Introduction to ML (CS771), Autumn 2020
Indian Institute of Technology Kanpur
Homework Assignment Number 2

Student Name: Deeksha Arora
Roll Number: 20111017
Date: November 27, 2020

QUESTION 1

Loss function for Logistic Regression model with no regularization is given by:

$$\mathcal{L}(\boldsymbol{w}) = -\sum_{n=1}^{N}\left(y_n \boldsymbol{w}^\top \boldsymbol{x}_n - \log\left(1+\exp\left(\boldsymbol{w}^\top \boldsymbol{x}_n\right)\right)\right) \tag{1}$$

$\mathcal{L}(\boldsymbol{w})$ is a convex function in $\boldsymbol{w}$, therefore we can find global minima for it.
Taking the derivative of $\mathcal{L}(\boldsymbol{w})$ with respect to $\boldsymbol{w}$, we get:

$$\begin{aligned}\frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w}} &= \frac{\partial}{\partial \boldsymbol{w}}\left[-\sum_{n=1}^{N}\left(y_n \boldsymbol{w}^\top \boldsymbol{x}_n - \log\left(1+\exp\left(\boldsymbol{w}^\top \boldsymbol{x}_n\right)\right)\right)\right] \\ &= -\sum_{n=1}^{N}\left(y_n \boldsymbol{x}_n - \frac{\exp\left(\boldsymbol{w}^\top \boldsymbol{x}_n\right)}{\left(1+\exp\left(\boldsymbol{w}^\top \boldsymbol{x}_n\right)\right)}\boldsymbol{x}_n\right)\end{aligned} \tag{2}$$

We can't obtain closed form expression for $\widehat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$ i.e by equating $\frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w}}$ to zero.
Therefore, using second order based methods (Ex: Newton's method) to get the solution.
The update equation for iteration $\mathbf{t}$ is given by:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \mathbf{H}^{(t)^{-1}} \boldsymbol{g}^{(t)} \tag{3}$$

Let $\mu_n = \frac{\exp\left(\boldsymbol{w}^\top \boldsymbol{x}_n\right)}{\left(1+\exp\left(\boldsymbol{w}^\top \boldsymbol{x}_n\right)\right)}$, therefore the gradient is given by:

$$\boldsymbol{g} = \frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w}} = -\sum_{n=1}^{N}\left(y_n - \mu_n\right)\boldsymbol{x}_n = \mathbf{X}^\top(\boldsymbol{\mu} - \boldsymbol{y}) \tag{4}$$

Differentiating equation (4) with respect to $\boldsymbol{w}$, we get:

$$\begin{aligned}\frac{\partial^2 \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w} \partial \boldsymbol{w}^\top} &= \frac{\partial \boldsymbol{g}^\top}{\partial \boldsymbol{w}} \\ &= -\frac{\partial}{\partial \boldsymbol{w}}\sum_{n=1}^{N}\left(y_n - \mu_n\right)\mathbf{x}_n^\top \\ &= \sum_{n=1}^{N}\frac{\partial \mu_n}{\partial \boldsymbol{w}}\boldsymbol{x}_n^\top\end{aligned} \tag{5}$$

We know that, $\mu_n = \frac{\exp\left(\boldsymbol{w}^\top \boldsymbol{x}_n\right)}{\left(1+\exp\left(\boldsymbol{w}^\top \boldsymbol{x}_n\right)\right)}$. Differentiating μ_n with respect to $\boldsymbol{w}$ gives:

$$\frac{\partial \mu_n}{\partial \boldsymbol{w}} = \frac{\partial}{\partial \boldsymbol{w}}\left(\frac{\exp\left(w^\top x_n\right)}{1+\exp\left(w^\top x_n\right)}\right) = \mu_n\left(1-\mu_n\right)x_n \tag{6}$$

Putting value of $\frac{\partial \mu_n}{\partial w}$ from equation (6) in equation (5) gives:

$$\frac{\partial^2 \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w} \partial \boldsymbol{w}^\top} = \sum_{n=1}^{N} \mu_n \left(1 - \mu_n\right) \mathbf{x}_n \mathbf{x}_n^\top \tag{7}$$

Therefore, the Hessian matrix is given by:

$$\mathbf{H} = \sum_{n=1}^{N} \mu_n \left(1 - \mu_n\right) \mathbf{x}_n \mathbf{x}_n^\top = \mathbf{X}^\top \mathbf{G} \mathbf{X} \tag{8}$$

where $\boldsymbol{G}$ is a diagonal matrix with its n^{th} diagonal element as $\mu_n(1-\mu_n)$.
The second-order optimization based update for t^{th} iteration is given by:

$$\begin{aligned}
\boldsymbol{w}^{(t+1)} &= \boldsymbol{w}^{(t)} - \mathbf{H}^{(t)^{-1}} \boldsymbol{g}^{(t)} \\
&= \boldsymbol{w}^{(\boldsymbol{t})} - \left(\mathbf{X}^\top \mathbf{G}^{(t)} \mathbf{X}\right)^{-1} \mathbf{X}^\top \left(\boldsymbol{\mu}^{(t)} - \boldsymbol{y}\right) \\
&= \boldsymbol{w}^{(\boldsymbol{t})} + \left(\mathbf{X}^\top \mathbf{G}^{(t)} \mathbf{X}\right)^{-1} \mathbf{X}^\top \left(\boldsymbol{y} - \boldsymbol{\mu}^{(\boldsymbol{t})}\right) \\
&= \left(\mathbf{X}^\top \mathbf{G}^{(t)} \mathbf{X}\right)^{-1} \left[\left(\mathbf{X}^\top \mathbf{G}^{(t)} \mathbf{X}\right) \boldsymbol{w}^{(\boldsymbol{t})} + \mathbf{X}^\top \left(\boldsymbol{y} - \boldsymbol{\mu}^{(\boldsymbol{t})}\right)\right] \\
&= \left(\mathbf{X}^\top \mathbf{G}^{(t)} \mathbf{X}\right)^{-1} \mathbf{X}^\top \left[\mathbf{G}^{(t)} \mathbf{X} \boldsymbol{w}^{(\boldsymbol{t})} + \boldsymbol{y} - \boldsymbol{\mu}^{(\boldsymbol{t})}\right] \\
&= \left(\mathbf{X}^\top \mathbf{G}^{(t)} \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{G}^{(t)} \left[\mathbf{X} \boldsymbol{w}^{(\boldsymbol{t})} + \mathbf{G}^{(\mathbf{t})^{-1}} \left(\boldsymbol{y} - \boldsymbol{\mu}^{(\boldsymbol{t})}\right)\right]
\end{aligned} \tag{9}$$

The objective function for Iteratively Reweighted Least Squares (IRLS) problem is given by:

$$\boldsymbol{w}^{(t+1)} = \arg\min_{w} \sum_{n=1}^{N} \gamma_n^{(t)} \left(\hat{y}_n^{(t)} - \boldsymbol{w}^\top \boldsymbol{x}_n\right)^2 \tag{10}$$

where $\gamma_n^{(t)}$ denotes the importance of the n^{th} training example. Differentiating equation (10) w.r.t. $\boldsymbol{w}$ gives:

$$\begin{aligned}
&\tfrac{\partial}{\partial(\boldsymbol{w})} \left(\boldsymbol{w}^{(t+1)}\right) = 0 \\
\implies &\textstyle\sum_{n=1}^{N} \gamma_n^{(t)} \left(\hat{y}_n^{(t)} - \boldsymbol{w}^\top \boldsymbol{x}_n\right) (-\boldsymbol{x}_n) = 0
\end{aligned} \tag{11}$$

$$\begin{aligned}
\implies \boldsymbol{w}^{(t+1)} &= \left(\sum_{n=1}^{N} \gamma_n^{(t)} \boldsymbol{x}_n \boldsymbol{x}_n^\top\right)^{-1} \left(\sum_{n=1}^{N} \gamma_n^{(t)} \hat{y}_n^{(t)} \boldsymbol{x}_n\right) \\
&= \left(\boldsymbol{X}^\top \gamma^{(t)} \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top \gamma^{(t)} \hat{\boldsymbol{y}}^{(t)}
\end{aligned} \tag{12}$$

On comparing equation (9) and (12), we can say that:

$$\boldsymbol{\gamma}^{(t)} = \boldsymbol{G}^{(t)} \quad \text{and,} \quad \hat{\boldsymbol{y}}^{(t)} = \mathbf{X} \boldsymbol{w}^{(\boldsymbol{t})} + \boldsymbol{\gamma}^{(\boldsymbol{t})^{-1}} \left(\boldsymbol{y} - \boldsymbol{\mu}^{(\boldsymbol{t})}\right)$$

Therefore, $\gamma_n^{(t)} = \mu_n(1-\mu_n)$
i.e. $\gamma_n^{(t)}$ is the n^{th} diagonal element of diagonal matrix $G^{(t)}$.
The expressions for $\gamma_n^{(t)}$ and $\hat{\boldsymbol{y}}^{(t)}$ makes sense because if a large error is made on $\boldsymbol{x}_n$ then $(\boldsymbol{y}_n - \boldsymbol{\mu}_n^t)$ will be large and thus the contribution of $\boldsymbol{x}_n$ to $\boldsymbol{w}^{(t)}$ is large. Thus the value of $\hat{\boldsymbol{y}}_n^{(t)}$ and $\gamma_n^{(t)}$ changes in each iteration based on the amount of misprediction and number of times mistake is made on $\boldsymbol{x}_n$.

Introduction to ML (CS771), Autumn 2020
Indian Institute of Technology Kanpur
Homework Assignment Number 2

Student Name: Deeksha Arora
Roll Number: 20111017
Date: November 27, 2020

QUESTION 2

The prediction rule of perceptron algorithm is given by:

$$\hat{y} = \text{sign}\left(\boldsymbol{w}^\top \boldsymbol{x}\right) \tag{13}$$

We know that the kernel function $k(a_1, a_2)$ finds the dot product similarity between the two inputs a_1 and a_2. Therefore, to derive a kernelized version of perceptron algorithm, we need to get rid of the weight vector $\boldsymbol{w}$ in the prediction rule. We know that, the weight vector learned using Perceptron algorithm is given by:

$$\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i \tag{14}$$

where α_i is total number of mistakes made by the perceptron algorithm on example (x_i, y_i). Using this result, the prediction rule of perceptron algorithm can be rewritten as:

$$\hat{y} = \text{sign} \sum_{i=1}^{N} \alpha_i y_i \left(\boldsymbol{x}_i^\top \cdot \boldsymbol{x}\right) \tag{15}$$

By plugging the equations (14) and (15) into the perceptron training loop, we will get *dual perceptron algorithm*. To get the kernelized perceptron algorithm, we can replace the dot product of inputs x_m and x_n with $K(x_m, x_n)$, where K is an arbitrary kernel function which gives the effect of the feature map ϕ.
Therefore, the new prediction rule becomes:

$$\hat{y} = \text{sign} \sum_{i=1}^{N} \alpha_i y_i K(x_i, x), \text{where } K(x_i, x) = \phi\left(\boldsymbol{x}_i\right) \cdot \phi(\boldsymbol{x}) \tag{16}$$

The Kernelized Perceptron algorithm:

1. Initialize $\boldsymbol{\alpha} = \mathbf{0}$, where $\boldsymbol{\alpha}$ is a vector of length N and b=0
2. for iter= $1, 2...T$ (where T is some fixed number of iterations or until some stopping criteria is met)
3. for each training example (x_j, y_j) do,
4. $\hat{y}_j \leftarrow \text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i \left(\phi\left(\boldsymbol{x}_i\right) \cdot \phi(\boldsymbol{x}_j)\right)\right)$
5. If $\hat{y}_j \neq y_j$ then,
6. $\alpha_j \leftarrow \alpha_j + 1$ incrementing the mistake counter

Therefore,

- **Initialization:** $\boldsymbol{\alpha} = \mathbf{0}$ and b=0
- **Mistake Condition:** $\hat{y}_j \neq y_j$ i.e. $\text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i \left(\phi\left(\boldsymbol{x}_i\right) \cdot \phi(\boldsymbol{x}_j)\right)\right) \neq y_j$
- **Update Equation:** $\alpha_j \leftarrow \alpha_j + 1$

Introduction to ML (CS771), Autumn 2020
Indian Institute of Technology Kanpur
Homework Assignment Number 2

Student Name: Deeksha Arora
Roll Number: 20111017
Date: November 27, 2020

QUESTION 3

Cost Sensitive Support Vector Machine:
Consider the training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in R^D, y_i \in \{+1, -1\}, I_+ = \{i : y_i = +1\}$ and $I_- = \{i : y_i = -1\}$.
The primal formulation of this cost sensitive SVM problem is:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{\|\boldsymbol{w}\|^2}{2} + \sum_{n=1}^{N} C_{y_n}\xi_n \tag{17}$$

subject to the constraints $y_n\left(\boldsymbol{w}^T\boldsymbol{x}_n + b\right) \geq 1 - \xi_n$ and $\xi_n \geq 0, \forall n$
C_{+1} and C_{-1} are the costs of misclassifying positive examples and misclassifying negative examples, respectively. This constrained optimization can be solved using Lagrange's method. Introducing Lagrange's multipliers in equation (17) gives:

$$\begin{aligned}\max_{\boldsymbol{\alpha}\geq 0,\boldsymbol{\beta}\geq 0} \min_{\boldsymbol{w},b,\xi} L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) =& \frac{1}{2}\|\boldsymbol{w}\|^2 + C_{+1}\sum_{i\in I_+}\xi_i + C_{-1}\sum_{i\in I_-}\xi_i \\ &+ \sum_{i=1}^{N}\alpha_i\left[1 - \xi_i - y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i + b\right)\right] + \sum_{i=1}^{N}\beta_i\left[-\xi_i\right]\end{aligned} \tag{18}$$

where $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_N]$ and $\boldsymbol{\beta} = [\beta_1, \beta_2, \ldots, \beta_N]$ are vector of Lagrange multipliers.
To get the dual problem we need to eliminate the primal variables $\boldsymbol{w}, b, \boldsymbol{\xi}$. To do so, take partial derivative of $L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ with respect to $\boldsymbol{w}, b$ and $\boldsymbol{\xi}$:
Taking partial derivative w.r.t. $\boldsymbol{w}$:

$$\begin{aligned}\frac{\partial L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \mathbf{w}} &= 0 \\ \mathbf{w} + \sum_{i=1}^{N}\alpha_i\left[-y_i\mathbf{x}_i\right] &= 0 \\ \implies \mathbf{w} &= \sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i\end{aligned} \tag{19}$$

Taking partial derivative w.r.t. b :

$$\begin{aligned}\frac{\partial L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial b} &= 0 \\ \sum_{i=1}^{N}\alpha_i\left(-y_i\right) &= 0 \\ \implies \sum_{i=1}^{N}\alpha_i y_i &= 0\end{aligned} \tag{20}$$

Taking partial derivative w.r.t. $\boldsymbol{\xi}$:

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \xi_i} = 0$$

$$\implies C_{+1} - \alpha_i - \beta_i = 0, \quad i \in I_+ \quad \text{and} \quad C_{-1} - \alpha_i - \beta_i = 0, \quad i \in I_- \tag{21}$$

Using $b_i \geq 0,\ C_{+1} - \alpha_i - \beta_i = 0,\ i \in I_+$ and $C_{-1} - \alpha_i - \beta_i = 0,\ i \in I_-$, we get $0 \leq \alpha_i \leq C_{+1}, i \in I_+$ and $0 \leq \alpha_i \leq C_{-1},\ i \in I_-$

Using equations (19),(20) and (21), we can rewrite equation (18) as :

$$\begin{aligned}
\max L(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2} \left\| \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \right\|^2 + C_{+1} \sum_{i \in I_+} \xi_i + C_{-1} \sum_{i \in I_-} \xi_i \\
+ \sum_{i=1}^{N} \alpha_i \left[1 - \xi_i - y_i \left(\left(\sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \right)^T \mathbf{x}_i + b \right) \right] \\
+ \sum_{i \in I_+} (C_{+1} - \alpha_i) [-\xi_i] + \sum_{i \in I_-} (C_{-1} - \alpha_i) [-\xi_i] \\
\implies \max L(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \alpha_i \left[y_i \left(\left(\sum_{j=1}^{N} \alpha_j y_j \mathbf{x}_j \right)^T \mathbf{x}_i \right) \right]
\end{aligned} \tag{22}$$

This equation is independent of the dual variable β_i. Therefore, the dual optimization problem is :

$$\max_{\alpha} L_D(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \tag{23}$$

$$\begin{aligned}
\text{such that, } 0 \leq \alpha_i \leq C_{+1}, i \in I_+ \\
0 \leq \alpha_i \leq C_{-1}, i \in I_- \\
\sum_{i=1}^{N} \alpha_i y_i = 0
\end{aligned}$$

The standard SVM problem minimizes the probability of error, assuming that all **misclassifications have the same cost** and the classes of dataset are balanced. This results in cost insensitive decision boundary. However, some applications of Machine Learning where Standard SVM fails are:

- Applications where certain errors are much more costly than others. For example, fraud detection or predicting whether a patient has cancer or not.
- When class-imbalanced datasets are involved i.e. when examples from different classes appear with substantially different probability. The imbalanced dataset skews the model towards the minority class and thus degrades the performance of the model.

The Cost-sensitive support vector machine overcomes the weakness of standard SVM by assigning different cost to misclassification of positive and negative examples. It also reduces the average misclassification costs on class-imbalanced data by assigning higher cost to minority class and lower cost to majority which reduces the skewness of decision boundary.
So, this is the difference between the C (Cost) variable of standard SVM dual problem and cost-sensitive SVM dual problem.

Introduction to ML (CS771), Autumn 2020
Indian Institute of Technology Kanpur
Homework Assignment Number 2

Student Name: Deeksha Arora
Roll Number: 20111017
Date: November 27, 2020

QUESTION 4

SGD for K-means Objective :
In online K-means clustering, in each iteration, we randomly select a data point and assign it to one of k clusters such that the loss incurred is the squared distance between the new point and the closest center and then the next point is selected. In each iteration, the cluster means are updated using one randomly sampled point instead of the entire dataset. The goal is to minimize the K-means objective function:

$$\mathcal{L} = \sum_{i=1}^{N} \sum_{k=1}^{K} z_{nk} \left\| \boldsymbol{x}_n - \boldsymbol{\mu}_k \right\|^2 \tag{24}$$

Stochastic K-means Algorithm
Consider randomly sampled N datapoints $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \ldots, \boldsymbol{x}_k$ and K cluster means are $\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_k$. Let number of points in cluster k denoted by n_k be, $n_k = 0, \forall k \in \{1, 2, 3, \ldots, k\}$

1. **Initialization**: Randomly initialize k cluster means $\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_k$ and initialize number of points in a cluster to 0 i.e. $n_k = 0$, $\forall k \in \{1, 2, 3, \ldots, k\}$
2. Randomly select a data point $\boldsymbol{x}_n$ and perform following steps:
   2.1 Determine the closest cluster center $\boldsymbol{\mu}_k$ to $\boldsymbol{x}_n$
   2.2 Assign $\boldsymbol{x}_n$ to cluster C_k and update the number of points in the cluster: $n_k = n_k + 1$
   2.3 Update the cluster center: $\boldsymbol{\mu}_k := \boldsymbol{\mu}_k + \frac{1}{n_k}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)$
3. If all the data points are assigned to clusters then stop, else go to step 2.

Elaborating the steps of Stochastic K-Means: Consider a random data point $\boldsymbol{x}_n$. Therefore the loss function for cluster assignment of data point $\boldsymbol{x}_n$ will be:

$$l_n(\boldsymbol{x}_n, \boldsymbol{z}_n, \boldsymbol{\mu}) = \sum_{k=1}^{K} z_{nk} \left\| \boldsymbol{x}_n - \boldsymbol{\mu}_k \right\|^2 \quad \text{derived from eq(24)} \tag{25}$$

where $\boldsymbol{z}_n$ is a K-dim one-hot vector such that $z_{nk} = 1$ if $\boldsymbol{x}_n$ is assigned to cluster k else $z_{nk} = 0$. Only one entry of $\boldsymbol{z}_n$ can be 1.

Step 2.1 and 2.2- Assigning x_n "greedily" to the "best" cluster : To find the value of $\boldsymbol{z}_{nk}$ (0 or 1), compute the distance of $\boldsymbol{x}_n$ from every cluster mean and assign x_n to nearest cluster. Therefore,

$$\boldsymbol{z}_n = \underset{k \in [1,K]}{argmin} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2$$

So loss for the cluster assignment of $\boldsymbol{x}_n$ is,

$$\begin{aligned} l_n(\boldsymbol{x}_n, \boldsymbol{z}_n, \boldsymbol{\mu}) &= \sum_{k=1}^{N} z_{nk} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2 \\ &= \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2 \end{aligned}$$

Therefore, the gradient $\boldsymbol{g}_n$ of loss function corresponding to $\boldsymbol{x}_n$ is,

$$\begin{aligned} \boldsymbol{g}_n &= 2(\boldsymbol{\mu}_k - \boldsymbol{x}_n) \\ &\quad or, \\ \boldsymbol{g}_n^t &= 2(\boldsymbol{\mu}_k^t - \boldsymbol{x}_n) \end{aligned} \tag{26}$$

PStep 2.3- Updating the cluster means using SGD : The equations for updation of mean using SGD are:

$$\begin{aligned} \boldsymbol{\mu}_k^{t+1} &= \boldsymbol{\mu}_k^t - \eta_t \boldsymbol{g}_n^t \\ \boldsymbol{\mu}_k^{t+1} &= \boldsymbol{\mu}_k^t - 2\eta_t(\boldsymbol{\mu}_k^t - \boldsymbol{x}_n) \quad \text{(from eq(26))} \end{aligned} \tag{27}$$

This update equation makes sense because if we add data point x_n to a cluster then this datapoint should have contribution in the mean and the update equation very well depicts this. So, the new mean is computed over updated cluster as :

$$\boldsymbol{\mu}_k^{t+1} = \frac{n_k^t \boldsymbol{\mu}_k^t + \boldsymbol{x}_n}{n_k^t + 1} \quad \text{and} \quad n_k^{t+1} = n_k^t + 1 \tag{28}$$

where n_k^t denotes number of points in cluster k in t^{th} iteration.
Therefore, from equation (27) and (28), the step size η can be derived as:

$$\eta t = \frac{1}{2(n_k^t + 1)} \tag{29}$$

Replacing value of η from equation (29) in equation (27), gives the update equation for mean in t^{th} iteration,

$$\boldsymbol{\mu}_k^{t+1} = \boldsymbol{\mu}_k^t - \frac{1}{n_k^t + 1}(\boldsymbol{\mu}_k^t - \boldsymbol{x}_n) \tag{30}$$

This stepsize depicts that when there are less points in the cluster then more weight is given to the newly added point and when there are large number of points in the cluster then the newly added point gets less weight (importance).

Introduction to ML (CS771), Autumn 2020
Indian Institute of Technology Kanpur
Homework Assignment Number 2

Student Name: Deeksha Arora
Roll Number: 20111017
Date: November 27, 2020

QUESTION 5

The standard K-means can only detect linearly separable clusters. To detect non-convex shaped clusters we use Kernel K-means. The procedure of Kernel K-means is:

- Using the kernel function, map the data points in the input space to a higher dimensional feature space.
- Now, perform K-means on the mapped feature space.

In kernelized K-means algorithm, we replace the Euclidean distances by their kernelized versions. Consider D dimensional data $\{\boldsymbol{x}_n\}_{n=1}^N$, therefore, the Euclidean similarity in kernelized K-means can be written as: $\left\|\phi\left(x_n\right)-\mu^{(\mathrm{k})}\right\|^2$

where $\mu^1, ...\mu^k$ denotes K-many cluster means and $\phi(\boldsymbol{x})$ is the feature mapping corresponding to input $\boldsymbol{x}$.
Using the **representor theorem**, we can say that the mean of a set of data is, almost by definition, in the span of that data. Therefore, as long as we initialize the means in the span of the data, we are guaranteed to have the means in the span of the data. Using this information we can write the mean as an expansion of the data: $\boldsymbol{\mu}^{(\mathrm{k})}=\sum_i \alpha_i^{(\mathrm{k})}\phi\left(x_i\right)$, for some parameters $\alpha_i^{(\mathrm{k})}$. Since there are N data points and K clusters, therefore we will have $N \times K$ such parameters. Thus, Euclidean similarity in kernelized K-means can be written as:

$$\begin{aligned}\left\|\phi\left(x_n\right)-\mu^{(\mathrm{k})}\right\|^2 &= \left\|\phi\left(x_n\right)-\sum_i \alpha_i^{(\mathrm{k})}\phi\left(x_i\right)\right\|^2 \\ &= \left\|\phi\left(x_n\right)\right\|^2+\left\|\sum_i \alpha_i^{(\mathrm{k})}\phi\left(x_i\right)\right\|^2-2\phi\left(x_n\right)\cdot\left[\sum_i \alpha_i^{(\mathrm{k})}\phi\left(x_i\right)\right] \\ &= \sum_i\sum_j \alpha_i^{(\mathrm{k})}\alpha_j^{(\mathrm{k})}\phi\left(x_i\right)\cdot\phi\left(x_j\right)-2\sum_i \alpha_i^{(\mathrm{k})}\phi\left(x_i\right)\cdot\phi\left(x_n\right)+\text{const} \qquad (31)\end{aligned}$$

Even if ϕ is an infinite dimensional feature map, we can implement the kernel K-means algorithm in practice. The feature map ϕ doesn't need to be stored or computed for data $\{\boldsymbol{x}_n\}_{n=1}^N$ or the cluster means $\{\boldsymbol{\mu}_k\}_{k=1}^K$, because the computations depend only on the kernel evaluations.

THE KERNELIZED K-MEANS ALGORITHM :

1. INITIALIZATION :
For data $\{\boldsymbol{x}_n\}_{n=1}^N$, randomly initialize K-many cluster means $\mu^1, ...\mu^k$.

2. CLUSTER ASSIGNMENT :
2.1 For each data point x_n and every cluster C_k find $\left\|\phi\left(x_n\right)-\mu^{(\mathrm{k})}\right\|^2$ using equation (31).

2.2 For each data point x_n, set cluster label z_n as:

$$\begin{aligned} z_n &= \arg\min_k \left\| \phi(x_n) - \mu^{(\mathrm{k})} \right\|^2 \\ &= \arg\min_k \sum_m \sum_{m'} \alpha_m^{(\mathrm{k})} \alpha_{m'}^{(\mathrm{k})} \phi(x_m) \cdot \phi(x_{m'}) - 2 \sum_m \alpha_m^{(\mathrm{k})} \phi(x_m) \cdot \phi(x_n) + \text{const} \end{aligned} \tag{32}$$

In this equation, the dot products $\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})$ can be replaced by Kernel entries $K(x, y)$.

$$z_n = \arg\min_k \sum_m \sum_{m'} \alpha_m^{(\mathrm{k})} \alpha_{m'}^{(\mathrm{k})} K(\boldsymbol{x_m}, \boldsymbol{x_{m'}}) - 2 \sum_m \alpha_m^{(\mathrm{k})} K(\boldsymbol{x_m}, \boldsymbol{x_n}) + \text{const} \tag{33}$$

3. MEAN COMPUTATION :
For each cluster k, update $\mu^{(\mathrm{k})} = \frac{1}{N_k} \sum_{n:z_n=k} \phi(x_n)$, where N_k is the number of training examples with $z_n = k$.

4. If not converged yet, then go to step 2 otherwise stop and return the final clusters $C_1, ...C_k$.

How are clusters stored in kernel K-means versus how they are stored in standard K-means?
In standard K-means the clusters means are computed and stored using the formula:

$$\frac{\sum_{n=1}^{N} z_{nk}(\boldsymbol{x}_n)}{\sum_{n=1}^{N} z_{nk}}$$

But in kernel K-means the cluster are in terms in infinte dimension feature map ϕ and it is not possible to store them. However, the beauty of this algorithm is that we don't need to store or compute the cluster means beacuse all the computions involving means are dependent only on kernel evaluations as described in equation(31).

Time Complexity :
Let N be the number of training examples (in D-dimensional space), K is the number of clusters, I is the number of iterations and T is the time to calculate the distance between two objects. Time complexities will be:

- **For K-means:** Time complexity for each iteration will be $\mathcal{O}(KNDT)$, therefore for I iterations the time complexity will be $\mathcal{O}(IKNDT)$.
- **For Kernel K-means:** To compute the time complexity of Kernel K-means algorithm, we need to find the cost of computation of equation (33). In equation (33), in each iteration the 1^{st} term is fixed for a cluster m' so it computed only once and stored. The 2^{nd} term is calculated once per datapoint and costs O(N) each time, so it takes $\mathrm{O}\left(N^2\right)$ per iteration. Since our data is D-dimensional, therefore the computation of kernel matrix takes $\mathrm{O}\left(N^2 D\right)$ time. Thus, the total time complexity of the algorithm is : $\mathrm{O}\left(N^2\right) + \mathrm{O}\left(N^2 D\right) = \mathrm{O}\left(N^2 D\right)$. Considering K clusters and I iterations, the total time complexity of the Kernelized K-means algorithm is $\mathcal{O}(N^2 KDI)$.

Therefore, we can say that the Kernelized K-means algorithm is $\mathcal{O}(N^2 T)$ costlier than the standard K-means algorithm.

Introduction to ML (CS771), Autumn 2020
Indian Institute of Technology Kanpur
Homework Assignment Number 2

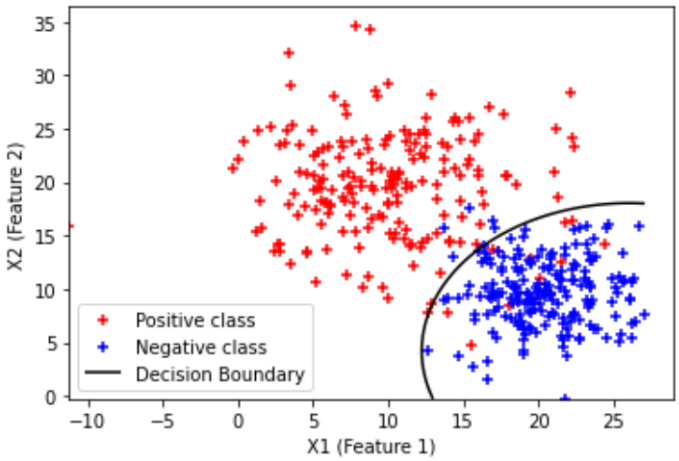
Student Name: Deeksha Arora
Roll Number: 20111017
Date: November 27, 2020

QUESTION
6

1 Using Generative classification

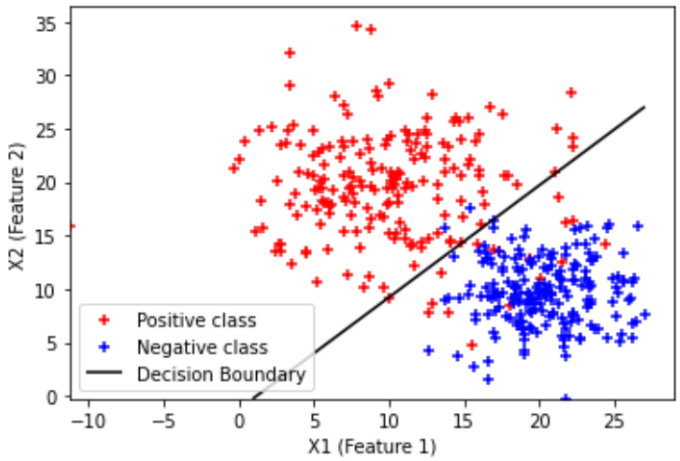
1.1 With different covariance matrices on binclass.txt

When different covariance matrix is used for both positive and negative classes then we get a non-linear decision boundary.



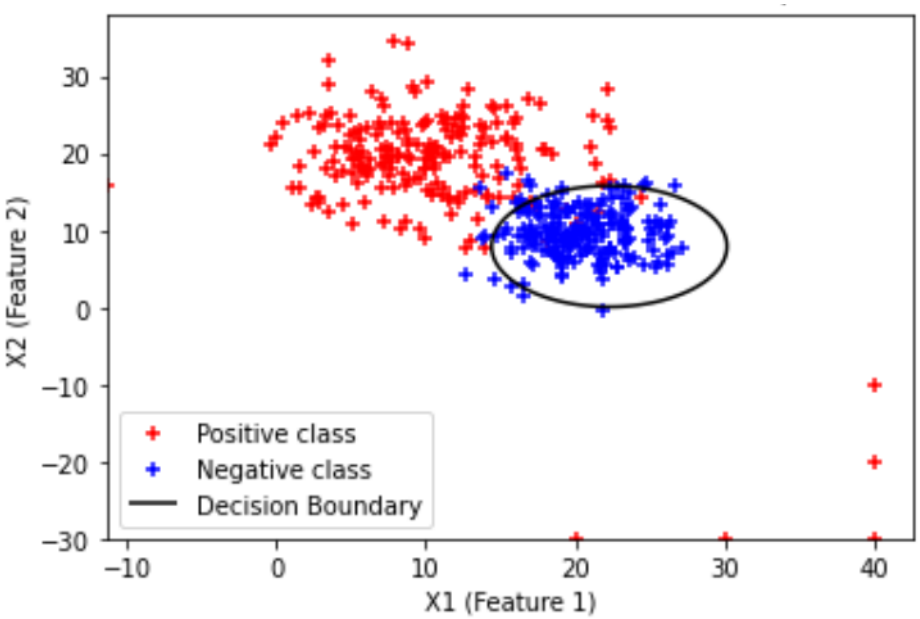
1.2 With equal covariance matrices on binclass.txt

When same covariance matrix is used for positive and negative classes then we obtain a linear decision boundary.



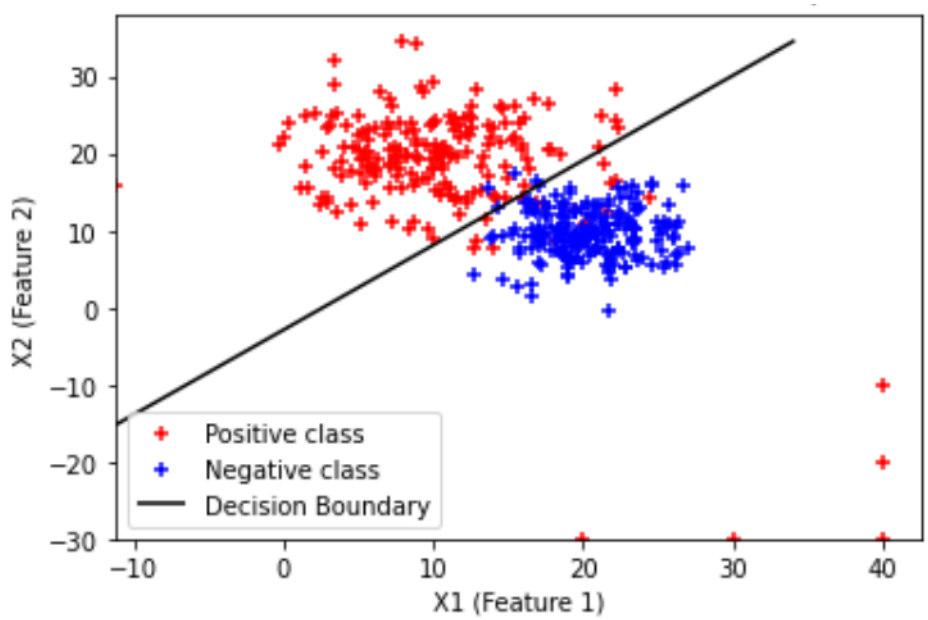
1.3 With different covariance matrices on binclassv2.txt

In this case, a non-linear decision boundary is obtained which is shifted towards the negative (blue) class. This happens because positive class has outliers due to which its variance increases and decision boundary comes close to negative(blue) class.



1.4 With equal covariance matrices on binclassv2.txt

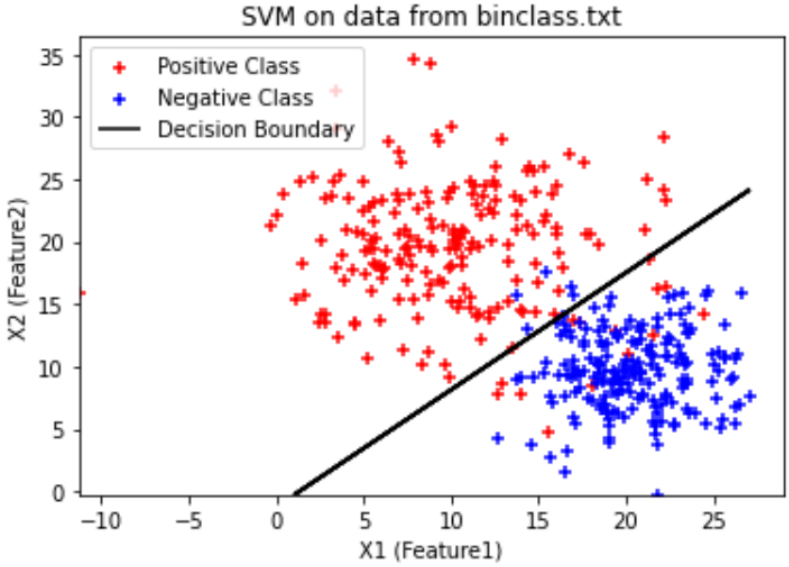
The decision boundary in this case is linear.



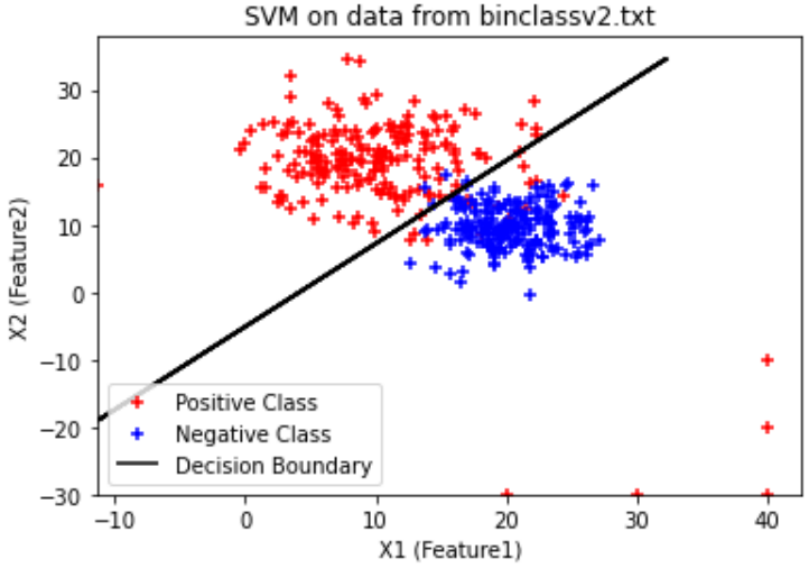
2 SVM

For this model Scikit-learn is used.

2.1 On data from binclass.txt



2.2 On data from binclassv2.txt



3 Observations

The following observations are made from the above plots:

- For the dataset binclass.txt, Generative classification with different covariance matrices gives non-linear decision boundary wheras Generative classification with same covariance matrices and SVM give linear decision boundary. However, it can be observed that all the 3 models gave equally good decision boundary on this dataset.
- The dataset binclassv2.txt, contains outliers for positive (red) class. In generative classification model with different covariance matrices, the decision boundary is closer to negative (blue) class and tries to overfit, possibly because variance for red class will increase due to the outliers and therefore the decision boundary shifts closer to the blue class. However, using the other two models (generative classification with same covariance matrices and SVM) the decision boundary shifts a little towards the red class and thus generative classification with same covariance matrices and SVM generalize well on this dataset.