Introduction to ML (CS771), Autumn 2020
Indian Institute of Technology Kanpur
Homework Assignment Number 1

Student Name: Machine Learner
Roll Number: 17001
Date: November 27, 2020

QUESTION 1

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

For logistic regression,

$$P(y_n = 1|\mathbf{x}_n, \mathbf{w}) = \mu_n = \frac{\exp(\mathbf{w}^T \mathbf{x}_n)}{1+\exp(\mathbf{w}^T \mathbf{x}_n)}$$

Using

$$\frac{\delta \mu_n}{\delta \mathbf{w}} = \mu_n(1-\mu_n)\mathbf{x}_n$$

, the gradient and the Hessian of this loss function are given by

$$\begin{aligned}
\mathbf{g} &= \frac{\delta L(\mathbf{w})}{(\delta \mathbf{w})} \\
&= -\sum_{n=1}^{N}\left(y_n \mathbf{x}_n - \frac{\exp(\mathbf{w}^T \mathbf{x}_n)\mathbf{x}_n}{1+\exp(\mathbf{w}^T \mathbf{x}_n)}\right) \\
&= -\sum_{n=1}^{N}\left(y_n - \frac{\exp(\mathbf{w}^T \mathbf{x}_n)}{1+\exp(\mathbf{w}^T \mathbf{x}_n)}\right)\mathbf{x}_n \\
&= \mathbf{X}^T(\mu - \mathbf{y}) \\
\mathbf{H} &= \frac{\delta \mathbf{g}^T}{\delta \mathbf{w}} \\
&= \sum_{n=1}^{N}\mu_n(1-\mu_n)\mathbf{x}_n \mathbf{x}_n^T \\
&= \mathbf{X}^T \mathbf{D} \mathbf{X}
\end{aligned}$$

with D being a diagonal matrix with its n^{th} entry diagonal entry $= \mu_n(1-\mu_n)$.

Therefore, the update equation is given by,

$$\begin{aligned}
\mathbf{w}^{(t+1)} &= \mathbf{w}^{(t)} - \mathbf{H}^{(t)^{-1}}\,\mathbf{g}^{(t)} \\
&= \mathbf{w}^{(t)} - (\mathbf{X}^T \mathbf{D}^{(t)} \mathbf{X})^{-1}\,\mathbf{X}^T(\mu^{(t)} - \mathbf{y}) \\
&= (\mathbf{X}^T \mathbf{D}^{(t)} \mathbf{X})^{-1}\,(\mathbf{X}^T \mathbf{D}^{(t)} \mathbf{X})\mathbf{w}^{(t)} + (\mathbf{X}^T \mathbf{D}^{(t)} \mathbf{X})^{-1}\,\mathbf{X}^T(\mathbf{y} - \mu^{(t)}) \\
&= (\mathbf{X}^T \mathbf{D}^{(t)} \mathbf{X})^{-1}\,[(\mathbf{X}^T \mathbf{D}^{(t)} \mathbf{X})\mathbf{w}^{(t)} + \mathbf{X}^T(\mathbf{y} - \mu^{(t)})] \\
&= (\mathbf{X}^T \mathbf{D}^{(t)} \mathbf{X})^{-1}\,\mathbf{X}^T\,[\mathbf{D}^{(t)} \mathbf{X}\mathbf{w}^{(t)} + \mathbf{y} - \mu^{(t)}] \\
&= (\mathbf{X}^T \mathbf{D}^{(t)} \mathbf{X})^{-1}\,\mathbf{X}^T \mathbf{D}^{(t)}\,[\mathbf{X}\mathbf{w}^{(t)} + \mathbf{D}^{(t)^{-1}}(\mathbf{y} - \mu^{(t)})] \\
&= (\mathbf{X}^T \mathbf{D}^{(t)} \mathbf{X})^{-1}\,\mathbf{X}^T \mathbf{D}^{(t)}\,\hat{\mathbf{y}^{(t)}}\,.
\end{aligned}$$

This solution is of the form of the coefficients of an iteratively reweighted least square problem, i.e.

$$\mathbf{w}^{(t+1)} = \arg\min_{\mathbf{w}} \sum_{n=1}^{N} \mathbf{D}_n^{(t)} (\hat{y_n^{(t)}} - \mathbf{w}^T \mathbf{x}_n)^2$$

where $\mathbf{D}_n^{(t)} = \gamma_n^{(t)} = \mu_n^{(t)}(1 - \mu_n^{(t)})$ is the n^{th} diagonal element of $\mathbf{D}^{(t)}$, and $\hat{y_n} = \mathbf{w}^T \mathbf{x}_n + (y_n - \mu_n)/(\mu_n(1 - \mu_n))$ is the modified real valued label.

Value of $\gamma_n = \mu_n(1 - \mu_n)$, indicating that if the input has an extreme value of μ_n (i.e. lies away from the separator, indicating that the model is doing a good job for that kind of example) then the weight γ_n tends to 0. Therefore, this point is not important for the improvement of the model as it is already doing a good job on classifying it. If the input lies on or close to the separator, i.e. has value of μ_n close to 0.5, then the corresponding weight γ_n will attain a high value. As this point is close to the separator, the model is not confident of its performance for this example and gives it a higher weight to learn the information from this example. Note that the highest value of γ_n is attained at $\mu_n = 0.5$, i.e. if the data point lies on the separator, indicating that the model needs to get better at classifying this and similar inputs.

Introduction to ML (CS771), Autumn 2020
Indian Institute of Technology Kanpur
Homework Assignment Number 1

Student Name: Machine Learner
Roll Number: 17001
Date: November 27, 2020

QUESTION 2

We are given a kernel k with feature map ϕ to make the perceptron learn non-linear decision boundaries. Ignoring the bias term, the hyperplane in the codomain space of the feature map becomes $\mathbf{w}^T\phi(\mathbf{x}) = 0$, where the dimensions of the weights now match the dimension of the codomain space of the feature map. Using the calculations for a standard perceptron, we get the subgradients for the update as

$$\mathbf{g}_n = \begin{cases} 0 & \text{for } y_n\mathbf{w}^T\phi(\mathbf{x}_n) \geq 0 \\ -y_n\phi(\mathbf{x}_n) & \text{for } y_n\mathbf{w}^T\phi(\mathbf{x}_n) < 0 \end{cases}$$

Therefore, SGD will update the weights only when the model makes mistakes. As the expression for the weights has feature map in it, we can't directly store the weights. Instead, we store how many times the algorithm makes mistakes (a_i for ith training example) and use them for prediction. The weights at all times of learning will be $w = \sum_{i=1}^{N} a_i y_i \phi(\mathbf{x}_i)$, similar to what we obtain for the standard perceptron with the $\mathbf{x}$ now being transformed to $\phi(\mathbf{x})$. Note that even for the kernel perceptron, we are updating weights only when the model makes a mistake.

The kernel perceptron algorithm can then be given by

1. Set $a_n = 0$ for all the n training examples (initialization similar to $\mathbf{w} = 0$ in standard perceptron), $t = 0$.
2. Pick some $(\mathbf{x}_n, y_n))$ randomly.
3. calculate

$$\begin{aligned} y &= sign(\mathbf{w}^{(t)^T}\phi(\mathbf{x}_n)) \\ &= sign((\sum_{i=1}^{N} y_i a_i \phi(x_i))^T\phi(\mathbf{x}_n)) \\ &= sign(\sum_{i=1}^{N} y_i a_i (\phi(\mathbf{x}_i))^T\phi(\mathbf{x}_n)) \\ &= sign(\sum_{i=1}^{N} y_i a_i k(\mathbf{x}_i, \mathbf{x}_n)) \end{aligned}$$

4. If $y \neq y_i$ (mistake condition), update a_i (and hence the weights)

$$a_i += 1$$

5. If not converged, go to step 2.

This gives the final weights in terms of a_i and prediction can be done using steps 3 and 4 in the above algorithm. Note that we don't need to explicitly use ϕ in any of the steps (inclusing prediction) if we're given the kernel k.

Introduction to ML (CS771), Autumn 2020
Indian Institute of Technology Kanpur
Homework Assignment Number 1

Student Name: Machine Learner
Roll Number: 17001
Date: November 27, 2020

QUESTION 3

For the class importance SVM, the Lagrangian problem becomes

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

Taking partial derivative w.r.t $\mathbf{w}, b, \xi_n$ and setting to 0 gives

$$\frac{\delta\mathbb{L}}{\delta\mathbf{w}} = 0 \rightarrow \mathbf{w} = \sum_{n=1}^{N}\alpha_n y_n \mathbf{x}_n$$
$$\frac{\delta\mathbb{L}}{\delta b} = 0 \rightarrow \sum_{n=1}^{N}\alpha_n y_n = 0$$
$$\frac{\delta\mathbb{L}}{\delta\xi_n} = 0 \rightarrow C_{y_n} - \alpha_n - \beta_n = 0$$

Using $C_{y_n} = \alpha_n + \beta_n$, $\beta_n \geq 0$ and $\alpha_n \geq 0$, the value of α_n is limited to the range $[0, C_{y_n}]$. Substituing these in the Lagrangian gives the dual as

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

Ignoring the bias term b and noting that β doesn't appear in the dual problem, we get

$$max_{\alpha_n\leq C_{y_n}\forall n}\mathbb{L}_D(\alpha) = \alpha^T\mathbf{1} - \frac{1}{2}\alpha^T\mathbf{G}\alpha$$

C variables penalize slack (ξ) for training examples in soft margin SVM. Therefore, having a larger value of C corresponding to a class (i.e. having a higher penalty for misclassifying that class) would deter the model to misclassfy that class at the cost of increasing the margin. Hence, this leads to unequal penalization for different classes and can factor in unequal class importance.

If we look at the problem from the viewpoint of α, the contribution of the C variable is in bounding the domain of α over which the dual is maximized. If the C_{+1} and C_{-1} differ and without loss of generality, say $C_{+1} > C_{-1}$, then the value of α corresponding to the training examples labeled as $y_n = +1$ have a bigger range ($= [0, C_{+1}]$) over which it needs to maximize the dual as compared to examples with $y_n = -1$ having a smaller range ($= [0, C_{-1}]$). The values of α tells us how important that training example is. Therefore, the model is giving more freedom to α corresponding to C_{+1} to explore, giving the model an more options to choose α for examples with $y_n = +1$. As higher freedom for α corresponds to a larger Lagrangian in the dual problem, the class with a larger range for α (i.e. higher value of C) would be penalized more on breaking constraints. Therefore, the loss on the model for a misclassification corresponding

to $y_n = +1$ is more, in response to which the algorithm will learn parameters which are more sensitized towards minimizing loss for examples with $y_n = +1$. Hence, setting different values of C corresponding to different classes provides a way to bias the model towards minimizing the loss corresponding to a class more than what's done for the other classes.

Student Name: Machine Learner
Roll Number: 17001
Date: November 27, 2020

QUESTION 4

Taking an online approach to K-means by randomly picking examples at a time (say x_n) would give a loss function

$$\mathbb{L} = \sum_{k=1}^{K} z_{nk} ||x_n - \mu_k||^2$$

Assigning x_n greedily to be the best cluster involves picking the μ_k closest to x_n and assigning kth cluster to it, as that μ_k minimizes $\mathbb{L}$. Therefore, we just need to loop over μ and pick the value closest to x_n as the cluster.

The gradient update for $\mathbb{L}$ for the chosen μ_k (Hence assigning $z_{nk} = 1$) with step size η is given by

$$\begin{aligned}
\frac{\delta \mathbb{L}}{\delta \mu_k} &= -2 z_{nk} (x_n - \mu_k) \\
\mu_k &= \mu_k - \eta \frac{\delta \mathbb{L}}{\delta \mu_k} \\
&= \mu_k + 2\eta (x_n - \mu_k) \\
&= (1 - 2\eta)\mu_k + 2\eta x_n
\end{aligned}$$

Intuitively, the update makes sense as it is moving μ_k along the line joining μ_k and x_n towards x_n (as the update is along $(x_n - \mu_k)$). Therefore, it seems like the value of μ_k is being updated online (if the value x_n is assigned the kth cluster) to take into account the effect of x_n.

A good step size would be $\eta = 1/(2n_k)$ where n_k is the number of times cluster k was picked during the online updates. With this value of stepsize, the update becomes

$$\mu_k = \frac{n_k - 1}{n_k} \mu_k + \frac{x_n}{n_k} .$$

This is a good choice becomes the update essentially becomes a running average of the x_ns assigned cluster k, which is equivalent to taking the average of all the X_n assigned k upto this point along with the initial value. Also, as the stepsize is adaptive, it becomes more robust to the new incoming x_n as the algorithm progresses by weighing them by $1/n_k$. As this weight is decaying, the variability in the estimate of μ_k reduces and the estimate become stable.

QUESTION 5

Since we can't store feature-map induced representation or cluster means in the kernel-induced feature space, we will rely on using the entire training set in each update of learning, foregoing the benefit of using the cluster means as summary statistics. Using the notation from class notes, the cluster mean in kernel-induced space is

$$\phi(\mu_k) = \frac{1}{|C_k|} \sum_{n:z_n=k} \phi(\mathbf{x}_n)$$

The Eucledian distance for the ϕ induced quantities can be written using only the kernel as

$$\begin{aligned}
||\phi(x_n) - \phi(\mu_k)||^2 &= 2||\phi(x_n)||^2 + ||\phi(\mu_k)||^2 - 2\phi(x_n)^T\phi(\mu_k) \\
&= k(x_n, x_n) + k(\mu_k, \mu_k) - 2k(x_n, \mu_k)\,. \\
k(\mu_k, \mu_k) &= \phi(\mu_k)^T\phi(\mu_k) \\
&= \left(\frac{1}{|C_k|} \sum_{n:z_n=k} \phi(x_n)\right)^T \left(\frac{1}{|C_k|} \sum_{n:z_n=k} \phi(x_n)\right) \\
&= \frac{1}{|C_k|^2} \sum_{m,n:z_n=z_m=k} \phi(x_m)^T\phi(x_n) \\
&= \frac{1}{|C_k|^2} \sum_{m,n:z_n=z_m=k} k(x_n, x_m)\,. \\
k(x_n, \mu_k) &= \phi(x_n)^T \left(\frac{1}{|C_k|} \sum_{i:z_i=k} \phi(x_i)\right) \\
&= \frac{1}{|C_k|} \sum_{i:z_i=k} k(x_n, x_i)\,.
\end{aligned}$$

Note that this computation did not use $\phi(\mu_k)$ explicitly.

The kernel K-means algorithm then becomes

1. Initialize z_n (cluster labels for training examples)
2. For n = 1,..,N , assign the closest cluster to x_n

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

   This step doesn't explicitly need $\phi(\mu_k)$ as noted in the above calculations.
3. Go to step 2 if not converged

We don't store the cluster means in kernel K-means. Instead, we loop over $n : z_n = k$ to perform computations with the help of the kernel. For prediction, we can follow step 2 of the algorithm

using the calculations presented above the steps of the algorithm.

For standard K-means, we need to compute the distance of each point x_n from each cluster mean μ_k, with each distance computation being of order D. Therefore, the total computation cost is $O(NKD)$. Recomputing the mean for each cluster is $O(N)$. Therefore, the total computation cost is $O(NKD) + O(NK) = O(NKD)$.

For kernel K-means, the distance computation requires computing the kernel N^2 times while calculating $k(\mu_k, \mu_k)$ (as it has a double sum over points previously assigned the cluster k). Assuming each kernel computation has cost of the order of D, each distance computation cost becomes $O(N^2D)$ as compared to just $O(D)$ in standard K-means. This computation needs to be performed only once in step 2 of the algorithm since it is independent of n. Therefore, it can be precomputed at the beginning of step 2 to serve in the loop for each k. The computation cost of $k(x_n, \mu_k)$ is also $O(N^2KD)$ since it requires going over all points previously assigned the cluster k, for each n and k. These terms dominate the remaining term which is $O(ND)$. The overall computation cost becomes $O(K \times N^2D) + O(N \times K \times ND) = O(N^2KD)$.