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

Student Name: Nidhi Hegde
Roll Number: 180472
Date: November 28, 2020

QUESTION 1

The loss function for the logistic regression model is:

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

Differentiating w.r.t $\mathbf{w}$, we get the gradient as:

$$\mathbf{g} = \frac{\partial L}{\partial \mathbf{w}} = -\sum_{n=1}^{N}\left(y_n \mathbf{x_n} - \frac{e^{\mathbf{w^T x_n}}}{1 + e^{\mathbf{w^T x_n}}}\mathbf{x_n}\right) = -\sum_{n=1}^{N}\left(y_n - \frac{e^{\mathbf{w^T x_n}}}{1 + e^{\mathbf{w^T x_n}}}\right)\mathbf{x_n}$$

Differentiating w.r.t $\mathbf{w}$ again and using chain rule,after simplification, we get the Hessian as an N X N matrix:

$$\mathbf{H} = \frac{\partial^2 L}{\partial \mathbf{w^2}} = \sum_{n=1}^{N}\frac{e^{\mathbf{w^T x_n}}}{(1 + e^{\mathbf{w^T x_n}})^2}\mathbf{x_n x_n^T}$$

Replacing these values in the second-order update equation for the given problem, we get:

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

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \left(\sum_{n=1}^{N}\frac{e^{\mathbf{w^T x_n}}}{(1 + e^{\mathbf{w^{(t)T} x_n}})^2}\mathbf{x_n x_n^T}\right)^{(t)^{-1}}\sum_{n=1}^{N}\left(y_n - \frac{e^{\mathbf{w^{(t)T} x_n}}}{1 + e^{\mathbf{w^{(t)T} x_n}}}\right)\mathbf{x_n} \qquad -(1)$$

The above update, given by Newton's method is equivalent to :

$$\arg\min_{w}\; L(\mathbf{w^{(t)}}) + \mathbf{g}^{(t)^T}(\mathbf{w} - \mathbf{w^{(t)}}) + \frac{1}{2}(\mathbf{w} - \mathbf{w^{(t)}})^T\mathbf{H}^{(t)}(\mathbf{w} - \mathbf{w^{(t)}})$$

Replacing values, we get the problem :

$$\arg\min_{w} -\sum_{n=1}^{N}(y_n \mathbf{w^{(t)T} x_n} - \log(1 + e^{\mathbf{w^{(t)T} x_n}})) - \sum_{n=1}^{N}\left(y_n - \frac{e^{\mathbf{w^{(t)T} x_n}}}{1 + e^{\mathbf{w^{(t)T} x_n}}}\right)\mathbf{x_n}^T(\mathbf{w} - \mathbf{w^{(t)}})$$
$$+\frac{1}{2}(\mathbf{w} - \mathbf{w^{(t)}})^T\left(\sum_{n=1}^{N}\frac{e^{\mathbf{w^{(t)T} x_n}}}{(1 + e^{\mathbf{w^{(t)T} x_n}})^2}\mathbf{x_n x_n^T}\right)(\mathbf{w} - \mathbf{w^{(t)}})$$

We need to show that the above problem is equivalent to solving the importance-weighted regression problem $\mathbf{w^{(t+1)}} = argmin_w \sum_{n=1}^{N}\gamma_n^{(t)}(\hat{y_n}^{(t)} - \mathbf{w^T x_n})^2 \qquad -(2)$
Thus, when we simplify the above problem by completing the squares, we get a form similar to (2), where on comparing equivalent terms, we get:

$$\gamma_n^{(t)} = \frac{\sigma_n^{(t)}(1-\sigma_n^{(t)})}{2}$$

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

$$where \;\; \sigma_n^{(t)} = \frac{e^{\mathbf{w^{(t)^T}x_n}}}{1 + e^{\mathbf{w^{(t)^T}x_n}}}$$

The expression for $\gamma_n^{(t)}$ makes intuitive sense here, since it gives more weight to those examples which the model is not confident on (points where model predicts both classes with equal confidence). This can be seen as $\gamma_n^{(t)}$ takes the maximum value for a point whose $\sigma_n^{(t)} = 0.5$, and hence such a point is given more weightage, in the argmin problem.
Hence, in order to minimize the contribution of that term in the argmin problem, the model learns to minimize the other part of the term $(\hat{y_n}^{(t)} - \mathbf{w^T x_n})^2$, which brings the point closer to the desired class in a way.

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

Student Name: Nidhi Hegde
Roll Number: 180472
Date: November 28, 2020

QUESTION 2

Since perceptron learns linear boundaries given by $y = w^T x + b$, we can kernelize it using the kernel k and new feature map $\phi(.)$, and we find a linear boundary in this new space, which gives us a non-linear boundary in the original space.
Applying the standard Perceptron algo in the new space gives us:

1. Initialization
$$\mathbf{w^{(0)}} = 0, \quad t = 0, \quad \eta_t = 1 \ \forall t$$
2. Pick a random (x_n, y_n)
3. If current $\mathbf{w}$ makes a mistake according to the **Mistake Condition**
$$y_n \mathbf{w^{(t)^T}} \phi(x_n) \leq 0$$
4. If **true**, then, perform the following update, using the **Update Equation**
$$\mathbf{w^{(t+1)}} = \mathbf{w^{(t)}} + y_n \phi(x_n)$$
5. If not converged, return to Step 2

At any point, we have $\mathbf{w^{(t)}}$ of the form $\sum y_j \phi(x_j)$.
Hence, we do not explicitly need to compute $\phi(x_n)$ in step 3. Instead we can use a condition $\sum y_j K(x_j, x_n) < 0$ as an equivalent mistake condition replacing step 3. Here, the summation is over the j's present in the weight vector.
Similarly, we need not compute the new weight vector at any time. Instead, we simply store the number of times any training example appears in the expression for $\mathbf{w}$.
The modified perceptron algorithm can be given as:

1. Given: K(an NXN matrix) and (x_n, y_n) $\forall n \in [1, N]$
2. Initialization: $t = 0, \quad \eta_t = 1 \ \forall t$, an array $\mathbf{A}$ of size N, initialized as $\bar{\mathbf{0}}$
3. Pick a random (x_n, y_n)
4. **Mistake Condition**
$$\sum_{j=1}^{N} A\left[j\right] . y_j . K(x_j, x_n) \leq 0$$
5. If mistake condition is **true**, then perform update as:
$$A[n] = A[n] + 1$$
$$t = t + 1$$
6. If not converged, return to Step 2

At any point in the above algorithm, the weight vector can be explicitly given as : $\sum_{j=1}^{N} A\left[j\right] y_j \phi(x_j)$.
However, in order to find a prediction for a new point, we can simply plug in the point by replacing '.' in the expression $\sum_{j=1}^{N} A\left[j\right] y_j K(x_j, .)$

Student Name: Nidhi Hegde
Roll Number: 180472
Date: November 28, 2020

QUESTION 3

The Lagrangian problem corresponding to the above SVM can be written as:

$$\max_{\boldsymbol{\alpha}\geq 0,\boldsymbol{\beta}\geq 0} \arg\min_{\mathbf{w},b,\boldsymbol{\xi}} L(\mathbf{w},b,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta}) \qquad where$$

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

Here, $\boldsymbol{\alpha}$ is the lagrange's multiplier for n constraints of the form $y_n(\mathbf{w^T x_n}+b) \geq 1-\xi_n \ \forall n$, while $\boldsymbol{\beta}$ is used for constraints of the form $\xi_n \geq 0 \ \forall n$.
Taking derivative w.r.t. the primal variables $\mathbf{w}, b, \boldsymbol{\xi}$, we get:

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \implies \mathbf{w} = \sum_{n=1}^{N}\alpha_n y_n \mathbf{x_n}$$

$$\frac{\partial L}{\partial b} = 0 \implies \sum_{n=1}^{N}\alpha_n y_n = 0$$

$$\frac{\partial L}{\partial \xi_n} = 0 \implies \begin{cases} C_{+1}-\alpha_n-\beta_n = 0 & \text{if } y_n = 1 \\ C_{-1}-\alpha_n-\beta_n = 0 & \text{if } y_n = -1 \end{cases}$$

Here, since $\beta_n \geq 0 \ \forall n$, the last relation gives us:

$$\begin{cases} \alpha_n \leq C_{+1} & \text{if } y_n = 1 \\ \alpha_n \leq C_{-1} & \text{if } y_n = -1 \end{cases}$$

On substituting $\mathbf{w}$ and $C_{+/-1}$ in terms of dual variables α and β, the ξ_n terms nullify, and hence dual variable β disappears from the problem, leading to the form of problem as:

$$\max_{\alpha_n\geq 0} \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}\alpha_n\alpha_m y_n y_m \mathbf{x_n^T x_m}$$

$$subject\ to \quad \begin{cases} \alpha_n \leq C_{+1} & \text{if } y_n = 1 \\ \alpha_n \leq C_{-1} & \text{if } y_n = -1 \end{cases} \quad \forall n \in [1,N]$$

$$and \qquad \sum_{n=1}^{N}\alpha_n y_n = 0$$

This problem differs from the standard SVM dual problem in the constraints, as can be seen above. Here, the value of α_n is constrained depending on the class the point belongs to.
INTERPRETATION OF C:
We know that the hyper-parameter C controls the trade off between large margin and small

training error. Here, since C is different for both the classes, it implies that we can weigh the balance of the training error vs large margin for both the classes differently. That means, we can punish a model more if it wrongly classifies a particular class by employing a larger C for that class, weighing the training loss for that class more, and thus learning a smaller margin for that class.
This is useful in those situations where having a positive outlier in the negative class is considered much worse than having a negative outlier in the positive class, or vice-versa.

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

Student Name: Nidhi Hegde
Roll Number: 180472
Date: November 28, 2020

QUESTION 4

Step 1:
We choose a point x_n randomly. We greedily assign this point to the best cluster, i.e. we compare the Euclidean distance of the new point to each cluster center, and assign it the cluster with minimum Euclidean distance. Thus, the cluster center assigned is:

$$k = argmin_{\mu_k, k \in [1,K]} \ \|x_n - \mu_k\|^2$$

Step 2:
Taking derivative w.r.t μ_k, we get:

$$\frac{\partial L}{\partial \mu_k} = -\sum_{n=1}^{N} 2 z_{nk}(x_n - \mu_k)$$

Thus, the gradient descent step takes the form:
(η_k is the learning rate for updation of cluster k's mean):

$$\mu_k^{(t+1)} = \mu_k^{(t)} + \eta_k \sum_{n=1}^{N} 2 z_{nk}(x_n - \mu_k^{(t)})$$

INTERPRETATION OF UPDATE:
On, expanding, we get the following update equation:
(Here, n_k represents the number of points in cluster k):

$$\mu_k^{(t+1)} = \mu_k^{(t)}(1 - 2\eta_k.n_k) + 2\eta_k \sum_{n=1}^{N} z_{nk} x_n$$

Since, z_{nk} takes the value 1 for all points belonging to the cluster k, the second term is the sum of all the points belonging to cluster k, multiplied by $2\eta_k$.
Intuitively, the update tries to increase the contribution of the points belonging to cluster k in the expression of μ_k. If we replace $2\eta_k$ by ζ, then the update is of the form:

$$\mu_k^{(t)}(1 - \zeta.n_k) + \zeta \sum_{n=1}^{N} z_{nk} x_n$$

Here, it is clearer that some portion of the original mean ($\zeta.n_k\mu_k^{(t)}$) is replaced by a term $\zeta \sum_{n=1}^{N} z_{nk} x_n$. This term is nothing but a constant(less than 1) multiplied by true mean of that cluster at that point.Thus, this update increases the contribution of the second term($\zeta \sum_{n=1}^{N} z_{nk} x_n$) in $\mu_k^{(t)}$, thus, bringing it closer to the true mean.

Step 3:
If we use a value $\eta_k = \frac{1}{2n_k}$, the equation reduces to:

$$\mu_k^{(t+1)} = \mu_k^{(t)}(1 - 1) + 2\eta_k \sum_{n=1}^{N} z_{nk} x_n \implies \frac{1}{n_k} \sum_{n=1}^{N} z_{nk} x_n$$

which is nothing, but the mean of points belonging to the k^{th} cluster. This is a good choice since μ_k accurately captures the mean of its cluster, after the update, for a given z matrix.

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

Student Name: Nidhi Hegde
Roll Number: 180472
Date: November 28, 2020

QUESTION 5

Initialization:
Randomly assign all points to any of the k clusters. For every point x_n assigned to a cluster, maintain an associated field c_{x_n} which is a value in the range $[1, k]$, storing the cluster number that the point belongs to.
We also maintain a list of points for every cluster k in the list $L(k)\ \forall k \in [1, K]$. (If the dimension of training points is large, then we can store the list of pointers to the training points belonging to that cluster instead to avoid space wastage)

Cluster assignment:
Assign a cluster to every training point, according to:

$$c_{x_n} = \underset{k\in[1,K]}{\arg\min}\ \|\phi(x_n) - \mu_k\|^2 \tag{1}$$

We do not explicitly store μ_k. But, we store the list of points belonging to that cluster, for every cluster. Thus, μ_k can be written as:

$$\mu_k = \frac{\sum_{x_j\in L(k)}\ \phi(x_j)}{\sum_{i=1|c_{x_i}=k}^{N}\ 1} = \frac{\sum_{x_j\in L(k)}\ \phi(x_j)}{n_k}$$

Here, the denominator n_k is the number of points belonging to that cluster.
Now, we need to prove that the means stored in the form of list of points for every cluster can be used to efficiently obtain a solution for (1).

$$\|\phi(x_n) - \mu_k\|^2 = \phi(x_n)^T\phi(x_n) - 2\phi(x_n)\mu_k + \mu_k^T\mu_k$$

$$= \phi(x_n)^T\phi(x_n) - 2\phi(x_n)^T\left(\frac{\sum_{x_j\in L(k)}\ \phi(x_j)}{n_k}\right) + (\frac{\sum_{x_j\in L(k)}\ \phi(x_j)}{n_k})^T(\frac{\sum_{x_j\in L(k)}\ \phi(x_j)}{n_k})$$

$$= \phi(x_n)^T\phi(x_n) - \frac{2}{n_k}\left(\sum_{x_j\in L(k)}\ \phi(x_n)^T\phi(x_j)\right) + \frac{1}{n_k^2}.\left(\sum_{x_i\in L(k)}\sum_{x_j\in L(k)}\ \phi(x_i)^T\phi(x_j)\right)$$

$$= \boldsymbol{\kappa}(x_n, x_n) - \frac{2}{n_k}.\left(\sum_{x_j\in L(k)}\ \boldsymbol{\kappa}(x_n, x_j)\right) + \frac{1}{n_k^2}.\left(\sum_{x_i\in L(k)}\sum_{x_j\in L(k)}\ \boldsymbol{\kappa}(x_i, x_j)\right)$$

Here, $\boldsymbol{\kappa}$ denotes the kernel matrix.
Hence, storing the means μ_k in this form lets us compute the Euclidean Distance, without any need of storing the means explicitly.

Mean computation:
We free the original lists from $L(1)$ to $L(K)$, and iterate through all the training examples from $n = 1$ to N. For every training point x_n encountered, we add it to the list $L(c_{xn})$, and proceed further.

After this step, we have the lists $L(1)$ to $L(K)$ storing the newly assigned clusters in step 2.
If the algorithm does not converge, then we move back to step 2. (We can use a flag to determine this, in step 2. If any point is not assigned to its old cluster, we set the value of this flag to be 1. In any iteration of the algorithm, we reset this flag to 0 at the starting of Step 2. Convergence is implied by flag = 0 after processsing all inputs in step 2).

Comparison with standard K-Means:
Standard K-Means algorithm takes $O(KD)$ time to compute the distance of any point from all the cluster centers, and hence a total of $O(NKD)$ time for all the points (i.e. step 2).
However here, the means are not stored explicitly which leads to a larger computation time, for assigning a cluster to each input (in step 2).
Assuming, we have already computed the kernel matrix $\boldsymbol{\kappa}$, the distance computation as shown above requires $O\left(N + (\sum_{k=1}^{K} n_k^2)\right)$ time for every training point, to calculate its distance from all the K cluster centers.
Since we have a constraint $\sum_{k=1}^{K} n_k = N$, in the worst case, all the points would belong to one cluster, leading to a computation time of $O(N^2)$ for computing distance of any training point from that cluster mean, as opposed to $O(D)$ in the standard K-Means. Hence in this case, Step 2 would take $O(N^3)$ time, involving a computation time of $O(N^2)$ for any one training point.

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

Student Name: Nidhi Hegde
Roll Number: 180472
Date: November 28, 2020

QUESTION
6

- **Generative classification(different σ)**
  The Gaussian class conditionals for the 2 classes can be given as:

$$\mathcal{N}(\mathbf{x}|\mu_+, \sigma_+^2\mathbf{I}_2) = \frac{1}{2\pi\sigma_+^2} e^{-\frac{\|\mathbf{x}-\mu_+\|^2}{2\sigma_+^2}}$$
$$\mathcal{N}(\mathbf{x}|\mu_-, \sigma_-^2\mathbf{I}_2) = \frac{1}{2\pi\sigma_-^2} e^{-\frac{\|\mathbf{x}-\mu_-\|^2}{2\sigma_-^2}}$$

  It is given that the prior probabilities for both the classes are 0.5 each. We can write the likelihood function for the data as:

$$p(\mathbf{y}|\mu_+, \mu_-, \sigma_+^2, \sigma_-^2) = \prod_{n=1}^{N} \left[0.5\mathcal{N}(\mathbf{x}|\mu_+, \sigma_+^2\mathbf{I}_2)\right]^{\frac{1+y_n}{2}} \left[0.5\mathcal{N}(\mathbf{x}|\mu_-, \sigma_-^2\mathbf{I}_2)\right]^{\frac{1-y_n}{2}}$$

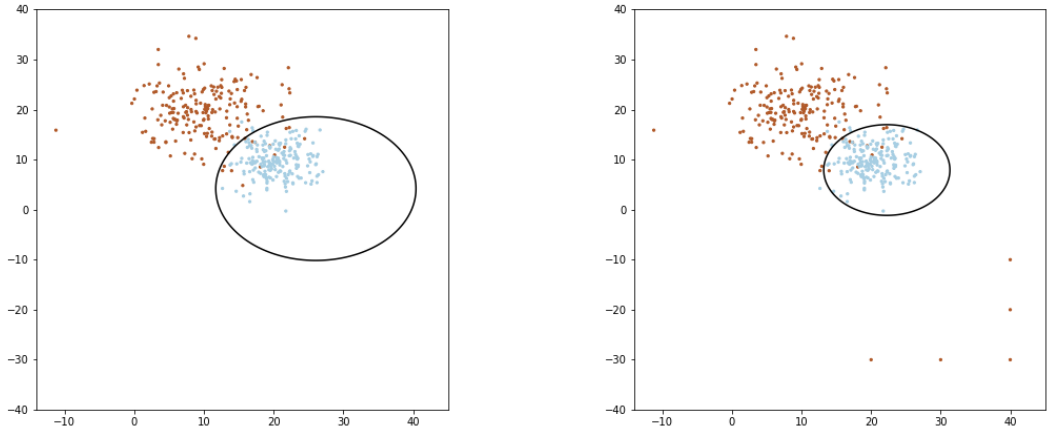
  Taking log and maximizing the likelihood, we find the MLE estimates for all the parameters as follows:

$$\mu_+ = \frac{\sum_{y_n=+1} x_n}{N_+}$$
$$\sigma_+^2 = \frac{\sum_{y_n=+1} \|x_n - \mu_+\|^2}{N_+}$$
$$\mu_- = \frac{\sum_{y_n=-1} x_n}{N_-}$$
$$\sigma_-^2 = \frac{\sum_{y_n=-1} \|x_n - \mu_-\|^2}{N_-}$$

  Any point is said to lie on the decision boundary when the probability of belonging to both the classes is equal. Thus, the learned decision boundary is given by equating both the Gaussians, as:

$$\frac{\|\mathbf{x}-\mu_+\|^2}{\sigma_+^2} - \frac{\|\mathbf{x}-\mu_-\|^2}{\sigma_-^2} = 2\log\left(\frac{\sigma_-}{\sigma_+}\right)$$

The figure 1 represents the decision boundary(Note that the plot is scaled differently along both the axes)



(a) 'binclass.txt' (b) 'binclassv2.txt'

Figure 1: Generative classification with different σ

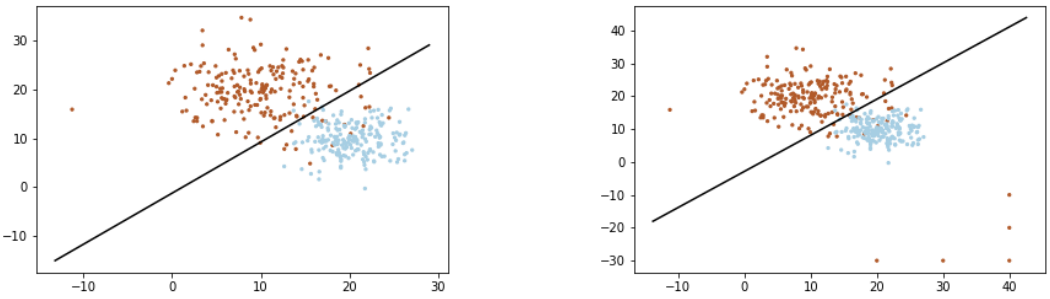
- **Generative classification(equal σ)**
  In case σ is same for both the classes, we get the estimate for σ^2 as:

$$\sigma^2 = \frac{\sum_{y_n=+1} \|x_n - \mu_+\|^2 + \sum_{y_n=-1} \|x_n - \mu_-\|^2}{N}$$

  The estimates for μ_- and μ_+ remain the same as the first part.
  The decision boundary simplifies to:

$$2\left(\mu_+ - \mu_-\right)^T \mathbf{x} - \left(\|\mu_+\|^2 - \|\mu_-\|^2\right) = 0$$

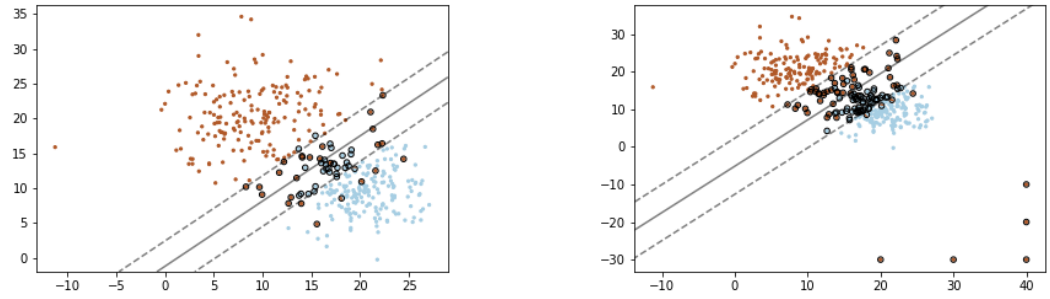
  which is linear.



(a) 'binclass.txt' (b) 'binclassv2.txt'

Figure 2: Generative classification with equal σ

- **SVM:** Plotted using scikit learn - Refer figure 3



(a) 'binclass.txt' (b) 'binclassv2.txt'

Figure 3: SVM plots using scikit-learn

Note that on experimentation, we obtain the following values for both the datasets:

Parameter	**binclass.txt**	**binclassv2.txt**
μ_+	(10.01,19.54)	(10.57,18.55)
σ_+^2	53.21	114.77
μ_-	(20.32,9.68)	(20.32,9.68)
σ_-^2	19.00	19.00
σ^2	36.10	66.88

Observations:

1. **Dataset: binclass.txt**
   Both generative methods and SVM seem to work in this case.

2. **Dataset: binclassv2.txt**
   Here, SVM's decision boundary does not vary much as compared to its boundary for *binclass.txt*. Generative classification, however in this case learns a very restricted boundary(Figure 1, part (b)) as compared to the boundary it learnt on other dataset. Hence, we observe that here, generative classification learns a boundary which may overfit on the training data.

We know that modelling the inputs using generative classification works well in case the shape of the classes is not uniform (let's say elongated along one direction). However, its use should be avoided in case "strong" outliers are present in the data. In these cases, it is preferrable to use SVM's since they still learn a well-balanced boundary since the presence of outliers doesn't seem to affect the decision boundary learnt by SVM's much.