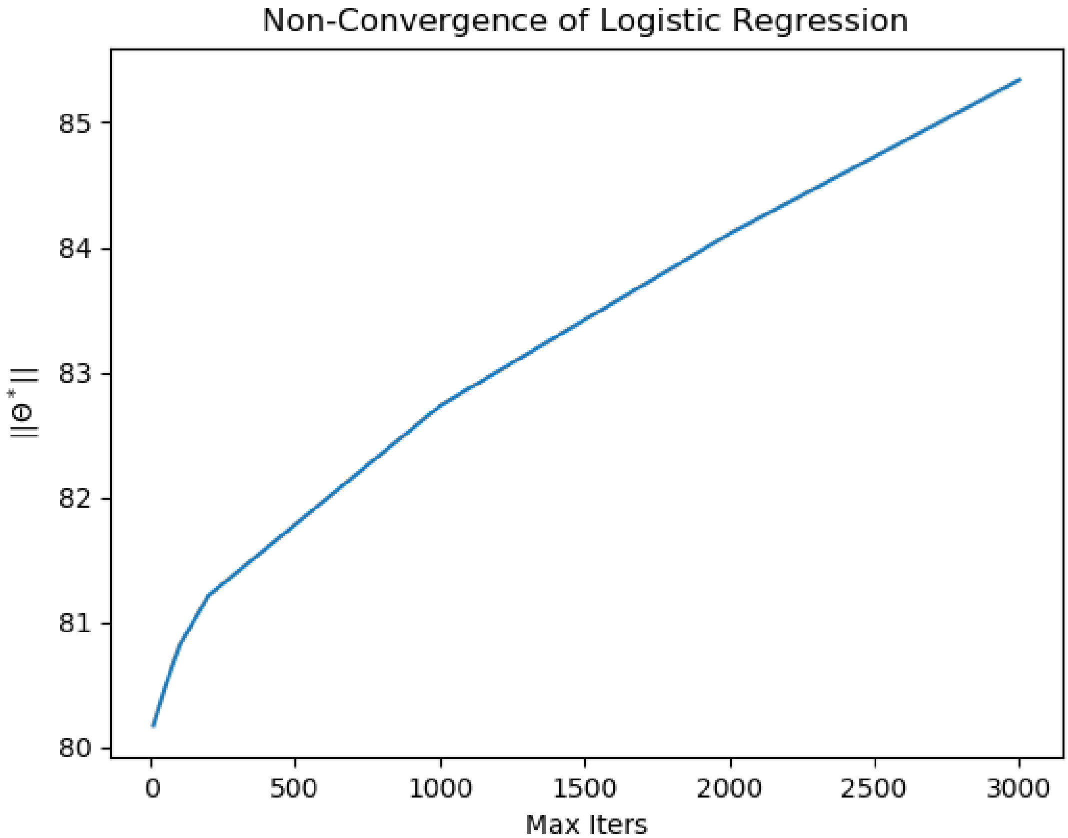


(f) M=2000



(g) M=3000

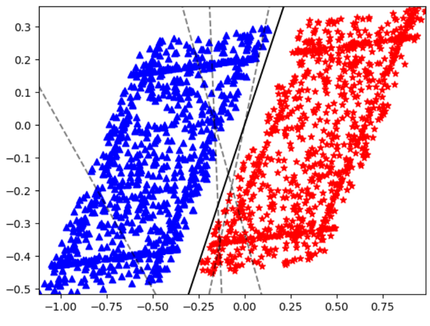
As can be observed, the decision boundary improves as the maximum number of iterations increases however, I observed that after attaining a good separating hyperplane there is no change in the decision boundary as we further increase the number of iterations.



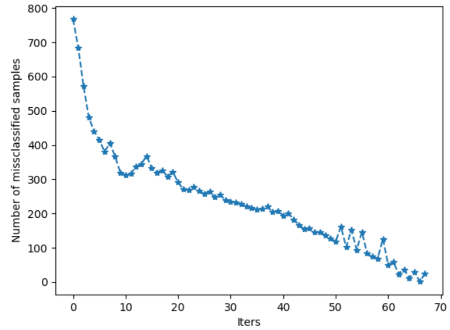
We can observe from the above plot that the magnitude of theta $||\boldsymbol{\theta}||$ monotonically increases as the number of iterations increases this further confirms our proof in exercise 1(a).

b Perceptron: Please find the code at page 15

(i) Online Mode

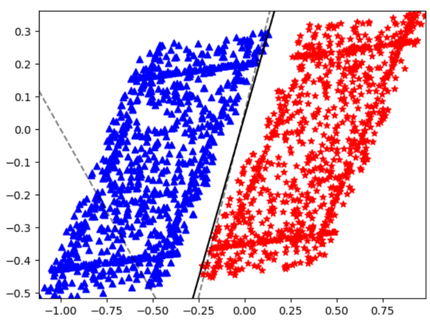


(h) decision boundary plot

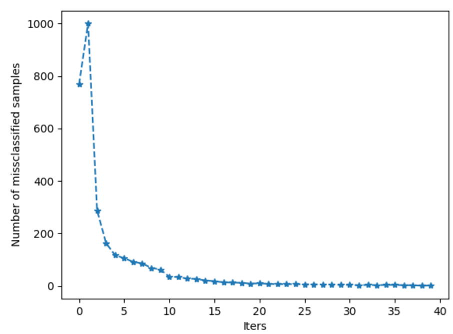


(i) Convergence

(i) Batch Mode



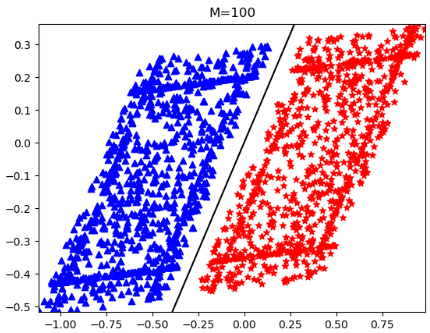
(j) decision boundary plot



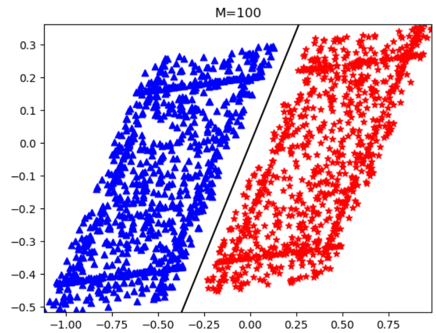
(k) Convergence

From the above plot for convergence of both the modes we observe that convergence is faster for Batch Mode. However, in terms of cost of computation online mode is cheaper than batch mode.

c SVM, Please find the code at page 15

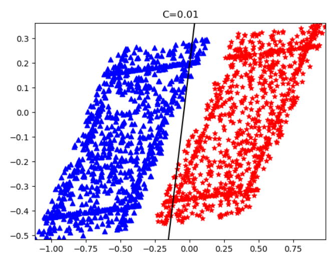


(l) Hard

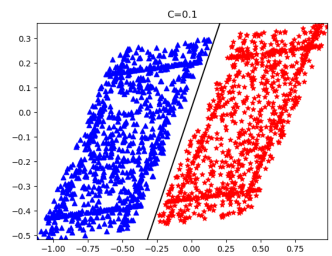


(m) soft

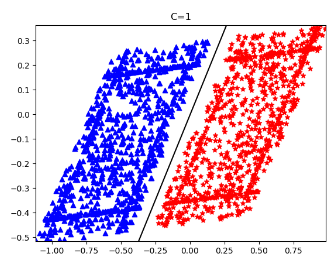
We can observe that the decision boundaries for the two cases are nearly. More insights can be gained from calculating the margin for the two cases. We can observe from the above plots that



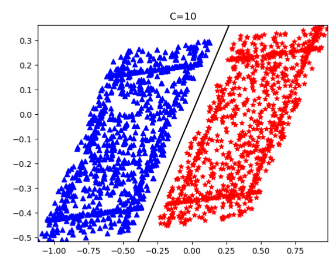
(n) C=0.01



(o) C=0.1



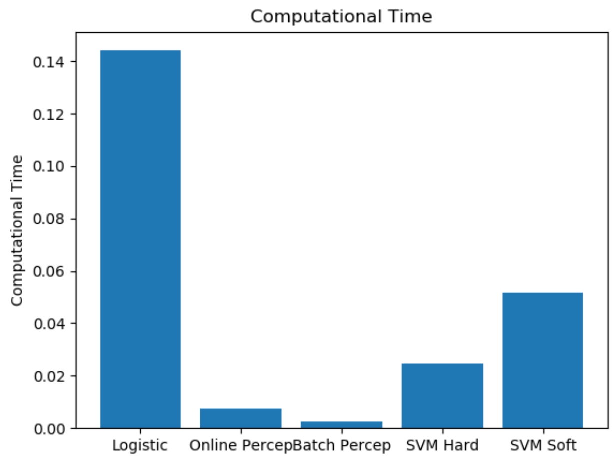
(p) C=1



(q) C=10

for the soft margin as C increases the decision boundary starts resembling the hard SVM decision boundary. Also, as the data is linearly separable with a good margin, therefore using a hard margin SVM is better suited.

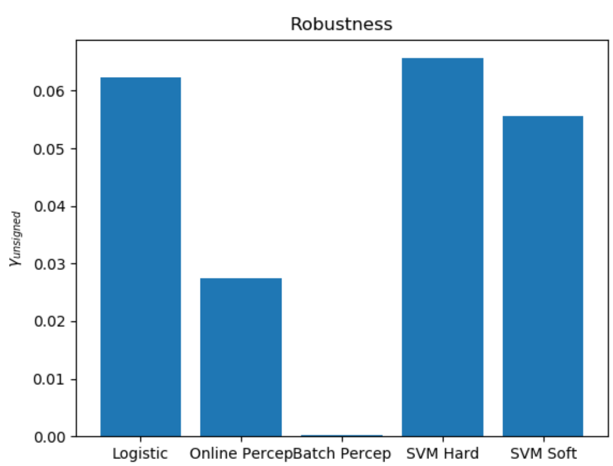
d Classification Error: For all of the classifiers we saw(from earlier plots) that we were able to get correct classification.



(r) Computational Time

Computational Time: We can observe that Logistic regression takes the maximum time to get a good solution as it requires more number of iterations. Also, The perceptron methods are cheaper than SVM because in perceptron we are not doing additional work of maximizing the margin.

Effect of Learning rate: It was observed that a smaller learning rate required more iterations to converge and a faster learning rate can give erroneous results.



(s) $\gamma_{unsigned}$

From the above plot we can observe that Perceptron Methods are the least robust and can be a poor method to classify a test dataset. Of all the classifiers SVM hard seems to be the most robust. This is because the dataset is linearly seperable and has no outliers, had there been outliers then SVM soft would have been the best option.

Main Code for Exercise 2

Listing 1: Source Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 18 16:47:50 2019
@author: rahul
"""
# import libraries
import numpy as np , csv
from functions import *
import matplotlib.pyplot as plt
#%% Read Data
#(a)
read_path = '../data/'
label_filename = 'hw04_labels.csv'
sample_filename = 'hw04_sample_vectors.csv'

samples = []
with open(read_path+sample_filename) as csvfile:
    readcsv = csv.reader(csvfile,delimiter=',')
    for row in readcsv:
        vals = list(map(float,row))
        samples.append(vals)
samples_copy = np.array(samples)
samples = np.ones((np.shape(samples_copy)[0],np.shape(samples_copy)[1]+1))
samples[:,:-1]=samples_copy
samples = samples.T

labels = []
with open(read_path+label_filename) as csvfile:
    readcsv = csv.reader(csvfile,delimiter=',')
    for row in readcsv:
        vals = float(row[0])
        labels.append(vals)
labels = np.array(labels)

scaled_labels = 2*labels-1 #percep labels in range -1 to +1
# Declare constants

rate = 0.1
#%%
#(a) Logistic regression based classification
log_star_store=[]
Ms= [10,50,100,200,1000,2000,3000]

for M in Ms:
    [log_theta_star,log_theta_store] = logistic(samples,labels,rate,M)
    log_star_store.append(np.linalg.norm(log_theta_star))
    freq = int(0.2*M)
#     plotdata(samples,labels,log_theta_store,freq,M)
#     plotdata_single(samples,labels,log_theta_star,M)
    log_gamma = min(scaled_labels*(log_theta_star.T@samples)/np.linalg.norm(
        log_theta_star))

#plt.figure()
#plt.plot(Ms,log_star_store)
```

```
#plt.xlabel('Max Iters')
#plt.ylabel('||$\Theta^*$||')
#plt.title('Non-Convergence of Logistic Regression')
#plt.show()

#%%(b) Perceptron Online mode
M = 100 # max number of iters
freq = int(0.2*M)
scaled_labels = 2*labels-1 #percep labels in range -1 to +1
##(i)
[ol_percp_theta_star,ol_percp_theta_store]= perceptron(samples,scaled_labels,rate,M,convplot=0)
#plotdata(samples,labels,ol_percp_theta_store,freq,[M])
ol_gamma = min((scaled_labels*(ol_percp_theta_star.T@samples))/np.linalg.norm(ol_percp_theta_star))


##(ii)
[bt_percp_theta_star,bt_percp_theta_store]= perceptron(samples,scaled_labels,rate,M,convplot=0,online=False)
#plotdata(samples,labels,bt_percp_theta_store,freq)
bt_gamma = min(scaled_labels*(bt_percp_theta_star.T@samples)/np.linalg.norm(bt_percp_theta_star))

#%%(c) SVM
#(i) Hard Margin
svm_hard_theta_star = SVM_hard(samples,scaled_labels)
#plotdata_single(samples,labels,svm_hard_theta_star,M)
hard_gamma = min((scaled_labels*(np.array(svm_hard_theta_star).T@samples))/np.linalg.norm(svm_hard_theta_star))

##(ii) Soft Margin
C = 1
#for C in [10**(-2),10**(-1),1,10]:
svm_soft_theta_star = SVM_soft_L1(samples,scaled_labels,C)
#plotdata_single(samples,labels,svm_soft_theta_star,C)
soft_gamma = min((scaled_labels*(np.array(svm_soft_theta_star).T@samples))/np.linalg.norm(svm_hard_theta_star))

#plt.figure()
#plt.bar(['Logistic','Online Percep','Batch Percep','SVM Hard','SVM Soft'],\
#        [log_gamma,ol_gamma,bt_gamma,hard_gamma,soft_gamma])
#plt.ylabel('$\gamma_{unsigned}$')
#plt.title('Robustness')
#plt.show()
```

Functions for Exercise 2

Listing 2: Functions

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 18 17:07:13 2019

@author: rahul
"""
# import libraries
```

```
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cvx
import time
#%% SVM
# SVM Soft Margin
def SVM_soft_L1(samples,labels,C):
    """
    Input:
    sample: np array in the format [[x1],[x2],....,[xn]] xi as col vectors
    labels: np array [y1,y2,....,yn]
    """
    t_in = time.time()
    dim,Nsamples = samples.shape
    theta = cvx.Variable(dim) # declaring dimension of variable
    lam = 1/C
    Jump_term = cvx.max_elemwise(0,1-cvx.mul_elemwise(labels,(theta.T@samples).T))
    Jump_term = cvx.sum_entries(Jump_term)
    obj_expr = Jump_term +(lam/2)*(cvx.sum_squares(theta[:-1]))
    obj = cvx.Minimize(obj_expr)
    prob = cvx.Problem(obj)
    prob.solve(solver = cvx.ECOS)
    theta_star = theta.value
    theta_star = theta_star.tolist()
    theta_star= [x[0] for x in theta_star]
    print('SVM soft time '+str(time.time() - t_in)+'\n')
    return(theta_star)

# SVM Hard Margin
def SVM_hard(samples,labels):
    """
    Input:
    sample: np array in the format [[x1],[x2],....,[xn]] xi as col vectors
    labels: np array [y1,y2,....,yn]
    """
    t_in = time.time()
    dim,Nsamples = samples.shape
    theta = cvx.Variable(dim) # declaring dimension of variable
    obj = cvx.Minimize(cvx.sum_squares(theta[:-1]))
    temp = cvx.mul_elemwise(labels,(theta.T@samples).T)
    const = [ temp >= 1 ]
    prob = cvx.Problem(obj,const)
    prob.solve(solver = cvx.ECOS)
    theta_star = theta.value
    #format change
    theta_star = theta_star.tolist()
    theta_star= [x[0] for x in theta_star]
    print('SVM hard time '+str(time.time() - t_in)+'\n')
    return(theta_star)


#%% Perceptron Method
def  percept_batch_tangent(theta,samples,true_labels):
    """
    Input:
        theta: decision boundary
        sample: np array in the format [[x1],[x2],....,[xn]] xi as col vectors
        labels: np array [y1,y2,....,yn]
    Out: J: Jacobian
```

```python
    """
    # predict samples labels using theta
    gx = theta.T@samples
    ygx = np.multiply(true_labels,gx)
    miss_idx = np.where(ygx<0)[0]
    N_miss = miss_idx.size
    all_miss_labels = true_labels[miss_idx]
    all_miss_x = samples[:,miss_idx]
    J = np.sum(all_miss_labels*all_miss_x,axis=1)
    return(J,N_miss)

def percept_online_tangent(theta,samples,true_labels):
    """
    Input:
        theta: decision boundary
        sample: np array in the format [[x1],[x2],....,[xn]] xi as col vectors
        labels: np array [y1,y2,....,yn]
    Out: J: Jacobian
    """
    # predict samples labels using theta
    gx = theta.T@samples
    ygx = np.multiply(true_labels,gx)
    miss_idx = np.where(ygx<0)[0]
    N_miss = miss_idx.size
    if N_miss>0:
        picked_idx = miss_idx[np.random.permutation(N_miss)]
        picked_idx = picked_idx[0]
        J = true_labels[picked_idx]*samples[:,picked_idx]
    else:
        J = np.zeros_like(samples[:,0]) #just to pass some value
    return(J,N_miss)

def perceptron(samples,labels,rate,Max_iter,convplot,online = True):
    """
    Input:
        samples: np array in the format [[x1],[x2],....,[xn]] xi as col vectors
        labels: np array [y1,y2,....,yn]
        rate: learning rate or the step length
        Max_iter: for the gradient descent
        online: if online mode then True(default) else False
    """
    t_in = time.time()
    xdim,Nsamples= samples.shape
    theta_k = np.ones(xdim) # initial guess
    theta_store = []
    N_store = []
    theta_store.append(np.copy(theta_k))
    if online:
        for k in range(Max_iter):
            grad_k,N_miss = percept_online_tangent(theta_k,samples,labels)
            if N_miss == 0:
                break
            theta_k += rate*grad_k
            theta_store.append(np.copy(theta_k))
            N_store.append(N_miss)
        print('online percep time '+str(time.time() - t_in)+'\n')
    else:
        for k in range(Max_iter):
            grad_k,N_miss = percept_batch_tangent(theta_k,samples,labels)
```

```
            if N_miss == 0:
                break
            theta_k += rate*grad_k
            theta_store.append(np.copy(theta_k))
            N_store.append(N_miss)
        print('batch percep time '+str(time.time() - t_in)+'\n')
    if convplot==1:
        plt.plot(np.arange(len(N_store)),N_store,'*--')
        plt.xlabel('Iters')
        plt.ylabel('Number of missclassified samples')
        plt.show()

    return(theta_k,theta_store)

#%% function for plotting data
def plotdata_single(samples,labels,theta_star,*argv):
    """
    Input:
        sample: np array in the format [[x1],[x2],....,[xn]] xi as col vectors
        labels: np array [y1,y2,....,yn]
        theta_star: decision boundary params
        N: frequency of plots for decision boundary
    """

    plt.figure()
    #plot the training set
    for i in range(np.shape(samples)[1]):
        if labels[i] == 1:
            plt.scatter(samples[0,i],samples[1,i],c='red',marker='*')
        else:
            plt.scatter(samples[0,i],samples[1,i],c='blue',marker='^')

    # plot the decision boundary
    x_min = min(samples[0,:])
    x_max = max(samples[0,:])
    y_min = min(samples[1,:])
    y_max = max(samples[1,:])
    x_line = np.linspace(x_min,x_max,10)
    y_line = -(theta_star[0]*x_line +theta_star[2])/theta_star[1]
    plt.plot(x_line,y_line,'k-')
    plt.xlim(x_min,x_max)
    plt.ylim(y_min,y_max)
    if argv!=None:
        plt.title('C='+str(argv[0]))
    plt.show()
    return()

def plotdata(samples,labels,theta_store,N,*argv):
    """
    Input:
        sample: np array in the format [[x1],[x2],....,[xn]] xi as col vectors
        labels: np array [y1,y2,....,yn]
        theta_store: decision boundary params
        N: frequency of plots for decision boundary
    """

    plt.figure()
    #plot the training set
    for i in range(np.shape(samples)[1]):
```

```
        if labels[i] == 1:
            plt.scatter(samples[0,i],samples[1,i],c='red',marker='*')
        else:
            plt.scatter(samples[0,i],samples[1,i],c='blue',marker='^')

    # plot the decision boundary
    x_min = min(samples[0,:])
    x_max = max(samples[0,:])
    y_min = min(samples[1,:])
    y_max = max(samples[1,:])
    x_line = np.linspace(x_min,x_max,10)
    count=0
    while count<len(theta_store):
        theta_k = theta_store[count]
        y_line = -(theta_k[0]*x_line +theta_k[2])/theta_k[1]
        plt.plot(x_line,y_line,'k--',alpha=0.5)
        count+=N
    theta_k = theta_store[-1]
    y_line = -(theta_k[0]*x_line +theta_k[2])/theta_k[1]
    plt.plot(x_line,y_line,'k-')
    plt.xlim(x_min,x_max)
    plt.ylim(y_min,y_max)
    if argv!=None:
        plt.title('M='+str(argv[0]))
    plt.show()
    return()

#%% Logistic regression function
# tangent for logistic function
def logistic_tangent(theta,samples,labels):
    """
    Input:
        theta: decision boundary
        sample: np array in the format [[x1],[x2],....,[xn]] xi as col vectors
        labels: np array [y1,y2,....,yn]
    Out: J: Jacobian
    """
    thetaTx  = theta.T@samples
    h_theta_x = 1/(1+np.exp(-thetaTx))
    temp = (h_theta_x - labels)*samples
    J  = np.sum(temp,axis=1)
    return(J)
# main logistic function
def logistic(samples,labels,rate,Max_iter):
    """
    Input:
        samples: np array in the format [[x1],[x2],....,[xn]] xi as col vectors
        labels: np array [y1,y2,....,yn]
        rate: learninig rate or the step length
        Max_iter: for the gradient descent
    """
    t_in = time.time()
    xdim,Nsamples= samples.shape
    theta_k = np.zeros(xdim) # initial guess
    theta_store = []
    theta_store.append(np.copy(theta_k))
    for k in range(Max_iter):
        theta_k -= rate*logistic_tangent(theta_k,samples,labels)
        theta_store.append(np.copy(theta_k))
```

```
    print('SVM logistic time '+str(time.time() - t_in)+'\n')
    return(theta_k,theta_store)
```