CSCI 5525 - Homework 2

Saurabh Mylavaram (mylav008@umn.edu id#:5593072)

October 10, 2020

1 Problem 1: Valid Kernel Functions

1.1 Part a

Given K_1,...K_m are valid kernel functions. It means their gram-matrices are all symmetric and positive semi-definite.

$$K_i(x_1, x_2) = K_i(x_2, x_1) \text{ and } u^T G_i u \geq 0 \quad \forall u \; \forall i \in \{1, 2, ...m\} \tag{1}$$

Here G_i is the gram matrix for the kernel K_i.
To prove K is a valid kernel, we need to prove that its gram-matrix G is also a symmetric and positive semi-definite matrix.

$$\begin{aligned} G_{i,j} = K(x_i, x_j) &= \sum_{p=1}^{m} w_p K_p(x_i, x_j) \\ &= \sum_{p=1}^{m} w_p K_p(x_j, x_i) \quad \text{(from eq (1))} \\ &= K(x_j, x_i) = G_{j,i} \quad \therefore \text{G is symmetric} \end{aligned}$$

Consider $u^T G u$ for some arbitrary vector u

$$\begin{aligned} u^T G u &= u^T (w_1 G_1 + ... + w_1 G_m) u \\ &= w_1 (u^T G_1 u) + ... + w_m (u^T G_m u) \\ &\geq 0 \quad (\because w_j \geq 0 \text{ and } u^T G_j u \geq 0 \; \forall j) \quad \text{(from eq 1)} \\ &\therefore G \text{ is positive semi-definite} \end{aligned}$$

We proved that gram matrix of K is symmetric and p.s.d. Hence K is a valid kernel function.

1.2 Part b

From the result of Part a, we know that if K_1,...K_m are valid kernel functions, then $K(x, x') = \sum_{j=1}^{m} w_j K_j(x, x')$ is also a valid kernel function.
By substituting $m = 2$ and $w_1 = w_2 = 1$ in the above result we get

$$K(x, x') = K_1(x, x') + K_2(x, x') \quad \text{is a valid kernel, if } K_1,\ K_2 \text{ are valid kernels}$$

Q.E.D

2 Problem 2: Linear SVM

The main idea behind Support Vector Machine classifier is to maximize the margin between data-points and the decision boundary. The decision boundary can be seen as a hyper-plane $w^T x + b = 0$.

In binary classification, all points for which the value of $w^T x + b > 0$ are classified as belonging to the class and if $w^T x + b < 0$, then the data-point does not belong to the class.

$$y_i^{\text{pred}} = \text{sign}(w^T x_i + b) \tag{2}$$

The SVM optimization problem (to maximize margin) can be expressed in its dual form in this optimization problem:

$$\max_{\lambda_i \geq 0} \sum_i \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j x_i^T x_j, \quad \text{subject to the conditions } 0 \leq \lambda_i \leq C \; \forall i \text{ and } \sum_i \lambda_i y_i = 0 \tag{3}$$

This is a quadratic problem which we solve with the '*cvxopt*' library. In this formulation, we try to find the optimal λ^* which maximizes the above expression. We then separate out the points for which $\lambda_i^* > 0$, these points are called 'support-vectors' (SV). Once we have λ^* and the support vectors, we can find the weight vector w^* using:

$$w^* = \sum_{i \in \text{SV}} y_i \lambda_i^* x_i$$

Using this w^* value, the predictions can be made for new data using the equation (2).

Training and Cross-validation

First we import the dataset and randomly split it into training set (80%) and testing set (20%). All the training and cross-validation is done on training set and only the final results are reported on testing set.

For the linear SVM binary classification problem, we only have one hyper-parameter C. We try to find the best possible value for C, by running 10-fold cross-validation with each value of C and selecting the one which gives lowest validation error. Results of the cross-validation are shown below.

C value:	1e-4	1e-3	1e-2	0.1	1	10	100	**1000**
Training error (%):	48.9 + 1.6	47.9 + 0.8	48 + 1.7	47.6 + 0.6	47.6 + 0.5	47.6 + 0.6	47.6 + 0.6	47.5 + 0.6
Validation error (%):	51.2 + 5.1	50.2 + 4.5	50.3 + 4.7	48.9 + 4.2	48.8 + 4.2	48.8 + 4.3	48.8 + 4.3	**48.7 + 4.2**

Table 1: Results for Linear SVM

The lowest validation error occured with C value of 1000. SO we use this value to train a model on the entire training dataset, and then use the resulting model to make predictions for testing data. **This gave us a testing error of: 51**.9%

Problem 3: Kernel SVM

As we can see, Linear SVM does not give a very good accuracy. This could be true because the data we're given is probably not linearly separable. When the data is not linearly separable Linear SVM fails.

Kernel Trick

To apply the same SVM method based on maximum margin, we need to transform the data ($\Phi(x)$) into a higher dimensional space where it becomes linearly separable. $x \to \Phi(x)$

But for the purposes of calculations in SVM, we do not need to compute this transformation explicitly. As we can see from equation (3), it contains the term $x_i^T x_j$. So, it is enough for us to know what the inner product in this transformed space is ($\Phi(x_i)^T \Phi(x_j)$).

The optimization problem for kernel SVM case becomes,

$$\max_{\lambda_i \geq 0} \sum_i \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \Phi(x_i)^T \Phi(x_j), \quad \text{subject to the conditions } 0 \leq \lambda_i \leq C \, \forall i \text{ and } \sum_i \lambda_i y_i = 0 \tag{4}$$

In linear case we assume this inner product to be the standard vector dot product. **So SVM with linear kernel is the same as problem 2 where we do not use a kernel. We do not reproduce those results here again.**
In case of RBF kernel, we assume the inner product to be:

$$\Phi(x_i)^T \Phi(x_j) = \exp\left(-\frac{||x_i - x_j||_2^2}{2\sigma^2}\right)$$

The method for solving the quadratic problem remains mostly the same. But there is difference in the way we make predictions. Since we do not explicitly know $\Phi(x)$, we cannot calculate w vector as we did before. Instead we use the following prediction function where we use kernel function to calculate the inner product:

$$y^{\text{pred}} = sign\left(\sum_{i \in \text{SV}} y_i \lambda_i \Phi(x_i)^T \Phi(x)\right)$$

We use the same cross-validation approach to select hyper-parameters as we did in problem 2. The only difference is that we now have two parameters C and σ. We run cross-validation with all possible combinations of them, and select the pair with lowest validation error.

C value:	100	100	100	1000	**1000**	1000	1200	1200
Sigma value:	1	10	50	1	**10**	50	10	50
Training error (%):	0 + 0	1.5 + 0.2	41.5 + 0.6	0.0 + 0.0	0.9 + 0.2	8.2 + 0.9	0.8 + 0.2	7.8 + 0.7
Validation error (%):	6.5 + 1.8	2.1 + 1.2	42.1 + 2.9	6.5 + 1.8	**1.9 + 1.0**	9.3 + 2.5	1.9 + 1.0	8.2 + 1.9

Table 2: Results for SVM with RBF Kernel

The results of all these combinations are too long to be shown here. Some combinations along with their results are shown below in a table. The full CLI outputs for all problems are attached in a separate file called '**CLI-outputs.txt**'.

We can see that the validation error rate goes down as C value is increased however, we do not choose a very high C value as we still want our model to generalize well and not choose a boundary with very small margin. Sigma controls how far away from the decision boundary the data-points are still able to exert influence on its shape. Although the $C = 1000$, $\sigma = 1$ gives 0 training error, it validation error is high which means there might be some farther away points which cause the boundary to flex further leading to poor generalization. So we choose a higher value of sigma at $\sigma = 10$.

Moreover the best validation error was seen with a $C = 1000$ and $\sigma = 10$. For a model with these values, the **testing error is 1.65**%

3 Problem 4: Multi-class Classification

In the previous sections we only dealt with binary classification where we decide if a datapoint belongs to the class or not. In case of multiple possible output classes, we choose the strategy of '**One-vs-All**'.

In One-vs-All strategy, we train 10 separate classifiers for each of the 10 classes. Each of these classifiers is trained by considering data-points belonging to all other classes as with label '-1'. So we re-label the training dataset each time before training classifier in the same way as previous problem.

For making predictions in this method, we do not consider the sign of the function output. Instead we assign the data-point to the class whose classifier returns the highest functions value.

$$y^{\text{pred}} = \underset{c}{\text{argmax}} \left(\sum_{i \,\in\, \text{SV}^c} y_i \lambda_i^c \Phi(x_i)^T \Phi(x) \right)$$

Results for Linear kernel

C value:	1e-4	1e-3	1e-2	0.1	1	10	100	1000
Training error (%):	0 + 0	0 + 0	0 + 0	0 + 0	0 + 0	0 + 0	0 + 0	0 + 0
Validation error (%):	3 + 0.8	2.8 + 0.8	2.8 + 0.7	2.9 + 0.8	2.7 + 0.7	2.8 + 0.8	2.8 + 0.8	2.7 + 0.7

Table 3: Multi-class SVM (Linear Kernel)

In this case, we see that changing C value doesn't affect accuracy a lot. This occurs when the decision boundary is already nearly optimal. Choosing a high C value results in a classifier with very small margin and a very low C value results in a very large margin. We choose a value of $C = 1$, so that it is good trade-off between large margin and not mis-classify many training points.

A Linear SVM model trained on the entire training set with $C = 1$ gave a **testing error of 1.25**%

Results for RBF kernel

C value:	1e-4	0.01	100	0.01	1	**100**	1000	1
Sigma value:	1e-4	100	100	1000	1000	**1000**	1000	1
Training error (%):	0 + 0	0 + 0	0 + 0	10.9 + 0.7	1.6 + 0.2	**0 + 0**	0 + 0	0 + 0
Validation error (%):	89.8 + 0	63.8 + 1.4	63.5 + 1.5	13.0 + 1.5	4.4 + 0.1	**3.3 + 0.6**	3.3 + 0.6	89.8 + 0

Table 4: Multi-class SVM (RBF Kernel)

We can see that the σ values affect accuracy by a lot. Sigma value decides how much influence is exerted by the far-away points from the decision boundary on the decision boundary. Lower sigma means that data points which are very far away from the decision boundary can still affect the boundary, making it more flat.

In our case where we are training the classifier for a single class, we know that there will be many data points which do not belong to the class. We do not want them to exert much influence on the shape of the decision boundary. Instead we want the decision boundary to depend more on the points which belong to our class and which are near the decision boundary. Hence we choose a high σ.

The C, σ combination with the best validation error rate is $C = 100$ $\sigma = 1000$. An RBF model trained with these parameters over the entire training set gave a **testing error of $\mathbf{3.25\%}$**