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

Student Name: Vishweshwar Tyagi
Roll Number: 191173
Date: November 27, 2020

QUESTION 1

Solution 1

Let $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ $\boldsymbol{x}_n \in \mathbb{R}^D$, $y_n \in \{0, 1\}$

For Logistic Regression, we have

$$p(y_n \mid \boldsymbol{w}, \boldsymbol{x}_n) = \text{Ber}(y_n;\ \mu_n) = (\mu_n)^{y_n}(1-\mu_n)^{1-y_n}$$

where

$$\mu_n = \sigma(\boldsymbol{w}^T\boldsymbol{x}_n) = \frac{e^{\boldsymbol{w}^T\boldsymbol{x}_n}}{1+e^{\boldsymbol{w}^T\boldsymbol{x}_n}} \tag{1}$$

Using, $p(y \mid \boldsymbol{w}, X) = \prod_{n=1}^{N} p(y_n \mid \boldsymbol{w}, \boldsymbol{x}_n)$, we obtain the Likelihood given by

$$p(y \mid \boldsymbol{w}, X) = \prod_{n=1}^{N} (\mu_n)^{y_n}(1-\mu_n)^{1-y_n}$$

Taking log, we get the log-Likelihood, $LL(\boldsymbol{w})$, given by

$$LL(\boldsymbol{w}) = \sum_{n=1}^{N} y_n \log(\mu_n) + (1-y_n)\log(1-\mu_n)$$

So, the loss function $L(\boldsymbol{w})$, which is the negative log-Likelihood, is given by

$$\begin{aligned}
L(\boldsymbol{w}) &= -\sum_{n=1}^{N} y_n \log(\mu_n) + (1-y_n)\log(1-\mu_n) \\
&= -\sum_{n=1}^{N} y_n \log(\sigma(\boldsymbol{w}^T\boldsymbol{x}_n)) + (1-y_n)\log(1-\sigma(\boldsymbol{w}^T\boldsymbol{x}_n)) \\
&= -\sum_{n=1}^{N} y_n \log(\frac{e^{\boldsymbol{w}^T\boldsymbol{x}_n}}{1+e^{\boldsymbol{w}^T\boldsymbol{x}_n}}) + (1-y_n)\log(1-\frac{e^{\boldsymbol{w}^T\boldsymbol{x}_n}}{1+e^{\boldsymbol{w}^T\boldsymbol{x}_n}}) \\
&= -\sum_{n=1}^{N} y_n\boldsymbol{w}^T\boldsymbol{x}_n - y_n\log(1+e^{\boldsymbol{w}^T\boldsymbol{x}_n}) - (1-y_n)\log(1+e^{\boldsymbol{w}^T\boldsymbol{x}_n}) \\
&= -\sum_{n=1}^{N} y_n\boldsymbol{w}^T\boldsymbol{x}_n - \log(1+e^{\boldsymbol{w}^T\boldsymbol{x}_n})
\end{aligned} \tag{2}$$

Therefore, we have

$$\begin{aligned}\nabla_{\boldsymbol{w}} L(\boldsymbol{w}) &= -\sum_{n=1}^{N} (y_n - \frac{e^{\boldsymbol{w}^T x_n}}{1 + e^{\boldsymbol{w}^T x_n}}) \boldsymbol{x}_n^T \\ &= -\sum_{n=1}^{N} (y_n - \mu_n) \boldsymbol{x}_n^T \end{aligned} \tag{3}$$

The **Hessian Matrix** is given by $\nabla^2_{\boldsymbol{w}\boldsymbol{w}^T} L(\boldsymbol{w})$, which is obtained as follows

$$\begin{aligned}\nabla^2_{\boldsymbol{w}\boldsymbol{w}^T} L(\boldsymbol{w}) &= \frac{\partial}{\partial \boldsymbol{w}^T} \nabla_{\boldsymbol{w}} L(\boldsymbol{w}) \\ &= -\frac{\partial}{\partial \boldsymbol{w}^T} \sum_{n=1}^{N} (y_n - \mu_n) \boldsymbol{x}_n^T \\ &= \sum_{n=1}^{N} \mu_n (1 - \mu_n) \boldsymbol{x}_n \boldsymbol{x}_n^T \end{aligned} \tag{4}$$

where we used the fact that,

$$\begin{aligned}\frac{\partial \mu_n}{\partial \boldsymbol{w}^T} &= \frac{\partial}{\partial \boldsymbol{w}^T} (\frac{e^{\boldsymbol{w}^T \boldsymbol{x}_n}}{1 + e^{\boldsymbol{w}^T \boldsymbol{x}_n}}) \\ &= \frac{e^{\boldsymbol{w}^T \boldsymbol{x}_n}}{(1 + e^{\boldsymbol{w}^T \boldsymbol{x}_n})^2} \boldsymbol{x}_n \\ &= \frac{e^{\boldsymbol{w}^T \boldsymbol{x}_n}}{1 + e^{\boldsymbol{w}^T \boldsymbol{x}_n}} \times \{1 - \frac{e^{\boldsymbol{w}^T \boldsymbol{x}_n}}{1 + e^{\boldsymbol{w}^T \boldsymbol{x}_n}}\} \times \boldsymbol{x}_n \\ &= \mu_n (1 - \mu_n) \boldsymbol{x}_n \end{aligned} \tag{5}$$

Let

$$\boldsymbol{y} = [\ y_1\ y_2\ \ldots\ y_N\ ]^T \in \mathbb{R}^{N \times 1}$$

$$\boldsymbol{\mu} = [\ \mu_1\ \mu_2\ \ldots\ \mu_N\ ]^T \in \mathbb{R}^{N \times 1}$$

$$X = [\ \boldsymbol{x}_1^T\ \boldsymbol{x}_2^T\ \ldots\ \boldsymbol{x}_N^T\ ]^T \in \mathbb{R}^{N \times D}$$

$$\gamma = \mathrm{diag}(\mu_1(1 - \mu_1),\ \mu_2(1 - \mu_2),\ \ldots\ \mu_N(1 - \mu_N)) \in \mathbb{R}^{N \times N}$$

The given second order **Newton Raphson**'s optimisation update is:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - H^{(t)^{-1}}\boldsymbol{g}^{(t)} \tag{6}$$

where

$$\begin{aligned}\boldsymbol{g} &= \nabla_{\boldsymbol{w}} L(\boldsymbol{w}) \\ &= -\sum_{n=1}^{N} (y_n - \mu_n)\boldsymbol{x}_n^T \\ &= -X^T(\boldsymbol{y} - \boldsymbol{\mu}) \end{aligned} \tag{7}$$

$$\begin{aligned}H &= \nabla^2_{\boldsymbol{w}\boldsymbol{w}^T} L(\boldsymbol{w}) \\ &= \sum_{n=1}^{N} \mu_n(1-\mu_n)\boldsymbol{x}_n\boldsymbol{x}_n^T \\ &= X^T\gamma X \end{aligned} \tag{8}$$

Using (6), (7) and (8), we get

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

$$\text{where } \hat{\boldsymbol{y}}^{(t)} = X\boldsymbol{w}^{(t)} + \boldsymbol{\gamma}^{(t)^{-1}}(\boldsymbol{y} - \boldsymbol{\mu}^{(t)}) \in \mathbb{R}^{N\times 1} \tag{10}$$

We know that this is the minimiser of weighted least squares $(\hat{\boldsymbol{y}}^{(t)} - X\boldsymbol{w}^{(t)})^T\boldsymbol{\gamma}^{(t)}(\hat{\boldsymbol{y}}^{(t)} - X\boldsymbol{w}^{(t)})$ and therefore,

$$\boldsymbol{w}^{(t+1)} = \arg\min_{\boldsymbol{w}^{(t)}}(\hat{\boldsymbol{y}}^{(t)} - X\boldsymbol{w}^{(t)})^T\boldsymbol{\gamma}^{(t)}(\hat{\boldsymbol{y}}^{(t)} - X\boldsymbol{w}^{(t)}) \tag{11}$$

$$= \arg\min_{\boldsymbol{w}^{(t)}} \sum_{n=1}^{N} \boldsymbol{\gamma}_{nn}^{(t)}\ (\hat{y}_n^{(t)} - \boldsymbol{w}^{(t)^T}\boldsymbol{x}_n)^2$$

$$= \arg\min_{\boldsymbol{w}^{(t)}} \sum_{n=1}^{N} \mu_n^{(t)}(1-\mu_n^{(t)})\ (\hat{y}_n^{(t)} - \boldsymbol{w}^{(t)^T}\boldsymbol{x}_n)^2$$

$$= \arg\min_{\boldsymbol{w}^{(t)}} \sum_{n=1}^{N} \gamma_n^{(t)}\ (\hat{y}_n^{(t)} - \boldsymbol{w}^{(t)^T}\boldsymbol{x}_n)^2 \tag{12}$$

$$\text{where } \gamma_n^{(t)} = \mu_n^{(t)}(1-\mu_n^{(t)}) \tag{13}$$

Hence, the required update is:

$$\boldsymbol{w}^{(t+1)} = \arg\min_{\boldsymbol{w}^{(t)}} \sum_{n=1}^{N} \gamma_n^{(t)} \ (\hat{y}_n^{(t)} - {\boldsymbol{w}^{(t)}}^T \boldsymbol{x}_n)^2 \tag{14}$$

$$\text{where } \gamma_n^{(t)} = {\mu_n^{(t)}}_n (1 - \mu_n^{(t)}) \text{ and } \hat{y}_n^{(t)} = [\hat{\boldsymbol{y}}^{(t)}]_n = [X\boldsymbol{w}^{(t)} + {\boldsymbol{\gamma}^{(t)}}^{-1} (\boldsymbol{y} - \boldsymbol{\mu}^{(t)})]_n \tag{15}$$

From this, we can conclude that the Newton update for $\boldsymbol{w}$ corresponds to solving importance-weighted regression problem given in (14) where γ_n denotes the importance of n'th example and $\hat{y}_n$ denotes a modified real-valued label, both of which are given in (15)

Note that $\hat{\boldsymbol{y}}$ is kind of like an adjusted response in comparison to $\boldsymbol{y}$ and the expression for γ_n makes sense because when $\mu_n \in \{0, 1\}$, $\gamma_n = 0$ and hence the weight for
$\hat{y}_n^{(t)} - {\boldsymbol{w}^{(t)}}^T \boldsymbol{x}_n$ becomes 0

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

Student Name: Vishweshwar Tyagi
Roll Number: 191173
Date: November 27, 2020

QUESTION 2

Solution 2

Given $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ $\boldsymbol{x}_n \in \mathbb{R}^{D\times 1}$ and $y_n \in \{\pm 1\}$, we will discuss Kernalized Perceptron

Ignoring the bias term $b \in \mathbb{R}$, let he hyperplane be of the form $\boldsymbol{w}^T\boldsymbol{x} = 0$, where $\boldsymbol{w} \in \mathbb{R}^{D\times 1}$

We know that the standard update for Stochastic gradient descent algorithm for Perceptron is given by $w^{(t+1)} = w^{(t)} + y_n\boldsymbol{x}_n$ when $y_n\boldsymbol{w}^T\boldsymbol{x}_n < 0$

Due to this update, it is easy to see that

$$\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n \tag{1}$$

where α_n denotes the number of times the algorithm makes mistake on n'th example $(\boldsymbol{x}_n, y_n)$

Let $\boldsymbol{\alpha} = [\alpha_1\ \alpha_2\ \ldots\ \alpha_N]^T \in \mathbb{R}^{N\times 1}$

Let $K : \mathbb{R}^{D\times 1}\times\mathbb{R}^{D\times 1} \to \mathbb{R}\ \ni\ (\boldsymbol{x}_i, \boldsymbol{x}_j) \mapsto \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$ be arbitrary kernel where $\phi : \mathbb{R}^D \to \mathcal{X}\ \ni\ \boldsymbol{x} \mapsto \phi(\boldsymbol{x})$ is the implicit feature mapping induced by K

While working in ϕ induced space $(\mathcal{X})$, from (1), we will have

$$\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \phi(\boldsymbol{x}_n) \tag{2}$$

Hence, the prediction $\hat{y}$ made on example $(\boldsymbol{x}, y)$ will be

$$\begin{aligned}
\hat{y} &= \text{sign}(\boldsymbol{w}^T\phi(\boldsymbol{x})) \\
&= \text{sign}(\ [\sum_{n=1}^{N} \alpha_n y_n \phi(\boldsymbol{x}_n)^T]\phi(\boldsymbol{x}\ )) \\
&= \text{sign}(\ \sum_{n=1}^{N} \alpha_n y_n [\phi(\boldsymbol{x}_n)^T\phi(\boldsymbol{x})]\ ) \\
&= \text{sign}(\ \sum_{n=1}^{N} \alpha_n y_n K(\boldsymbol{x}_n, \boldsymbol{x})\ ) \qquad (3)
\end{aligned}$$

Since we don't explicitly know what ϕ is, we will have to solve the dual formulation for $\boldsymbol{\alpha}$ as follows:

Kernelized Perceptron − Stochastic gradient descent

1. Initialize : $\alpha = \alpha^{(0)} = \mathbf{0} \in \mathbb{R}^{N\times 1},\ t = 0,$ and set the learning rate $\eta_t = 1\ \forall t$
2. Pick random example : $(\boldsymbol{x}_n, y_n)\ \ n \in \{1, 2\ \ldots\ N\}$
3. Let : $\hat{y}_n = \text{sign}(\ \sum_{i=1}^{N} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}_n)\ )$
4. Check : If $y_n \hat{y}_n < 0,$ then update $\alpha_n = \alpha_n + 1$
   (Increase the number of mistakes made on $(\boldsymbol{x}_n, y_n)$)
5. Stopping condition : Go to 2. if not yet converged

Student Name: Vishweshwar Tyagi
Roll Number: 191173
Date: November 27, 2020

QUESTION 3

Solution 3

Given $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ $\boldsymbol{x}_n \in \mathbb{R}^{D\times 1}$ and $y_n \in \{\pm 1\}$

The objective for SVM with unequal class importance is

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{||\boldsymbol{w}||}{2} + \sum_{n=1}^{N} C_{y_n}\xi_n \\ \text{subject to} \quad & y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1 - \xi_n \;\; \forall n = 1, 2 \dots N \\ & \xi_n \geq 0 \quad \forall n = 1, 2 \dots N \end{aligned} \tag{1}$$

The Lagrangian (primal) problem for this is given by

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad \frac{||\boldsymbol{w}||}{2} + \sum_{n=1}^{N} C_{y_n}\xi_n + \sum_{n=1}^{N} \alpha_n((1-\xi_n) - y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b)) - \sum_{n=1}^{N} \beta_n\xi_n \tag{2}$$

Let

$$L_p(\boldsymbol{w}, b, \boldsymbol{\xi}) = \frac{||\boldsymbol{w}||}{2} + \sum_{n=1}^{N} C_{y_n}\xi_n + \sum_{n=1}^{N} \alpha_n((1-\xi_n) - y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b)) - \sum_{n=1}^{N} \beta_n\xi_n \tag{3}$$

Setting partial derivatives to 0, we obtain

$$\begin{aligned} & \nabla_{\boldsymbol{w}} L_p = 0 \\ \Rightarrow \; & \boldsymbol{w} - \sum_{n=1}^{N} \alpha_n y_n x_n = 0 \\ \Rightarrow \; & \boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n x_n \end{aligned} \tag{4}$$

$$\begin{aligned} & \nabla_b L_p = 0 \\ \Rightarrow \; & \sum_{n=1}^{N} \alpha_n y_n = 0 \end{aligned} \tag{5}$$

$$\begin{aligned} & \nabla_{\xi_n} L_p = 0 \\ \Rightarrow \; & C_{y_n} - \alpha_n - \beta_n = 0 \\ \Rightarrow \; & \alpha_n = C_{y_n} - \beta_n \end{aligned} \tag{6}$$

Substituting (4) − (6) in (3) and using (2) gives

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j - \sum_{n=1}^{N}\alpha_n \\ \text{subject to} \quad & 0 \le \alpha_n \le C_{y_n} \;\; \forall n = 1, 2 \ldots N \\ & \sum_{n=1}^{n}\alpha_n y_n = 0 \end{aligned}$$

or, we can state the **dual** formulation (maximisation problem) as

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j \\ \text{subject to} \quad & 0 \le \alpha_n \le C_{y_n} \;\; \forall n = 1, 2 \ldots N \\ & \sum_{n=1}^{n}\alpha_n y_n = 0 \end{aligned} \tag{7}$$

(7) gives the required dual (maximisation) formulation.

This differs from the standard dual problem because here we have separate upper limits for α_n corresponding to $\boldsymbol{x}_n$ depending on whether $y_n = +1$ (in which case the upper bound will be C_1) and $y_n = -1$ (in this case upper bound will be C_{-1}). This makes sense since we want to upper bound the number of true positives and false negatives by different values because one of these might cost more than other in practical applications.

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

Student Name: Vishweshwar Tyagi
Roll Number: 191173
Date: November 27, 2020

QUESTION 4

Solution 4

Here, we would like to improve upon the standard K-means batch algorithm and make it "online" using Stochastic gradient descent.

Let $\{\boldsymbol{x}_n\}_{n=1}^N \in \mathbb{R}^{D\times 1}$, K denote the number of clusters (given) and $\boldsymbol{\mu}_k \in \mathbb{R}^{D\times 1}$ denote the centroids (to be learned) $k = 1, 2 \dots K$

Before giving the algorithm, for sole purpose of convenience, we would like to rewrite the given K-means objective function

We are given that $\mathcal{L} = \sum_{n=1}^{N}\sum_{k=1}^{K} z_{nk}||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2$

where $\boldsymbol{z}_n = [z_{n1} \ \dots \ z_{nK}] \in \mathbb{R}^{1\times K}$ is one-hot representation of the $\boldsymbol{x}_n$ depending on which cluster it belongs in.

Equivalently, this objective can be re-written as

$$\mathcal{L} = \sum_{n=1}^{N} ||\boldsymbol{x}_n - \boldsymbol{\mu}_{z_n}||^2$$

For a single vector $\boldsymbol{x}_n$, this objective is given by:

$$\mathcal{L}(\boldsymbol{x}_n, \boldsymbol{\mu}_{z_n}) = ||\boldsymbol{x}_n - \boldsymbol{\mu}_{z_n}||^2$$

More generally, define

$$\mathcal{L}(\boldsymbol{x}_n, \boldsymbol{\mu}_k) = ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2 \ \text{ for } n = 1 \dots N, \ k = 1 \dots K \tag{1}$$

They key to designing the online algorithm is to observe the following:

Let $\mu^{(n)}$ denote the mean of a sample of n points $\{\boldsymbol{x}_1, \dots \boldsymbol{x}_n\}$, then, we have

$$\boldsymbol{\mu}^{(n)} = \frac{1}{n}\sum_{j=1}^{n}\boldsymbol{x}_j = \frac{1}{n}\sum_{j=1}^{n-1}\boldsymbol{x}_j + \frac{1}{n}\boldsymbol{x}_n = \frac{n-1}{n}\boldsymbol{\mu}^{(n-1)} + \frac{1}{n}\boldsymbol{x}_n \tag{2}$$

Hence, $\boldsymbol{\mu}^{(n)}$ is a linear, infact, a convex combination of $\boldsymbol{\mu}^{(n-1)}$ and $\boldsymbol{x}_n$

Further, we can see that

$$\boldsymbol{\mu}^{(n)} = \frac{n-1}{n}\boldsymbol{\mu}^{(n-1)} + \frac{1}{n}\boldsymbol{x}_n = \boldsymbol{\mu}^{(n-1)} - \frac{1}{n}\boldsymbol{\mu}^{(n-1)} + \frac{1}{n}\boldsymbol{x}_n$$

Therefore,

$$\boldsymbol{\mu}^{(n)} = \boldsymbol{\mu}^{(n-1)} + \frac{1}{n}(\boldsymbol{x}_n - \boldsymbol{\mu}^{(n-1)}) \tag{3}$$

After observing this, we have the following online algorithm (4)

1. Initialise centroids : $\{\boldsymbol{\mu}_k\}_{k=1}^{K}$ randomly, Set $n_k = 1$ for $k = 1 \dots K$
   where n_k denotes size of cluster with centroid $\boldsymbol{\mu}_k$
2. Input : Suppose $\boldsymbol{x}_n$ is the incoming vector
3. Find closest centroid (Greedy) : $\boldsymbol{\mu}_j = \arg\min_i \mathcal{L}(\boldsymbol{x}_n, \boldsymbol{\mu}_i)$ refer (1)
4. Assign cluster to the input : $\boldsymbol{z}_n = [0_1\ 0_2\ \dots 0_{j-1}\ 1_j\ 0_{j+1}\ \dots\ 0_K]$
5. Update the closest centroid : $\boldsymbol{\mu}_j = \boldsymbol{\mu}_j + \frac{1}{n_j + 1}(\boldsymbol{x_n} - \boldsymbol{\mu}_j)$
6. Update size of cluster : $n_j = n_j + 1$
7. Stopping condition : Stop if there are no more incoming vectors, i.e, all have been assigned a cluster, else repeat steps $(2) - (7)$

From the algorithm above, we can answer the questions asked in assignment:

Q. How would you solve step 1? (step 1 with respect to assignment)
A. We are greedily assigning incoming vector $\boldsymbol{x}_n$ to the cluster of the closest centroid based on Euclidean distance.

Q. What will be the SGD-based cluster mean update equations for step 2?
Intuitively, why does the update equation make sense?
A. The SGD-based cluster mean/centroid update is given in step 5. of algorithm in (4)

$$\boldsymbol{\mu}_j = \boldsymbol{\mu}_j + \frac{1}{n_j + 1}(\boldsymbol{x_n} - \boldsymbol{\mu}_j)$$

which makes intuitive sense from (2) and (3) because we are taking weighted average of the incoming vector with the vectors already present in the cluster.

Q. For your derived SGD update, suggest a good choice of the step size and mention why you think it is a good choice.
A. From step 5. we see that the step size is given by $\frac{1}{n_j + 1}$

This is a good choice for step size because of two reasons:
One, it makes intuitive sense from (2) and (3) since,
we're supposed to divide by the new size of the cluster in weighted average.
Two, it is always ≤ 1 and monotonically decreasing as more data points (vectors) are incoming.

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

Student Name: Vishweshwar Tyagi
Roll Number: 191173
Date: November 27, 2020

QUESTION 5

Solution 5

Suppose we are given $\{\boldsymbol{x}_n\}_{n=1}^N$ where $\boldsymbol{x}_n \in \mathbb{R}^{D\times 1}$, K be the number of clusters (given). We want to implement **kernelized** K-means clustering

Let $K(\cdot,\cdot) : \mathbb{R}^{D\times 1} \times \mathbb{R}^{D\times 1} \to \mathbb{R}$ be arbitrary kernel that implicitly induces the map $\phi(\cdot) : \mathbb{R}^{D\times 1} \to \mathcal{F}$ where $\mathcal{F}$ is abstract space, possibly infinite dimensional, and assume that we can use the **kernel trick**

The challenge here is due to the possibility of $\mathcal{F}$ being infinite dimensional, we are unable to store $\phi(\{\boldsymbol{x}_n\}_{n=1}^N)$, i.e, the kernel-induced feature map representation of the data points nor the cluster centroids since these lie in $\mathcal{F}$

To overcome, we will use the **kernel trick** that replaces the Euclidean inner product $\langle\phi(\cdot),\phi(\cdot)\rangle$ by $K(\cdot,\cdot)$. This allows us to operate in the original space $\mathbb{R}^{D\times 1}$ without having to care about $\mathcal{F}$

For convenience, let for each $\boldsymbol{x}_n \in \mathbb{R}^{D\times 1}$, assign $\boldsymbol{z}_n \in \mathbb{R}^{1\times K}$ a one-hot representation for the cluster that $\boldsymbol{x}_n$ belongs to, i.e, for example if $\boldsymbol{x}_n$ belongs to the cluster of centroid $\boldsymbol{\mu}_j$, then $\boldsymbol{z}_n = [0_1,\ 0_2,\ \ldots 0_{j-1}\ 1\ 0_{j+1}\ \ldots 0_K]$

Also, let $\mathcal{C}_j$ represent the set of data points that belong in the cluster with centroid $\boldsymbol{\mu}_j$

The standard K-means clustering randomly initialises the K centroids $\boldsymbol{\mu}_n \in \mathbb{R}^{D\times 1}$, for $n = 1, 2 \ldots K$

Then, we assign cluster to each data point by the rule

$$\boldsymbol{z}_n = \arg\min_{k\in\{1,2\ldots K\}} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2 \tag{1}$$

After this, each centroid is updated by the rule

$$\boldsymbol{\mu}_j = \frac{1}{|\mathcal{C}_j|} \sum_{\boldsymbol{x}_n \in \mathcal{C}_j} \boldsymbol{x}_n \tag{2}$$

and we repeat this until the data points are re-assigned or some given metric of convergence is satisfied.

In the **kernelized** version of this, we will randomly initialise the K centroids $\boldsymbol{\mu}_k$ in the original space itself, i.e, $\boldsymbol{\mu}_k \in \mathbb{R}^{D\times 1}$ which will represent the clusters $\mathcal{C}_k$, for $k = 1, 2 \dots K$

The assignment step will require us to compute for each $\boldsymbol{x}_n$

$$\boldsymbol{z}_n = \arg\min_{k=1,2\dots K} ||\phi(\boldsymbol{x}_n) - \boldsymbol{\mu}_k||^2 \tag{3}$$

and the centroids will then be recomputed as

$$\boldsymbol{\mu}_j = \frac{1}{|\mathcal{C}_j|} \sum_{\boldsymbol{x}_n \in \mathcal{C}_j} \phi(\boldsymbol{x}_n) \tag{4}$$

Now note that, we can compute (3) using (4) and the **kernel trick**

$$\begin{aligned}
\boldsymbol{z}_n &= \arg\min_{k=1,2\dots K} ||\phi(\boldsymbol{x}_n) - \boldsymbol{\mu}_k||^2 \\
&= \arg\min_{k=1,2\dots K} \langle \phi(\boldsymbol{x}_n), \phi(\boldsymbol{x}_n) \rangle + \langle \boldsymbol{\mu}_k, \boldsymbol{\mu}_k \rangle - 2\langle \phi(\boldsymbol{x}_n), \boldsymbol{\mu}_k \rangle \\
&= \arg\min_{k=1,2\dots K} \langle \boldsymbol{\mu}_k, \boldsymbol{\mu}_k \rangle - 2\langle \phi(\boldsymbol{x}_n), \boldsymbol{\mu}_k \rangle \\
&= \arg\min_{k=1,2\dots K} \Big\langle \frac{1}{|\mathcal{C}_k|} \sum_{\boldsymbol{x}_i \in \mathcal{C}_k} \phi(\boldsymbol{x}_i), \frac{1}{|\mathcal{C}_k|} \sum_{\boldsymbol{x}_j \in \mathcal{C}_k} \phi(\boldsymbol{x}_j) \Big\rangle - 2\Big\langle \phi(\boldsymbol{x}_n), \frac{1}{|\mathcal{C}_k|} \sum_{\boldsymbol{x}_j \in \mathcal{C}_k} \phi(\boldsymbol{x}_j) \Big\rangle \\
&= \arg\min_{k=1,2\dots K} \frac{1}{|\mathcal{C}_k|^2} \sum_{\boldsymbol{x}_i \in \mathcal{C}_k} \sum_{\boldsymbol{x}_j \in \mathcal{C}_k} \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle - \frac{2}{|\mathcal{C}_k|} \sum_{\boldsymbol{x}_j \in \mathcal{C}} \langle \phi(\boldsymbol{x}_n), \phi(\boldsymbol{x}_j) \rangle \\
&= \arg\min_{k=1,2\dots K} \frac{1}{|\mathcal{C}_k|^2} \sum_{\boldsymbol{x}_i \in \mathcal{C}_k} \sum_{\boldsymbol{x}_j \in \mathcal{C}_k} K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \frac{2}{|\mathcal{C}_k|} \sum_{\boldsymbol{x}_j \in \mathcal{C}_k} K(\boldsymbol{x}_n, \boldsymbol{x}_j)
\end{aligned} \tag{5}$$

which is independent of the centroids $\boldsymbol{\mu}_k$

Hence, the **kernelized K-means** algorithm can be given as (6)

1. Initialise the clusters : Initialise $\{\boldsymbol{\mu}_k\}_{k=1}^K$ randomly as one of the given data points
   Set $\mathcal{C}_k = \{\boldsymbol{\mu}_k\}$ which represents the $k'th$ cluster

2. Assign cluster (Greedy) : for $j = 1, 2 \dots N$
   $$\boldsymbol{z}_n = \arg\min_{k=1,2\dots K} \frac{1}{|\mathcal{C}_k|^2} \sum_{\boldsymbol{x}_i \in \mathcal{C}_k} \sum_{\boldsymbol{x}_j \in \mathcal{C}_k} K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \frac{2}{|\mathcal{C}_k|} \sum_{\boldsymbol{x}_j \in \mathcal{C}_k} K(\boldsymbol{x}_n, \boldsymbol{x}_j)$$

3. Update the clusters : for $j = 1, 2 \dots K$
   $$\mathcal{C}_j = \{\boldsymbol{x}_n \mid \boldsymbol{z}_n = [0_1, \dots, 0_{j-1}\ 1\ 0_{j+1} \dots 0_K]\}$$

4. Stopping condition : Stop if the data points are re-assigned or if any other given metric of convergence is met, else repeat (2) − (4)

Q. What is the difference between how the clusters centroids would need to be stored in kernel K-means versus how they are stored in standard K-means?

A. In standard K-means, we have to store explicitly the centroids for each iteration.
In kernelized K-means, we don't have to store the centroids, only have to maintain the clusters, i.e, maintain which data points belong to which cluster.

Q. Assuming each input to be D-dimensional in the original feature space, and N to be the number of inputs, how does kernel K-means compare with standard K-means in terms of the cost of input to cluster mean distance calculation?

A. In standard K-means, cost of computing distance between a data point and a cluster centroid is of $\mathcal{O}(D)$ time complexity since we are calculating Euclidean distance between two D dimensional points
In kernelized K-means, however, distance between data point and cluster centroid is done using the objective in (5) which is of $\mathcal{O}(N^2)$
During cluster assignment of $\boldsymbol{x}_n$ the standard K-means requires us to calculate the distance of $\boldsymbol{x}_n$ from all the K clusters which is done in $\mathcal{O}(KD)$. Adding for all data points, this gives $\mathcal{O}(KDN)$ time complexity in total
For cluster assignment in kernelized K-means, we have to calculate step 2. of (6) which is $\mathcal{O}(KN^2)$ for each $\boldsymbol{x}_n$ and adding for all data points gives $\mathcal{O}(KN^3)$ in total
We see that kernelized version is slower than standard one.

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

Student Name: Vishweshwar Tyagi
Roll Number: 191173
Date: November 27, 2020

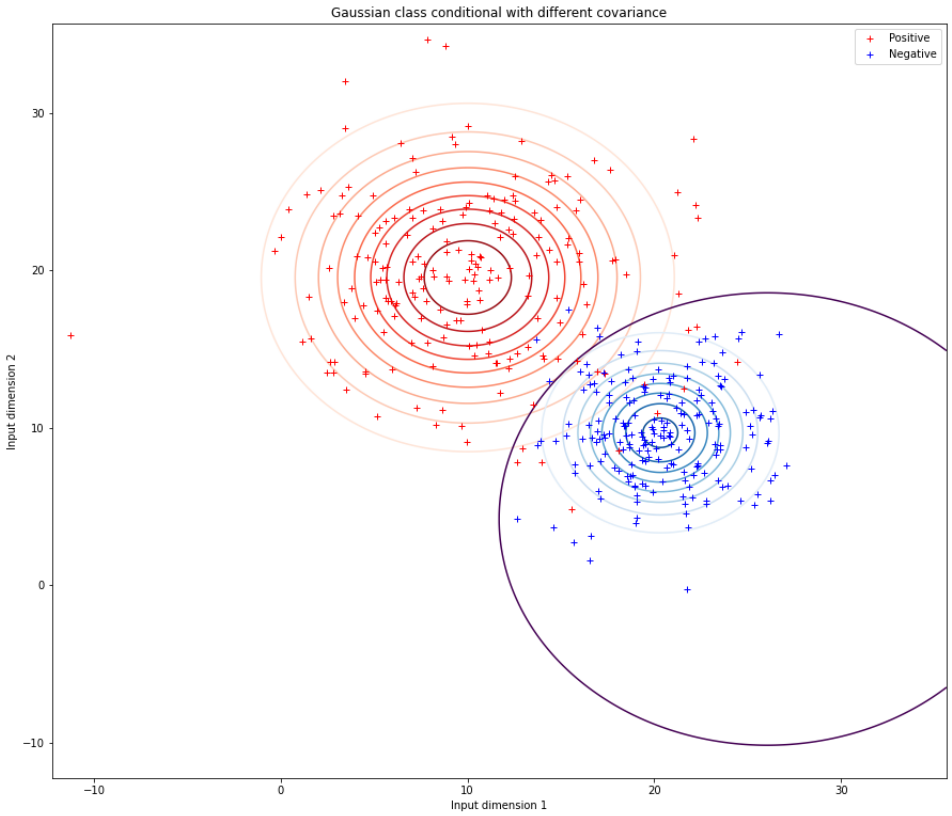
QUESTION 6

Solution 6

Dataset : binclass.txt

Model : Positive and negative labeled examples are sampled from $\mathcal{N}(\mu_+, \sigma_+^2 I_2)$ and $\mathcal{N}(\mu_-, \sigma_-^2 I_2)$ respectively

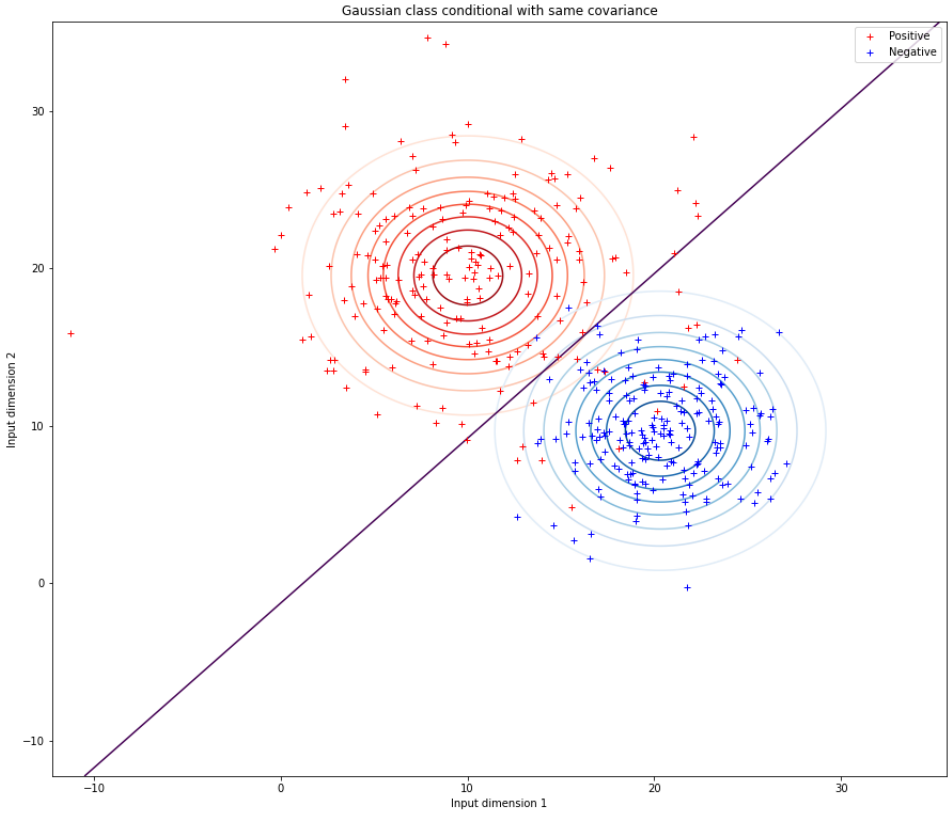
Boundary : $\mathcal{N}(\mu_+, \sigma_+^2 I_2) - \mathcal{N}(\mu_-, \sigma_-^2 I_2) = 0$, Linear



Dataset : binclass.txt

Model : Positive and negative labeled examples are sampled from $\mathcal{N}(\mu_+, \sigma^2 I_2)$ and $\mathcal{N}(\mu_-, \sigma^2 I_2)$ respectively

Boundary : $\mathcal{N}(\mu_+, \sigma^2 I_2) - \mathcal{N}(\mu_-, \sigma^2 I_2) = 0$, Linear

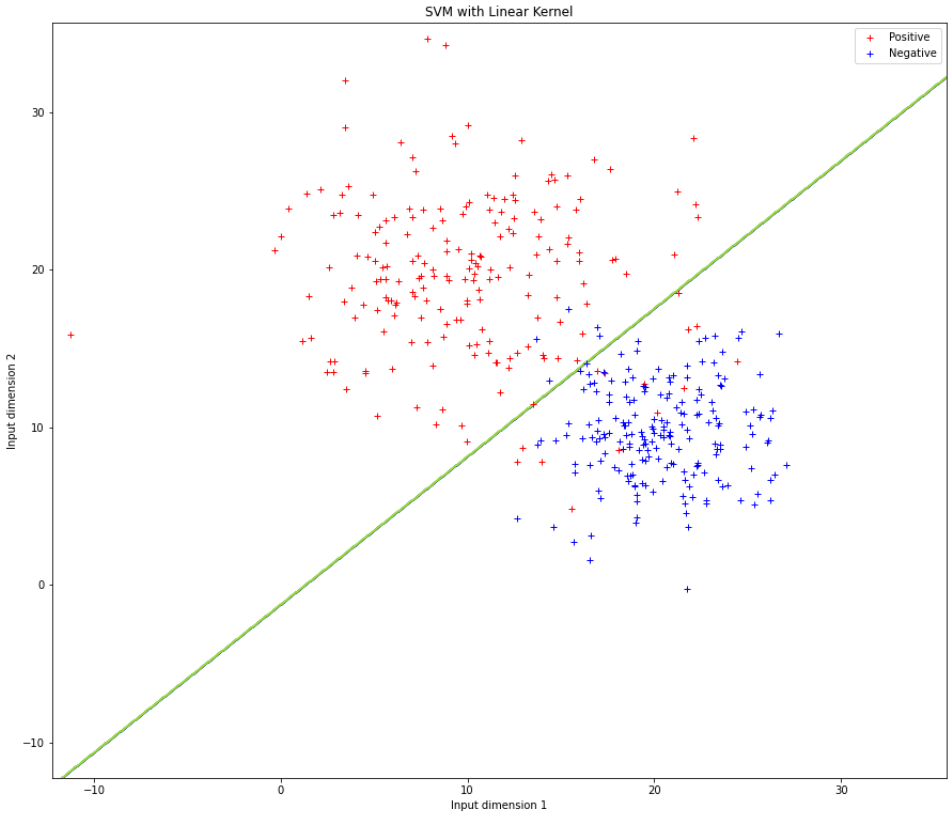


Dataset : binclass_libSVM.txt

Model : SVM with linear kernel and hyper-parameter $C = 1$

Boundary : Linear

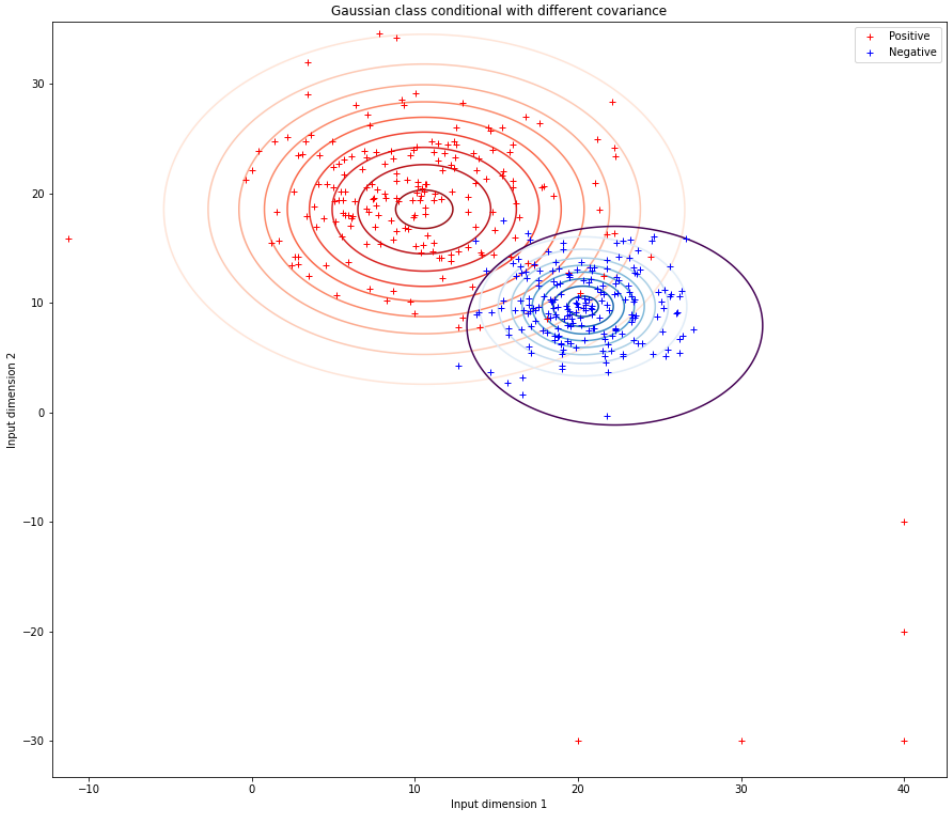
Library : SVC from sklearn.svm



Dataset : binclassv2.txt

Model : Positive and negative labeled examples are sampled from $\mathcal{N}(\mu_+, \sigma_+^2 I_2)$ and $\mathcal{N}(\mu_-, \sigma_-^2 I_2)$ respectively

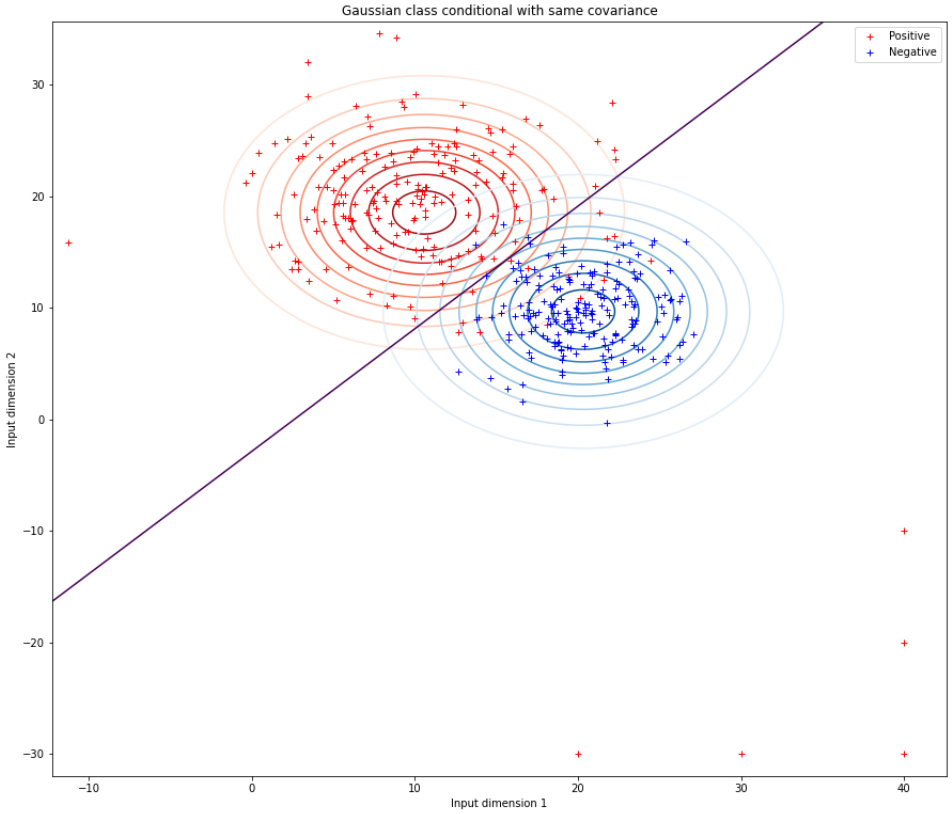
Boundary : $\mathcal{N}(\mu_+, \sigma_+^2 I_2) - \mathcal{N}(\mu_-, \sigma_-^2 I_2) = 0$, Linear



Dataset : binclassv2.txt

Model : Positive and negative labeled examples are sampled from $\mathcal{N}(\mu_+, \sigma^2 I_2)$ and $\mathcal{N}(\mu_-, \sigma^2 I_2)$ respectively

Boundary : $\mathcal{N}(\mu_+, \sigma^2 I_2) - \mathcal{N}(\mu_-, \sigma^2 I_2) = 0$, Linear

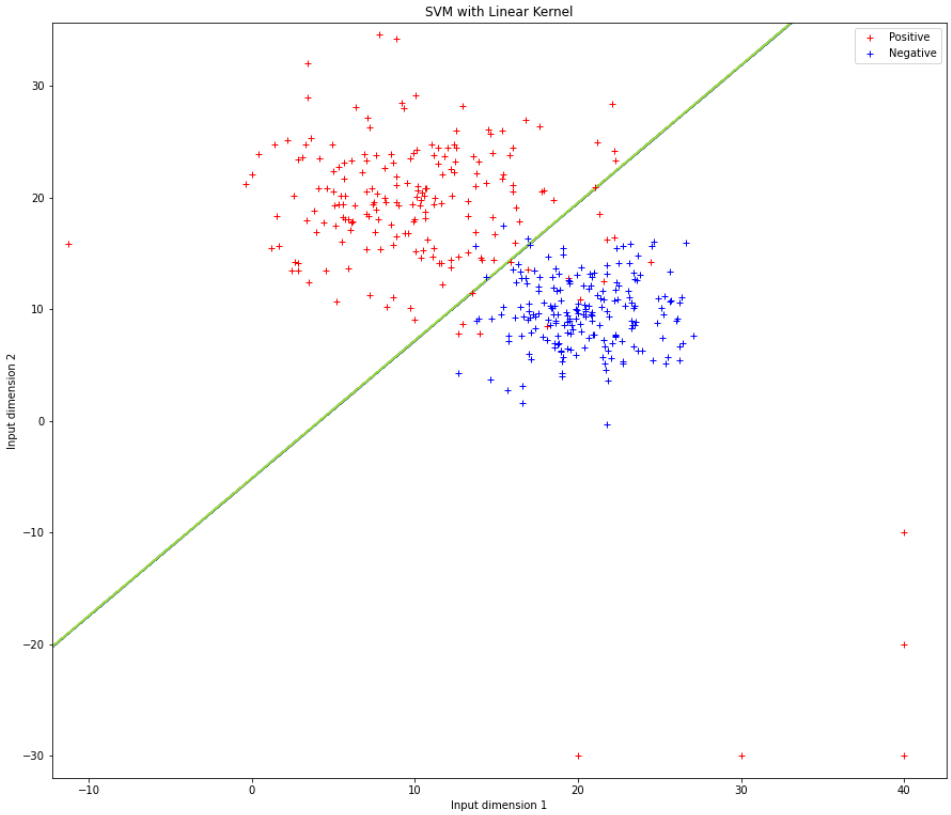


Dataset : binclassv2_libSVM.txt

Model : SVM with linear kernel and hyper-parameter $C = 1$

Boundary : Linear

Library : SVC from sklearn.svm



Conclusion

For the first dataset, binclass.txt, all of the three models, i.e, Gaussian with different/same covariance and linear SVM seem to do equally good job. However, for the second dataset, binclassv2.txt, we see that this dataset contains some outliers in the +ve (red) class due to which Gaussian with different covariance is overfitting the data, whereas Gaussian with same covariance and linear SVM are still able to generalise well.