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

Student Name: Sakshi
Roll Number: 180653
Date: November 27, 2020

QUESTION 1

A vector symbol $\mathbf{b}$, a symbol in blackboard font $\mathbb{R}$, a symbol in calligraphic font $\mathcal{A}$, some colored text

Given the loss function:

$$\mathcal{L}(\boldsymbol{w}) = -\sum_{n=1}^{N}(y_n \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x_n} - \log(1 + exp(\boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x_n})))$$

For the update equation, we need the gradient and the Hessian, which are derived as follows:

$$\begin{aligned}
\boldsymbol{g} &= \frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w}} \\
&= -\sum_{n=1}^{N}(y_n \boldsymbol{x_n} - \frac{exp(\boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x_n})}{1 + exp(\boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x_n})} \boldsymbol{x_n}) \\
&= -\sum_{n=1}^{N}(y_n \boldsymbol{x_n} - \sigma_n \boldsymbol{x_n}) \qquad \text{.......taking} \quad \frac{exp(\boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x_n})}{1 + exp(\boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x_n})} = \sigma_n \\
&= -\sum_{n=1}^{N}(y_n - \sigma_n)\boldsymbol{x_n} \\
&= \boldsymbol{X}^{\boldsymbol{T}}(\boldsymbol{\sigma} - \mathbf{y})
\end{aligned}$$

Here $\boldsymbol{X}$ is the standard feature matrix, $\boldsymbol{y}$ is the output vector and $\boldsymbol{\sigma}$ is a vector of σ_n. We can't get a closed form solution by putting the gradient to 0. We use second order optimization based iterative updates.
Now, we find the Hessian of the loss function:

$$\begin{aligned}
\boldsymbol{\mathcal{H}} &= \frac{\partial^2 \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w} \partial \boldsymbol{w}^{\boldsymbol{T}}} \\
&= \frac{\partial \boldsymbol{g}^{\boldsymbol{T}}}{\partial \boldsymbol{w}} \\
&= \frac{\partial}{\partial \boldsymbol{w}}(-\sum_{n=1}^{N}(y_n - \sigma_n)\boldsymbol{x}_{\boldsymbol{n}}^{\boldsymbol{T}})
\end{aligned}$$

Using:

$$\frac{\partial \sigma_n}{\partial \boldsymbol{w}} = \sigma_n(1 - \sigma_n)\boldsymbol{x_n}$$

We get:

$$\begin{aligned}
\boldsymbol{\mathcal{H}} &= \sum_{n=1}^{N} \sigma_n(1 - \sigma_n)\boldsymbol{x_n}\boldsymbol{x}_{\boldsymbol{n}}^{\boldsymbol{T}} \\
&= \boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{D}\boldsymbol{X}
\end{aligned}$$

Here D is a diagonal matrix with $D_{ii} = \sigma_i$.
Now, using the properties of arg-min, the update equation is derived as follows:

$$\begin{aligned}
\boldsymbol{w^{t+1}} &= arg\ \underset{\boldsymbol{w}}{min}\ \left(\mathcal{L}(\boldsymbol{w^t}) + \boldsymbol{(g^t)^T}(\boldsymbol{w} - \boldsymbol{w^t}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w^t})^T\boldsymbol{\mathcal{H}^t}(\boldsymbol{w} - \boldsymbol{w^t})\right) \\
&= arg\ \underset{\boldsymbol{w}}{min}\ \left(-\sum_{n=1}^{N}(y_n - \sigma_n^t)\boldsymbol{x_n^T}(\boldsymbol{w} - \boldsymbol{w^t}) + \frac{1}{2}\sum_{n=1}^{N}\sigma_n^t(1-\sigma_n^t)(\boldsymbol{x_n^T}(\boldsymbol{w} - \boldsymbol{w^t}))^T(\boldsymbol{x_n^T}(\boldsymbol{w} - \boldsymbol{w^t}))\right) \\
&= arg\ \underset{\boldsymbol{w}}{min}\ \left(-\sum_{n=1}^{N}(y_n - \sigma_n^t)\boldsymbol{x_n^T}(\boldsymbol{w} - \boldsymbol{w^t}) + \frac{1}{2}\sum_{n=1}^{N}\sigma_n^t(1-\sigma_n^t)(\boldsymbol{x_n^T}(\boldsymbol{w} - \boldsymbol{w^t}))^2\right)
\end{aligned}$$

We have used the fact that while doing arg min operations, modulus square can be simply replaced by square. Now, adding a term that doesn't depend on $\boldsymbol{w}$ doesn't affect the optimizer's value. Thus,

$$\begin{aligned}
\boldsymbol{w^{t+1}} &= arg\ \underset{\boldsymbol{w}}{min}\ \left(\sum_{n=1}^{N}\frac{1}{2}\sigma_n^t(1-\sigma_n^t)\big[-2.\frac{y_n - \sigma_n^t}{\sigma_n^t(1-\sigma_n^t)}\boldsymbol{x_n^T}(\boldsymbol{w} - \boldsymbol{w^t}) + (\boldsymbol{x_n^T}(\boldsymbol{w} - \boldsymbol{w^t}))^2 + \big(\frac{y_n - \sigma_n^t}{\sigma_n^t(1-\sigma_n^t)}\big)^2\big]\right) \\
&= arg\ \underset{\boldsymbol{w}}{min}\ \left(\sum_{n=1}^{N}\frac{1}{2}\sigma_n^t(1-\sigma_n^t)\big[\frac{y_n - \sigma_n^t}{\sigma_n^t(1-\sigma_n^t)} - \boldsymbol{x_n^T}(\boldsymbol{w} - \boldsymbol{w^t})\big]^2\right) \\
&= arg\ \underset{\boldsymbol{w}}{min}\ \left(\sum_{n=1}^{N}\frac{1}{2}\sigma_n^t(1-\sigma_n^t)\big[\frac{y_n - \sigma_n^t}{\sigma_n^t(1-\sigma_n^t)} - (\boldsymbol{w} - \boldsymbol{w^t})^T\boldsymbol{x_n}\big]^2\right) \\
&= arg\ \underset{\boldsymbol{w}}{min}\ \left(\sum_{n=1}^{N}\gamma_n^t(\hat{y_n}^t - \boldsymbol{w^T x_n})^2\right)
\end{aligned}$$

This is the desired form. Here,

1. $\gamma_n^t = \frac{1}{2}\sigma_n^t(1-\sigma_n^t)$
2. $\hat{y_n}^t = \frac{y_n - \sigma_n^t}{\sigma_n^t(1-\sigma_n^t)} + (\boldsymbol{w^t})^T\boldsymbol{x_n}$

Thus, we have shown that second-order optimization for logistic regression yields the update equation as given in the problem. Moreover, we can observe that the weight γ for each input varies according to σ. When there is ambiguity in assigning the classes, the weight or cost should be more i.e. γ should be more. Thus, when there is no ambiguity, cost is less.

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

Student Name: Sakshi
Roll Number: 180653
Date: November 27, 2020

QUESTION 2

We are given that the weight vector learned by Perceptron can be written as:

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

And we know that the prediction rule is:

$$\begin{aligned} y_* &= sign(\boldsymbol{w^T x_*}) \\ &= sign(\sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x_n^T x_*}) \end{aligned}$$

Here α_n is the number of times the perceptron makes mistake on example n.

Thus, for the kernelized form with feature space as ϕ it'll be:

$$\begin{aligned} y_* &= sign\left(\sum_{n=1}^{N} \alpha_n y_n \boldsymbol{\phi(x_n)^T \phi(x_*)}\right) \\ &= sign\left(\sum_{n=1}^{N} \alpha_n y_n \boldsymbol{k(x_n, x_*)}\right) \end{aligned}$$

1. **Initialization:** Make $\alpha_i = 0 \ \forall i \in \{1, 2, .., n\}$.
2. **Mistake condition:** For randomly chosen training example $(\boldsymbol{x_n}, y_n)$, using the update condition we can say there is a mistake if: $\left(y_n * \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{k(x_i, x_*)}\right) < 0$. This is based on the current values of α_i's.
3. **Updation:** If there is a mistake, just update the corresponding α_n by incrementing it. Thus updation is $\alpha_n + +$.

These steps are repeated until convergence. And the prediction rule has been specified above itself. The thing to note here is that we haven't used the weight vector anywhere, as the feature mapping ϕ can be infinite-dimensional making it impossible to compute and store the weight vector. Thus, we make use of the kernel definitions, and modify the algorithm so that it is independent of the weight vector. α takes care of what the weight vector would have been.

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

Student Name: Sakshi
Roll Number: 180653
Date: November 27, 2020

QUESTION 3

The primal formulation for the given soft margin SVM optimization problem is:

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

$$\text{subject to } y_n(\boldsymbol{w}^T\boldsymbol{x_n} + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \quad n \in \{1,2,\dots,N\}$$

The Dual Lagrangian is given as:

$$\min_{\boldsymbol{w},b,\xi} \max_{\alpha\geq 0,\beta\geq 0} \mathcal{L}(\boldsymbol{w},b,\xi,\alpha,\beta) = \frac{||\boldsymbol{w}||^2}{2} + \sum_{n=1}^{N} C_{y_n}\xi_n + \sum_{n=1}^{N}\alpha_n\{1 - y_n(\boldsymbol{w}^T\boldsymbol{x_n} + b) - \xi_n\} - \sum_{n=1}^{N}\beta_n\xi_n$$

Here $\alpha = [\alpha_1, \alpha_2, \dots, \alpha_N]$ and $\beta = [\beta_1, \beta_2, \dots, \beta_N]$ are the Lagrangian multipliers for the two inequalities in the original optimization problem.

To get dual problem containing only the dual variables α & β, we eliminate the primal variables $\boldsymbol{w}, b$ and ξ. Differentiating the Dual Lagrangian, we get:

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}} = 0 \implies \boldsymbol{w} = \sum_{n=1}^{N}\alpha_n y_n\boldsymbol{x_n}, \quad \frac{\partial\mathcal{L}}{\partial b} = 0 \implies \sum_{n=1}^{N}\alpha_n y_n = 0, \quad \frac{\partial\mathcal{L}}{\partial\xi_n} = 0 \implies C_{y_n} - \alpha_n - \beta_n = 0$$

Substituting these, we get the lagrangian $\mathcal{L}$ as:

$$\max_{0\leq\boldsymbol{\alpha}\leq C_{y_n},\boldsymbol{\beta}\geq 0} \mathcal{L}(\boldsymbol{\alpha},\boldsymbol{\beta}) = \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{m,n=1}^{N}\alpha_m\alpha_n y_m y_n(\boldsymbol{x_m}^T\boldsymbol{x_n}), \quad \text{subject to} \sum_{n=1}^{N}\alpha_n y_n = 0$$

Just like the original equation for soft margin SVM, this equation is also independent of $\boldsymbol{\beta}$.
The matrix notation of the dual problem will be given as:

$$\max_{0\leq\boldsymbol{\alpha}\leq C_{y_n}} \mathcal{L}_{\mathcal{D}}(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^T\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{G}\boldsymbol{\alpha}, \quad \text{subject to } \sum_{n=1}^{N}\alpha_n y_n = 0, \quad \text{where } \boldsymbol{G} = \alpha_n y_m y_n(\boldsymbol{x_m}^T\boldsymbol{x_n}).$$

In this modified dual lagrangian we have different trade-off hyperparameters for the two classes. For support vectors, the cost of mis-classification of a point of a one class will be larger than that of the other based on whether C_{+1} is greater or C_{-1}. This essentially changes the upper bound for α_n differently for different classes. Thus, the class having greater C can have a larger α. Doing this would shift the separating hyperplane more towards the class having lower C, so as to reduce miss-classifications of the other class.

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

Student Name: Sakshi
Roll Number: 180653
Date: November 27, 2020

QUESTION 4

Given objective function:

$$\mathcal{L} = \sum_{n=1}^{N}\sum_{k=1}^{K} z_{nk}\|\mathbf{x}_n - \mu_k\|^2$$

We are given that $\{\mu_k\}_{k=1}^{K}$ has been initialized randomly and we are reading one data point at a time. For a random point x_n we have to do the following:

1. assigning x_n "greedily" to the "best" cluster.
   $\implies$ This can be done by simply assigning x_n to the closest cluster center as is done in K-means algorithm.
   $$z_n = arg\,min_{k\in\{1,2,\ldots,K\}}\ \|\mathbf{x}_n - \mu_k\|^2$$
2. updating the cluster means using SGD on objective $\mathcal{L}$.
   $\implies$ The objective function is:
   $$\hat{\mu_k} = arg\,min_{\mu} \sum_{n:\hat{z_n}=k} \|\mathbf{x}_n - \mu_k\|^2$$
   Thus, $\mathcal{L}$ can be written as:
   $$\mathcal{L} = \sum_{k=1}^{K} z_k\|\boldsymbol{x_t} - \boldsymbol{\mu_k}\|^2$$
   Differentiating wrt μ_k, we get:
   $$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\mu_k} &= 2\sum_{n=1}^{K} z_k(\mu_k - \mathbf{x}_n)\\ &= 2(\mu_{z_k} - \mathbf{x}_n)\end{aligned}$$
   Thus, we get the SGD update equation for all k as:
   $$\begin{aligned}\mu_k^{(t+1)} &= \mu_k^{(t)} - 2\eta^{(t)}(\mu_k^{(t)} - \mathbf{x}_n^{(t)})\\ &= (1-2\eta^{(t)})\mu_k^{(t)} + 2\eta^{(t)}\mathbf{x}_n^{(t)}\end{aligned}$$
   This update equation shows that the new mean is a combination of old cluster mean and the new data added to the cluster. Keeping value of η high would make it tend more towards the new point. Thus, we should not let that happen once the number of points in the cluster increases.
   As we can see, we need to set an appropriate learning rate. A reasonable learning rate would be the inverse of total number of points in the cluster k. This is a good choice because, as the no. of points increases in a cluster the weight or importance of the newly added point reduces, thereby not causing much change in the cluster mean. So, the cluster means start to vary less as time(or no. of points in the cluster) increases.

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

Student Name: Sakshi
Roll Number: 180653
Date: November 27, 2020

QUESTION 5

The objective function of the K-Means algorithm as mentioned in class is,

$$\mathcal{L} = \sum_{n=1}^{N}\sum_{k=1}^{K} z_{nk}\|\mathbf{x}_n - \mu_k\|^2$$

We have to optimize this function to obtain the class means.

We replace the Euclidean distances in K-means algorithms with the kernelized form of distance,

$$d(\mathbf{x}_n, \mu_k) = \|\phi(\mathbf{x}_n) - \phi(\mu_k)\|$$

$$\begin{aligned}\|\phi(\mathbf{x}_n) - \phi(\mu_k)\|^2 &= \|\phi(\mathbf{x}_n)\|^2 + \|\phi(\mu_k)\|^2 - 2\phi(\mathbf{x}_n)^T\phi(\mu_k)\\ &= k(x_n, x_n) + k(\mu_k, \mu_k) - 2k(x_n, \mu_k)\end{aligned}$$

Here $k(.\ ,.)$ is the kernel function and ϕ is its infinitely dimensional feature mapping.

Here, $\phi(\mu_k)$ is the mean of ϕ mappings of the data points assigned to cluster k and is given by:

$$\phi(\mu_k) = \frac{1}{|\mathcal{C}_k|}\sum_{n:z_n=k}\phi(x_n)$$

ϕ is infinite dimensional, we cannot store the cluster means. We substitute the above derived equation in the distance equation:

$$\begin{aligned}\|\phi(\mathbf{x}_n) - \phi(\mu_k)\|^2 &= \|\phi(\mathbf{x}_n) - \frac{1}{|\mathcal{C}_k|}\sum_{i:z_i=k}\phi(\mathbf{x}_i)\|^2\\ &= \|\phi(\mathbf{x}_n) - \frac{1}{|\mathcal{C}_k|}\sum_{i:z_i=k}\phi(\mathbf{x}_i)\|^2\\ &= [\phi(\mathbf{x}_n) - \frac{1}{|\mathcal{C}_k|}\sum_{i:z_i=k}\phi(\mathbf{x}_i)]^T[\phi(\mathbf{x}_n) - \frac{1}{|\mathcal{C}_k|}\sum_{i:z_i=k}\phi(\mathbf{x}_i)]\\ &= k(\mathbf{x}_n, \mathbf{x}_n) - \frac{2}{|\mathcal{C}_k|}\sum_{i:z_i=k}k(\mathbf{x}_i, \mathbf{x}_n) + \frac{1}{|\mathcal{C}_k|^2}\sum_{j:z_j=k}\sum_{i:z_i=k}k(\mathbf{x}_j, \mathbf{x}_i)\end{aligned}$$

Thus, the above derived expression can be used to calculate the distance of a point from the cluster means in the ϕ feature map.

So the steps required for the kernel K-Means algorithm in this case is as follows:

1. **Initialization** : For initial cluster for each data point, assign the points randomly to one of the clusters. Also, compute the kernel matrix and store each entry in the $N \times N$ matrix. This step takes $\mathcal{O}(N^2D)$ time complexity.

2. **Cluster Assignment** : Following is how points are assigned to each cluster.

$$z_n = arg\,min_{k \in \{1,....,K\}} \|\phi(\mathbf{x}_n) - \phi(\mu_k)\|^2$$

   We have already derived above how this can be computed without storing the means. Thus, the above expression can be used for each assignment. It can be seen that the second term in the expression takes $\mathcal{O}(|C_k|)$ and third term takes $\mathcal{O}(|C_k|^2)$ time for one class. Overall this step will be done for $\sum_{i=1}^{K} \mathcal{O}(|C_k|^2)$ times. Thus, the overall time complexity becomes $\mathcal{O}(N^2)$ for assigning class to one point. This is to be done for all N points. Thus, complete time complexity is $\mathcal{O}(N^3)$.

3. **Mean Computation** : We do not need to compute and store the cluster means in this case as expression for finding euclidean distance from cluster means has been replaced by the above derived expression. Thus, the means change automatically as the cluster assignment changes.

This case differs from the standard K-means algorithm as we compute and store the cluster means in the standard K-means algorithm whereas in this case, the feature mapping is infinite-dimensional making it impossible to store the cluster means. Thus, we derive the distance of points from cluster means using definition of kernels and don't need to store the cluster means. But finding the cluster means for each step turns out to be quite expensive in terms of time complexity.

For standard K-Means the time complexity comes out to be $\mathcal{O}(NKD)$ for each iteration but for the kernel K-means algorithm shown above the time complexity is $\mathcal{O}(N^2 * max(D, N))$ for each iteration. Thus, the standard K-Means is much faster than the algorithm mentioned above.

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

Student Name: Sakshi
Roll Number: 180653
Date: November 27, 2020

QUESTION 6

Red points : positive class and Blue points : negative class.
SVM part has been implemented using Scikit Learn.

Part 1: (binclass.txt)

1. Given different variances σ_+ & σ_-
   $\sigma_- = 4.359$ & $\sigma_+ = 7.295$

   $\mu_- = \begin{bmatrix} 20.325 \\ 9.688 \end{bmatrix}$ $\mu_+ = \begin{bmatrix} 10.011 \\ 19.550 \end{bmatrix}$

   Plot:

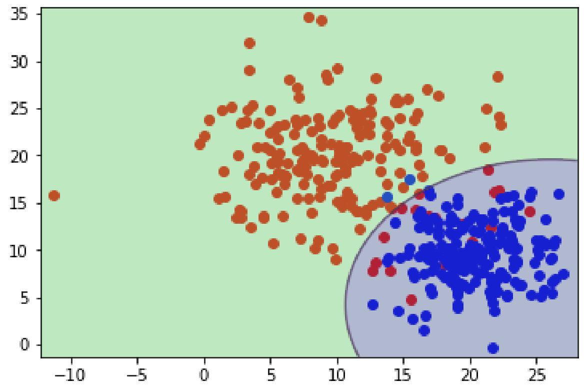

Figure 1: Generative model with quadratic boundary for binclass.txt

2. Same σ
   Plot:

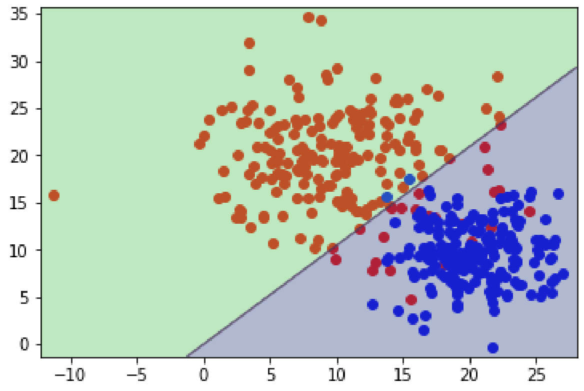

Figure 2: Generative model with linear boundary for binclass.txt

3. SVM classifier (linear boundary) Plot:

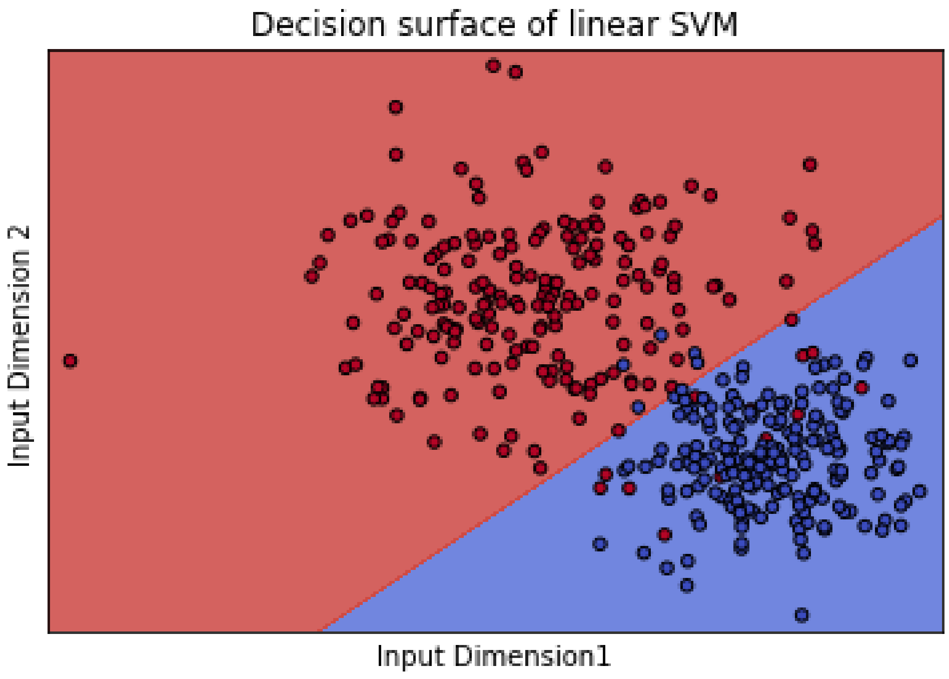

Figure 3: SVM model with linear boundary for binclass.txt

Part 2: (binclassv2.txt)

1. Given different variances σ_+ & σ_-
   $\sigma_- = 4.359$ & $\sigma_+ = 10.713$

   $$\mu_- = \begin{bmatrix} 20.325 \\ 9.688 \end{bmatrix} \quad \mu_+ = \begin{bmatrix} 10.575 \\ 18.557 \end{bmatrix}$$

   Plot:

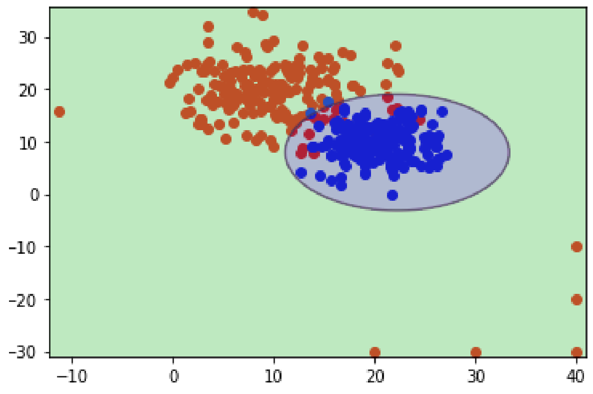

Figure 4: Generative model with quadratic boundary for binclassv2.txt

2. Same σ. Plot:

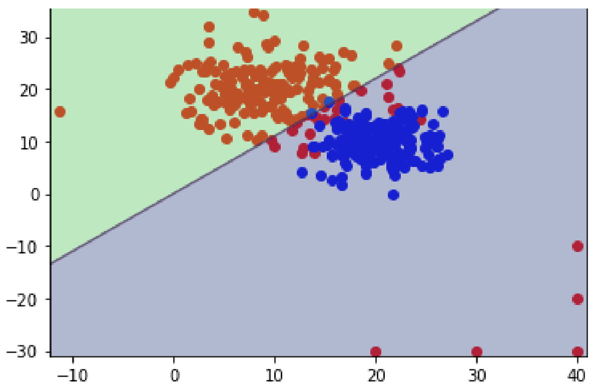

Figure 5: Generative model with linear boundary for binclassv2.txt

3. SVM classifier (linear boundary) Plot:

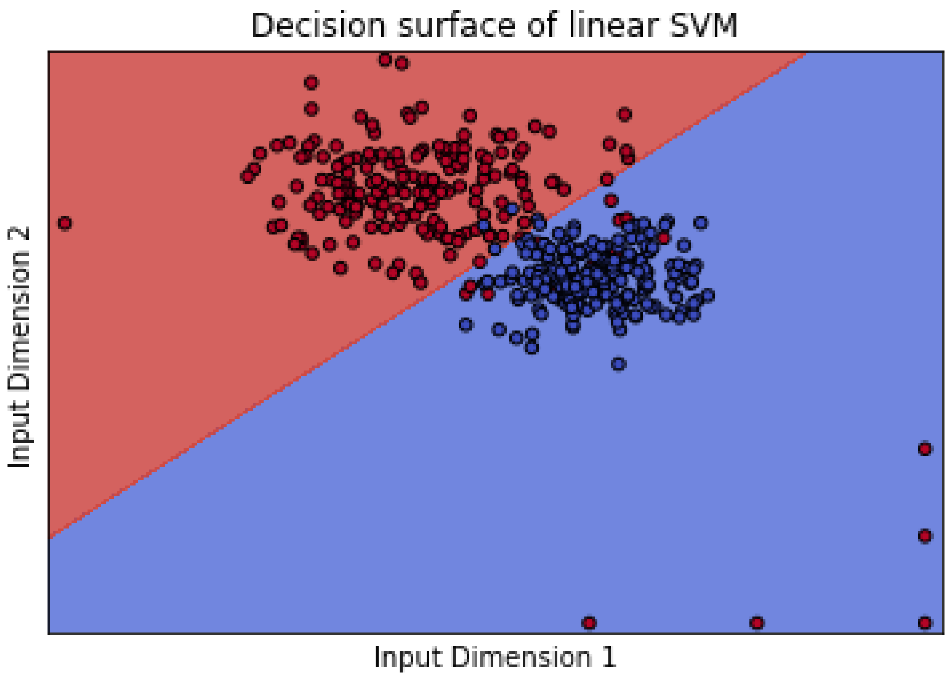

Figure 6: SVM model with linear boundary for binclassv2.txt

Conclusion: For the first dataset (binclass.txt), Generative classification works better whereas for the second dataset (binclassv2.txt), where there are many outliers from positive (red) class, SVM works better. Thus, SVM works well with outliers also.