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

Student Name: Utkarsh Gupta
Roll Number: 180836
Date: November 24, 2020

QUESTION 1

The Logistic Regression loss function is given by:

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

$$\implies L(\mathbf{w}) = -\sum_{n=1}^{N}(y_n \log(\sigma(\mathbf{w}^T \mathbf{x}_n)) + (1-y_n)\log(1-\sigma(\mathbf{w}^T \mathbf{x}_n)))$$

Taking derivative with respect to $\mathbf{w}$ we get the gradient $\mathbf{g}^{(t)}$ as:

$$\mathbf{g}^{(t)} = \sum_{n=1}^{N}(-y_n \mathbf{x}_n(1-\sigma(\mathbf{w}^T \mathbf{x}_n)) + (1-y_n)\mathbf{x}_n \sigma(\mathbf{w}^T \mathbf{x}_n)) = \sum_{n=1}^{N}(\mathbf{x}_n(\sigma(\mathbf{w}^T \mathbf{x}_n) - y_n)) = \mathbf{X}^T(\hat{\mathbf{y}} - \mathbf{y})$$

Hessian is nothing but the second derivative of $\mathbf{g}^{(t)}$:

$$\mathbf{H}^{(t)} = \nabla^2 L(\mathbf{w}) = \sum_{n=1}^{N}(\frac{\partial \mathbf{g}^{(t)}}{\partial \mathbf{w}^T}) = \sum_{n=1}^{N}(\mathbf{x}_n \mathbf{x}_n^T \sigma(\mathbf{w}^T \mathbf{x}_n)(1-\sigma(\mathbf{w}^T \mathbf{x}_n)))$$

This is equivalent to concatenating column vectors $\mathbf{x}_v \in \mathbb{R}^D$ into a matrix X of size $D \times N$ such that $\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^T = \mathbf{X}\mathbf{X}^T$. The scalar terms are combined in a diagonal matrix $\mathbf{R}$ such that $R_{ii} = \sigma(\mathbf{w}^T \mathbf{x}_i)(1-\sigma(\mathbf{w}^T \mathbf{x}_i))$.

$$\mathbf{H}^{(t)} = \mathbf{X}\mathbf{R}^{(t)}\mathbf{X}^T$$

Substituting the Hessian and Gradient in the weight update equation we get, (so far we wrote $\mathbf{w}^{(t)}$ as $\mathbf{w}$):

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - (\mathbf{X}\mathbf{R}^{(t)}\mathbf{X}^T)^{-1}\mathbf{X}^T(\hat{\mathbf{y}} - \mathbf{y})$$

$$\implies \mathbf{w}^{(t+1)} = (\mathbf{X}\mathbf{R}^{(t)}\mathbf{X}^T)^{-1}\mathbf{X}^T\mathbf{R}^{(t)}(\mathbf{X}\mathbf{w}^{(t)} - \mathbf{R}^{-1}(\hat{\mathbf{y}} - \mathbf{y})) \tag{1}$$

The solution obtained looks suspiciously familiar. The given weighted least squares problem can be written as ($\mathbf{C}^{(t)}$ is a diagonal matrix with $C_{nn} = \gamma_n^{(t)}$):

$$\mathbf{w}^{(t+1)} = \arg\min_{\mathbf{w}} \sum_{n=1}^{N} \gamma_n^{(t)}(\hat{y}^{(t)} - \mathbf{w}^T \mathbf{x}_n)^2 = \arg\min_{\mathbf{w}}(\hat{\mathbf{Y}}^{(t)} - \mathbf{X}\mathbf{w})^T \mathbf{C}^{(t)}(\hat{\mathbf{Y}}^{(t)} - \mathbf{X}\mathbf{w})$$

Taking the derivative and setting it to 0, we get:

$$\mathbf{w}^{(t+1)} = (\mathbf{X}^T\mathbf{C}^{(t)}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{C}^{(t)}\hat{\mathbf{Y}}^{(t)} \tag{2}$$

Comparing 2 with 1, we get:

$$\boxed{\mathbf{C}^{(t)} = \mathbf{R}^{(t)} \implies \gamma_n^{(t)} = \sigma(\mathbf{w}^{(t)T}\mathbf{x}_n)(1 - \sigma(\mathbf{w}^{(t)T}\mathbf{x}_n))}$$

$$\boxed{\hat{\mathbf{Y}}^{(t)} = (\mathbf{X}\mathbf{w}^{(t)} - \mathbf{R}^{-1}(\hat{\mathbf{y}} - \mathbf{y})) \implies \hat{y}_n^{(t)} = \mathbf{w}^{(t)T}\mathbf{x}_n - \frac{\sigma(\mathbf{w}^{(t)T}\mathbf{x}_n) - y_n}{\sigma(\mathbf{w}^{(t)T}\mathbf{x}_n)(1 - \sigma(\mathbf{w}^{(t)T}\mathbf{x}_n))}}$$

We know that in case of Logistic Regression $\sigma(\mathbf{w}^{(t)T}\mathbf{x}_n)$ represents the probability that an input $\mathbf{x}_n$ belongs to the "0" class. If we let $\sigma(\mathbf{w}^{(t)T}\mathbf{x}_n) = p_n$ then $\gamma_n^{(t)} = p_n(1 - p_n)$, which attains it's maximum value at $p_n = \frac{1}{2}$, this implies that the loss function gives more weightage to those examples that lie close to the decision boundary. Thus, the expression intuitively makes sense.

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

Student Name: Utkarsh Gupta
Roll Number: 180836
Date: November 24, 2020

QUESTION 2

We know that the weights obtained from perceptron algorithm can be written as;

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$$

where α_i is the number of times $\mathbf{x}_i$ was missclassified, and hence, caused $\mathbf{w}$ to update. Substituting this result in the original perceptron algorithm we get:

$$\hat{y} = sgn(\mathbf{w}^T\mathbf{x}) = sgn((\sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i)^T\mathbf{x})$$
$$= sgn(\sum_{i=1}^{N} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}))$$

Since we have an inner product, we can apply the Kernel trick. The inner product can now be replaced directly thereby, taking benefit of the Feature Map ϕ without explicitly calculating it.

$$\hat{y} = sgn(\sum_{i=1}^{N} \alpha_i y_i \mathbf{K}(\mathbf{x}_i, \mathbf{x}))$$

The algorithm below gives the required initialization, mistake condition and update equation. K is the number of iterations for which we want the algorithm to run. $\mathbf{X}$ and $\mathbf{Y}$ correspond to the input features and outputs respectively.

Algorithm 1: Kernel Perceptron

Function `K_Perceptron(`$\mathbf{X}, \mathbf{Y}, K$`):`
- `// Initialization`
- $\boldsymbol{\alpha}$:= zero N x 1 vector
- **for** $k = 1$ *to* K **do**
  - **for** $j = 1$ *to* N **do**
    - $\hat{y} = sgn(\sum_{i=1}^{N} \alpha_i y_i \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j))$
    - `// Mistake Condition`
    - **if** $\hat{y} \neq y_j$ **then**
      - `// Update equation`
      - $\alpha_j = \alpha_j + 1$
  - **end**
- **end**

return
repeat
- `K_Perceptron(`$\mathbf{X}, \mathbf{Y}, K$`)`

until *convergence*

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

Student Name: Utkarsh Gupta
Roll Number: 180836
Date: November 24, 2020

QUESTION 3

The primal formulation of the Imbalanced Class SVM is:

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

subjected to $y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n$ and $\xi_n \geq 0\ \forall n$.
The Langrangian is given by:

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

Taking partial derivatives of $\mathcal{L}$ w.r.t. $\mathbf{w}$, b and ξ_n and setting to zero gives:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{w}} = 0 \implies \boxed{\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n}\ ,\ \frac{\partial\mathcal{L}}{\partial b} = 0 \implies \boxed{\sum_{n=1}^{N} \alpha_n y_n = 0}\ ,\ \frac{\partial\mathcal{L}}{\partial\xi_n} = 0 \implies \boxed{C_{y_n} - \alpha_n - \beta_n = 0}$$

Using $\beta_n \geq 0$ and $C_{y_n} - \alpha_n - \beta_n = 0$, we get: $0 \leq \alpha_n \leq C_{y_n}$.
Substituting these in the original $\mathcal{L}$, we get the Dual Langrangian:

$$\boxed{\max_{\boldsymbol{\alpha}\leq\mathbf{C},\boldsymbol{\beta}\geq\mathbf{0}} \mathcal{L}_D(\boldsymbol{\alpha},\boldsymbol{\beta}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{m,n=1}^{N} \alpha_m\alpha_n y_m y_n (\mathbf{x}_m^T\mathbf{x}_n) \quad s.t. \quad \sum_{n=1}^{N} \alpha_n y_n = 0}$$

$$\boxed{\max_{\boldsymbol{\alpha}\leq\mathbf{C}} \mathcal{L}_D(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^T\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{G}\boldsymbol{\alpha} \quad s.t. \quad \sum_{n=1}^{N} \alpha_n y_n = 0}$$

$\mathbf{C}$ is a Nx1 vector with n^{th} entry being C_{y_n}. As observed earlier in case of soft-margin SVM, the dual variables $\boldsymbol{\beta}$ do not affect the Dual Problem. The $\boldsymbol{\alpha}$ vector will be sparse as well (KKT conditions) and non-zero α_n will correspond to the support vectors.

In the standard SVM, $\mathbf{C}$ is a vector having all values as C (a fixed scalar). This performs well for balanced classes but as the data becomes imbalanced, the ratio between the positive and negative support vectors also becomes imbalanced.

This causes the separating hyperplane to be skewed towards the minority class, which finally yields a suboptimal model. In the new problem setting, the misclassification cost vector $\mathbf{C}$ is redefined. By assigning a higher misclassification cost for the minority class examples than the majority class examples, the aforementioned effect of class imbalance can be reduced. Let's assume that negative examples are in minority, then if we make $C_{\mathbf{-1}} > C_{\mathbf{+1}}$, we assign a higher cost of misclassification to negative class example. Thus, we can achieve the desired effect.

Student Name: Utkarsh Gupta
Roll Number: 180836
Date: November 24, 2020

QUESTION 4

In order to perform step 1, we can calculate distances from all the cluster means and then assign the incoming point $\mathbf{x}_n$ to the cluster whose cluster mean has the least distance from the incoming point. In other words, we can solve the following optimization problem:

$$\hat{k} = \arg\min_k ||\boldsymbol{\mu_k} - \mathbf{x}_n||^2$$

After we perform Step 1 in an online setting, the gradient required for update will be with respect to $\mathbf{x}_n$ only:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu}_{\hat{\boldsymbol{k}}}} = -2(\mathbf{x}_n - \boldsymbol{\mu}_{\hat{\boldsymbol{k}}})$$

Thus, update equation we use will be:

$$\boldsymbol{\mu}_{\hat{\boldsymbol{k}}}^{(t+1)} = (1-\eta)\boldsymbol{\mu}_{\hat{\boldsymbol{k}}}^{t} + \eta \mathbf{x}_n$$

where η is the learning rate. A good choice for it could be $\frac{1}{\mathcal{V}[\hat{k}]}$, i.e., the inverse of the new total number of points in the cluster $\hat{k}$. The SGD based K-means algorithm can be summarised as:

Algorithm 2: SGD based Online K-Means Clustering

Function `SGD_K_Means`$(\mathbf{x}_n, K, [\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \ldots, \boldsymbol{\mu_K}])$**:**
- $\hat{k} = -1$
- $d =$ INFINITY
- `// Find the cluster nearest to the input point`
- **for** $k = 1$ *to* K **do**
  - **if** $||\boldsymbol{\mu_k} - \mathbf{x}_n|| \leq d$ **then**
    - $d = ||\boldsymbol{\mu_k} - \mathbf{x}_n||$
    - $\hat{k} = k$
- **end**
- $\mathcal{V}[\hat{k}] = \mathcal{V}[\hat{k}] + 1$ `// Increase the count of points in the cluster`
- $\eta = 1/\mathcal{V}[\hat{k}]$ `// Set learning rate`
- $\boldsymbol{\mu}_{\hat{\boldsymbol{k}}} = (1-\eta)\boldsymbol{\mu}_{\hat{\boldsymbol{k}}} + \eta \mathbf{x}_n$ `// The cluster-mean update equation`

return

Function `Initialize`$(\mathbf{X}, K)$**:**
- $\mathcal{V} :=$ zero K x 1 vector `// Set the number of inputs in each cluster.`
- $\boldsymbol{\mu_i}$: Randomly Initialize Cluster means.
- **repeat**
  - **for** *each* $\mathbf{x}_i$ *in* $\mathbf{X}$ **do**
    - `SGD_K_Means` $(\mathbf{x}_n, K, [\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \ldots, \boldsymbol{\mu_K}])$
  - **end**
- **until** *convergence*

return

The update equation makes sense as it gives weightage to both the existing means and the new point. Initially, when the means are set randomly, learning rate η is high ($= 1$) which makes sense as the assigned point is a better choice over the randomly selected mean. As the algorithm progresses, we encounter more points, and the learning rate falls. The update equation takes into account the fact that the present cluster mean represents the points in the cluster encountered so far and hence, assigns it a weight in proportion to it $(1 - \eta)$. The term $\eta \mathbf{x}_n$ takes in the information from the new point added to the cluster and incorporates it into the new cluster mean. All this ensures that contribution from each element is the same (as it would be in true cluster mean). Thus, we can see how both the update equation and the learning rate make sense.

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

Student Name: Utkarsh Gupta
Roll Number: 180836
Date: November 24, 2020

QUESTION 5

The two major steps of K-means are Cluster assignment and Cluster mean update. For Kernel K-means with feature map ϕ we have:

$$z_n = \arg\min_k ||\phi(\mathbf{x}_n) - \phi(\boldsymbol{\mu_k})||^2 \quad \dots \text{Cluster Assignment}$$

$$\phi(\boldsymbol{\mu_k}) = \frac{1}{N_k} \sum_{\mathbf{x}_n | z_n = k} \phi(\mathbf{x}_n) \qquad \dots \text{Cluster Mean Update}$$

We can convert the cluster assignment to inner product form in order to apply the kernel trick:

$$||\phi(\mathbf{x}_n) - \phi(\boldsymbol{\mu_k})||^2 = \phi(\mathbf{x}_n)^T\phi(\mathbf{x}_n) - 2\phi(\mathbf{x}_n)^T\phi(\boldsymbol{\mu_k}) + \phi(\boldsymbol{\mu_k})^T\phi(\boldsymbol{\mu_k})$$
$$\implies ||\phi(\mathbf{x}_n) - \phi(\boldsymbol{\mu_k})||^2 = \mathcal{K}(\mathbf{x}_n, \mathbf{x}_n) - 2\mathcal{K}(\mathbf{x}_n, \boldsymbol{\mu_k}) + \mathcal{K}(\boldsymbol{\mu_k}, \boldsymbol{\mu_k})$$

An important thing to note here is the fact how $\boldsymbol{\mu_k}$ are (can be) calculated:

$$\boldsymbol{\mu_k} = \phi^{-1}(\frac{1}{N_k} \sum_{\mathbf{x}_n | z_n = k} \phi(\mathbf{x}_n))$$

Clearly, if we have an infinite dimensional feature space, we won't be able to store the actual cluster mean but can perform step 1 using the Kernel Trick, as discussed earlier. We can get around the apparent problem by not explicitly calculating the cluster means. We can assign random values to $\boldsymbol{\mu_k}$ to obtain initial values of z_n. We can then continue to re-assign clusters by combining the two steps. Substituting $\phi(\boldsymbol{\mu_k})$ in the kernelised assignment equation we get:

$$z_n = \arg\min_k \left\{ \phi(\mathbf{x}_n)^T\phi(\mathbf{x}_n) - \frac{2}{N_k}\phi(\mathbf{x}_n)^T \sum_{\mathbf{x}_i | z_i = k} \phi(\mathbf{x}_i) + \frac{1}{N_k^2} \sum_{\mathbf{x}_i, \mathbf{x}_j | z_i = z_j = k} \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) \right\}$$

$$\implies z_n = \arg\min_k \left\{ \mathcal{K}(\mathbf{x}_n, \mathbf{x}_n) - \frac{2}{N_k} \sum_{\mathbf{x}_i | z_i = k} \mathcal{K}(\mathbf{x}_n, \mathbf{x}_i) + \frac{1}{N_k^2} \sum_{\mathbf{x}_i, \mathbf{x}_j | z_i = z_j = k} \mathcal{K}(\mathbf{x}_i, \mathbf{x}_i) \right\}$$

The entire process can be summarised in the following algorithm:

Algorithm 3: Kernel K-Means

for $k = 1$ *to* K **do**
  $N_k := 0$
end
$\boldsymbol{\mu_i}$: Randomly Initialize cluster means. `// Initialization`
for $n = 1$ *to* N **do**
  $\hat{k} = \arg\min_k \mathcal{K}(\mathbf{x}_n, \mathbf{x}_n) - 2\mathcal{K}(\mathbf{x}_n, \boldsymbol{\mu_k}) + \mathcal{K}(\boldsymbol{\mu_k}, \boldsymbol{\mu_k})$
  $z_n = \hat{k}$ `// Initial Cluster Assignment`
  $N_{\hat{k}} = N_{\hat{k}} + 1$
end
repeat
  for $n = 1$ *to* N **do**
    $k' = z_n$
    $\hat{k} = \arg\min_k \left\{ \mathcal{K}(\mathbf{x}_n, \mathbf{x}_n) - \frac{2}{N_{k'}} \sum_{\mathbf{x}_i | z_i = k'} \mathcal{K}(\mathbf{x}_n, \mathbf{x}_i) + \frac{1}{N_{k'}^2} \sum_{\mathbf{x}_i, \mathbf{x}_j | z_i = z_j = k'} \mathcal{K}(\mathbf{x}_i, \mathbf{x}_i) \right\}$
    $z_n = \hat{k}$ `// Cluster Re-assignment (Update)`
    $N_{k'} = N_{k'} - 1$
    $N_{\hat{k}} = N_{\hat{k}} + 1$
  end
until *convergence*

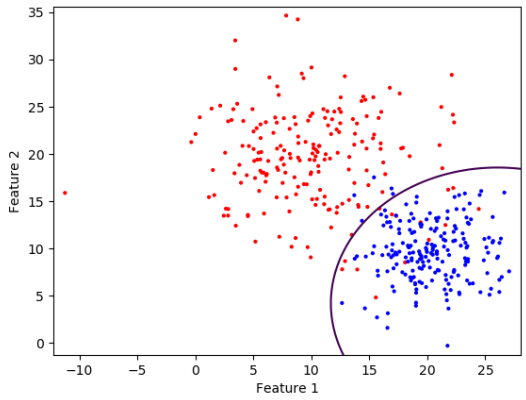
An interesting thing to note here is that similar to classic K-means, we have the Initialization and Updation phase but unlike classic K-means, we do not have the explicit Mean Computation step. The means are never explicitly calculated and the updates are made using the Kernel trick and number of cluster points directly.

In case of classic K-means, we have to calculate K distances for each of the N inputs which can be done in $O(ND)$ time for all inputs (per cluster) and $O(D)$ per input. For Kernel K-means, the Kernel function can be calculated in $O(D)$ time. So, from the update equation, time take to update using each input is $O(D) + O(N_k D) + O(N_k^2 D)$. So, for N inputs, Kernel K-means has a $O(N^3 D)$ cost for all inputs and $O(N^2 D)$ per input.

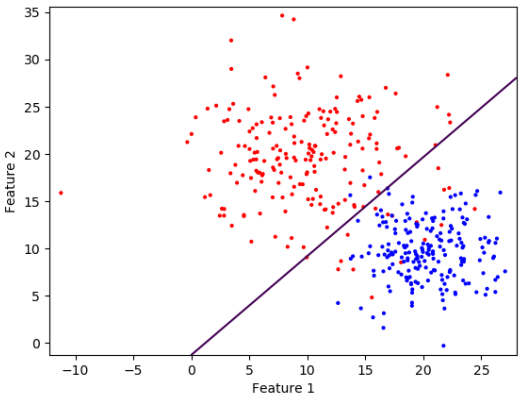
Introduction to ML (CS771), Autumn 2020
Indian Institute of Technology Kanpur
Homework Assignment Number 2

Student Name: Utkarsh Gupta
Roll Number: 180836
Date: November 24, 2020

QUESTION 6

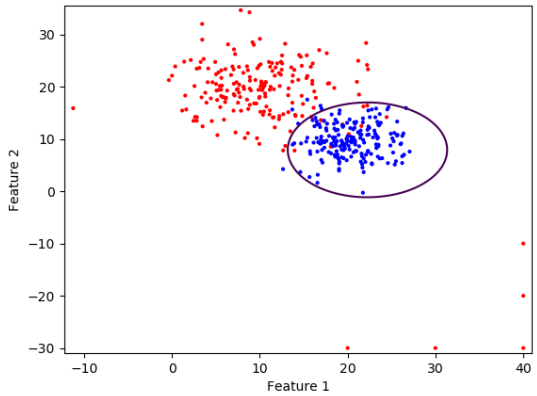


Task 1: Different Covariances

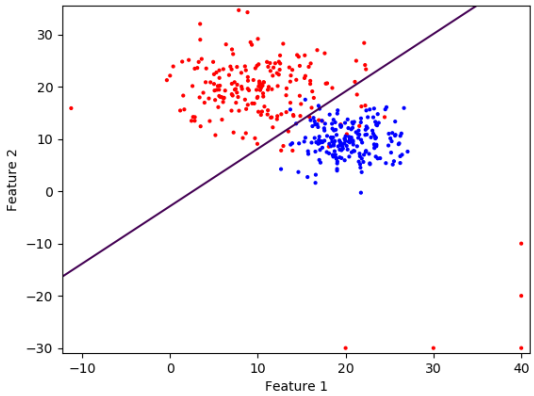


Task 2: Same Covariance

Figure 1: The plots for `binclass.txt`



Task 1: Different Covariances



Task 2: Same Covariance

Figure 2: The plots for `binclassv2.txt`

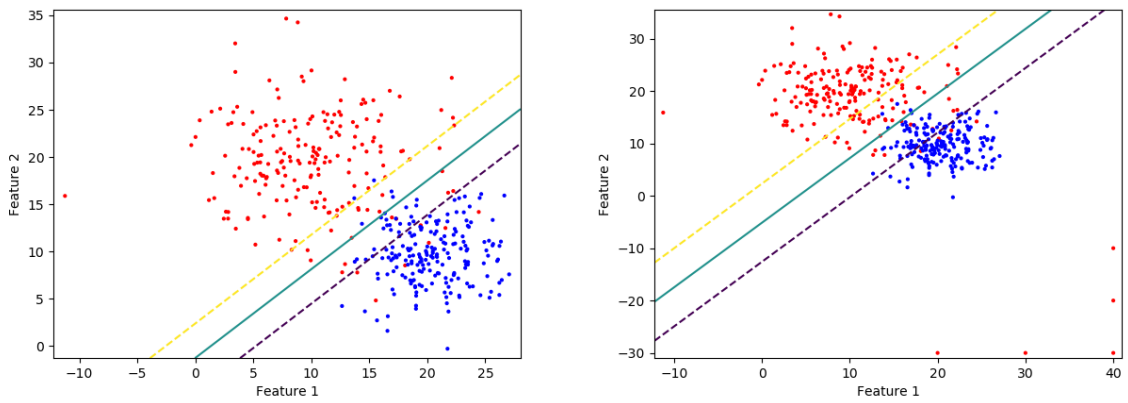
The formulae used:

$$\boldsymbol{\mu_+} = \frac{\sum_{n=1 \mid y_n=1}^{N} \mathbf{x}_n}{N_+} \quad \boldsymbol{\mu_-} = \frac{\sum_{n=1 \mid y_n=-1}^{N} \mathbf{x}_n}{N_-}$$

$$\sigma_+^2 = \frac{\sum_{n=1 \mid y_n=1}^{N} ||\mathbf{x}_n - \boldsymbol{\mu_+}||^2}{DN_+}$$

$$\sigma_-^2 = \frac{\sum_{n=1 \mid y_n=-1}^{N} ||\mathbf{x}_n - \boldsymbol{\mu_-}||^2}{DN_-}$$

$$\sigma^2 = \frac{\sum_{n=1 \mid y_n=1}^{N} ||\mathbf{x}_n - \boldsymbol{\mu_+}||^2 + \sum_{n=1 \mid y_n=-1}^{N} ||\mathbf{x}_n - \boldsymbol{\mu_-}||^2}{DN}$$



Linear Kernel SVM for `binclass.txt` Linear Kernel SVM for `binclassv2.txt`

Figure 3: SVM Plots generated using *scikit-learn* library.

For both the datasets `binclass.txt` as well as `binclassv2.txt`, having separate covariance matrices for the 2 classes does a better job of classifying the points as the data is not linearly separable. In our case, the data has classes that are elliptical in shape. In general cases too, we expect having different covariance for each of the classes will do better. Having same covariance matrix for 2 classes means that the model produces a Linear Decision Boundary between them. Same is the case of Linear SVM which produce a linear decision boundary as well. Linear SVM perform better is the data is linearly separable.