Homework 5 *

Ishan Upadhyaya

February 22, 2016

1 Bias Variance Tradeoff

1.1 Part A

The closed form solution for the given optimization problem can be written as

$$\hat{\beta_\lambda} = (X^T X + n\lambda I_p)^{-1} X^T \boldsymbol{y} \text{ ; Where } \boldsymbol{y} \text{ is the output vector}$$

Using the given linear model, $y_i = x_i^T \beta^\star + \epsilon$, we can write the above equation as

$$\hat{\beta_\lambda} = \underbrace{(X^T X + n\lambda I_p)^{-1} X^T}_{A_\lambda} (X\beta^\star + \mathcal{E}) = A_\lambda X \beta^\star + A_\lambda \mathcal{E}$$

Here, $\mathcal{E}$ is a vector given by $\mathcal{E} = \begin{bmatrix} \epsilon_1 & \epsilon_2 & \cdots & \epsilon_n \end{bmatrix}^T$ and ϵ_i is the noise for ith data point

Using properties of affine transformation of Gaussian random vectors, we can say that $\hat{\beta_\lambda}$ is a Gaussian random vector with mean and variance given below

$$\mu_{\hat{\beta_\lambda}} = A_\lambda X \beta^\star \text{ and } C_{\hat{\beta_\lambda}} = A_\lambda^T C_\mathcal{E} A_\lambda$$
Here $C_\mathcal{E}$ is the convariance matrix of $\mathcal{E}$

1.2 Part B

Using linearity property of Expectation, we get that $\mathbb{E}[x^T \hat{\beta_\lambda}] = x^T \mathbb{E}[\hat{\beta_\lambda}]$. Using the above expression for given probelm we get

$$\begin{aligned} \mathbb{E}[x^T \hat{\beta_\lambda}] - x^T \beta^\star &= x^T \{\mathbb{E}[\hat{\beta_\lambda}] - \beta^\star\} = x^T \{A_\lambda X \beta^\star - \beta^{\beta^\star}\} \\ &= x^T \underbrace{\{A_\lambda X - I_p\}}_{Q} \beta^\star = b_\lambda \end{aligned}$$

$$\begin{aligned} Q &= A_\lambda X - I_p = (X^T X + n\lambda I_p)^{-1} X^T X - I_p \\ &= (I_p - n\lambda (X^T X)^{-1})^{-1} - I_p = I_p - n\lambda (X^T X + n\lambda I_p)^{-1} - I_p \\ &= -n\lambda (X^T X + n\lambda I_p)^{-1} \end{aligned}$$

*COMP SCI 260 - Machine Learning Algorithms ; Fall 2015

1.3 Part C

The expression $\mathbb{E}[\{x^T\hat{\beta_\lambda} - \mathbb{E}[x^T\hat{\beta_\lambda}]\}^2]$ can be written as $\underbrace{\mathbb{E}[\{x^T\{\hat{\beta_\lambda} - \mathbb{E}[\hat{\beta_\lambda}]\}\}^2]}_{P}$. Simplyfying P further we get

$$\begin{aligned} P &= \mathbb{E}[\{x^T(A_\lambda X\beta^\star + A_\lambda\mathcal{E} - A_\lambda X\beta^\star)\}^2] \\ &= \mathbb{E}[\{x^T A_\lambda\mathcal{E}\}^2] = \mathbb{E}[x^T A_\lambda\mathcal{E}\mathcal{E}^T A_\lambda^T x] = x^T A_\lambda\mathbb{E}[\mathcal{E}\mathcal{E}^T]A_\lambda^T x = x^T A_\lambda C_\mathcal{E} A_\lambda^T x \end{aligned}$$

1.4 Part D

Consider behavior of P and A_λ as $\lambda \to 0$ and $\lambda \to \infty$.

$$\lim_{\lambda\to 0} P = 0 \text{ and } \lim_{\lambda\to\infty} P = -nI_p$$
$$\lim_{\lambda\to 0} A_\lambda = X^{-1} \text{ and } \lim_{\lambda\to\infty} A_\lambda = \frac{1}{n\lambda}X^T = \mathbf{0}_{pxn}$$

Therefore we get

$$\lim_{\lambda\to 0} b_\lambda = 0 \text{ and } \lim_{\lambda\to\infty} b_\lambda = -x^T nI_p\beta^\star = -nx^T\beta^\star$$
$$\lim_{\lambda\to 0} P = x^T X^{-1} C_\mathcal{E}(X^{-1})^T x \text{ and } \lim_{\lambda\to\infty} P = 0$$

In words, the bias becomes zeros as $\lambda = 0$ while the variance term goes to zero as $\lambda \to \infty$

2 Kernilized Perceptron

2.1 Part A

We show that $\boldsymbol{w}$ is a linear weighted sum of feature vectors using an argument similar to that of Induction. We know that $\boldsymbol{w}$ is intialized as 0, thus the first time that $\boldsymbol{w}$ is updated, we have $\boldsymbol{w} = y_k\varphi(x_k)$, where 'k' is the first sample for which $\boldsymbol{w}$ misclassifies. Consequently, we can write this $\boldsymbol{w}$ as $\sum_{i=1}^n \alpha_i\varphi(x_i)$, where $\alpha_i = 0\ \forall i \neq k$ and $\alpha_k = y_k$. Now, suppose the immediately subsequent update to $\boldsymbol{w}$ occurs for j^{th} sample point, then we write $\boldsymbol{w} = \sum_{i=1}^n \alpha_i\varphi(x_i) + y_j\varphi(x_j) = \sum_{i=1}^n \hat{\alpha_i}\varphi(x_i)$ such that $\hat{\alpha_i} = \alpha_i \forall i \neq j$ and $\hat{\alpha_i} = \alpha_i + y_i$ for $i = k$. Thus we have shown that $\boldsymbol{w}$ is a linear sum of weighted feature vectors after two updates. Thus we claim, that it is a weighted sum after 'm' updates. Now consider the $(m+1)^{th}$ update at say p^{th} data point. Using, the same logic as above, we can write $\boldsymbol{w} = \sum_{i=1}^n \alpha_i\varphi(x_i) + y_p\varphi(x_p) = \sum_{i=1}^n \hat{\alpha_i}\varphi(x_i)$ such that $\hat{\alpha_i} = \alpha_i \forall i \neq p$ and $\hat{\alpha_i} = \alpha_i + y_i$ for $i = p$. Hence, we have proved that if the weight vector is initialized as a zero vector, the we can write it as a linear weighted sum of feature vectors, i.e. $\boldsymbol{w} = \sum_{i=1}^n \alpha_i\varphi(x_i)$.

2.2 Part B

The prediction of perceptron is given by $\hat{y_k} = \text{sign}(\boldsymbol{w}^T\varphi(x_k))$. Expanding this equation further we get,

$$\begin{aligned}\hat{y_k} &= sign(\boldsymbol{w}^T\varphi(x_k)) \\ &= \text{sign}\{(\sum_{i=1}^{n}\alpha_i\varphi(x_i))^T\varphi(x_k)\} = \text{sign}\{\sum_{i=1}^{n}\alpha_i \underbrace{\varphi(x_i)^T\varphi(x_k)}_{k(x_i,x_k)}\} \\ &= \text{sign}\sum_{i=1}^{n}\alpha_i k(x_i, x_k)\end{aligned}$$

Hence, the prediction rule for perceptron can be written in terms of a kernel function.

2.3 Part C

From part A we know that $\boldsymbol{w} = \sum_{i=1}^{n}\alpha_i\varphi(x_i)$. During the training phase, for any update we have $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_k\varphi(x_k) = \sum_{i=1}^{n}\alpha_i\varphi(x_i) + y_k\varphi(x_k)$. This equation can be modified as $\alpha_k \leftarrow \alpha_k + y_k$ to bring about the same change in the weight vector. Also notice that the prediction of a new data point $\varphi(x_k)$ is given by sign $\sum_{i=1}^{n}\alpha_i k(x_i, x_k)$, which is independent of the explicit form of weight vecotr $\boldsymbol{w}$. Thus, during the training time, we only need access to α_i's and x_i's to update the weight vector.

During testing time, we are just predicting the label of a data point x_k using the sign $\sum_{i=1}^{n}\alpha_i k(x_i, x_k)$, which just requires access to x_i's and α_i's to make prediction. Thus proving that we don't need access to explicit form of $\boldsymbol{w}$ during testing time as well.

The algorithm for kernilized matrix is given as follows.

1. Initialize $\alpha_i = 0\ \forall i$.
2. Set counter k=0
3. For current data point 'k', predict label $\hat{y_k} = \text{sign}\sum_{i=1}^{n}\alpha_i k(x_i, x_k)$
4. If $\hat{y_k} \neq y_k$ update α_k as $\alpha_k \leftarrow \alpha_k + y_k$
5. k $\leftarrow$ k+1
6. Iterate till convergence of $\boldsymbol{w}$

3 Kernels

Let $\boldsymbol{v} \in \mathbb{R}^n$

3.1 Part A

For matrix $K_3 = a_1K_1 + a_2K_2$, the term $\boldsymbol{v}^TK_3\boldsymbol{v}$ is given by

$$\boldsymbol{v}^TK_3\boldsymbol{v} = a_1\boldsymbol{v}^TK_1\boldsymbol{v} + a_2\boldsymbol{v}^TK_2\boldsymbol{v} \geq 0$$

$\because a_1, a_2 \geq 0$ and $\boldsymbol{v}^TK_1\boldsymbol{v}, \boldsymbol{v}^TK_2\boldsymbol{v} \geq 0$ As k_1 and k_2 are positive semi definite

$\therefore K_3$ is positive semidefinite

3.2 Part B

The matrix generated by the function $\mathrm{k}_4(x_1, x_2)$ is shown below

$$K_4 = \begin{bmatrix} f(x_1)f(x_1) & f(x_1)f(x_2) & \cdots & f(x_1)f(x_n) \\ f(x_2)f(x_1) & f(x_2)f(x_2) & \cdots & f(x_2)f(x_n) \\ \vdots & \vdots & \ddots & \vdots \\ f(x_n)f(x_1) & f(x_n)f(x_2) & \cdots & f(x_n)f(x_n) \end{bmatrix} = \begin{bmatrix} f(x_1) & (x_2) & \cdots & f(x_n) \end{bmatrix} \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix}$$
$$= \boldsymbol{f}\boldsymbol{f}^T$$

Therefore, $\boldsymbol{v}^T k_4 \boldsymbol{v}$ is given by

$$\boldsymbol{v}^T k_4 \boldsymbol{v} = \boldsymbol{v}^T \boldsymbol{f}\boldsymbol{f}^T \boldsymbol{v} = \{\boldsymbol{f}^T \boldsymbol{v}\}^2 \geq 0$$

Therefore, K_4 is positive semi definite.

3.3 Part C

Given that $\mathrm{k}_5 = \mathrm{k}_1(x_1, x_2)\mathrm{k}_2(x_1, x_2)$, where k_1 and k_2 are kernel functions, we can alternatively express k_1 and k_2 as

$$k_1(x_1, x_2) = \varphi_1(x_1)^T \varphi_1(x_2) = \sum_{i=1}^{n} \varphi_1(x_1)_i * \varphi_1(x_2)_i$$
$$k_2(x_1, x_2) = \varphi_2(x_1)^T \varphi_2(x_2) = \sum_{i=1}^{m} \varphi_2(x_1)_i * \varphi_2(x_2)_i$$

Using the above equations we can write k_5 as

$$\begin{aligned} k_5(x_1, x_2) &= (\sum_{i=1}^{n} \varphi_1(x_1)_i * \varphi_1(x_2)_i)(\sum_{j=1}^{m} \varphi_2(x_1)_j * \varphi_2(x_2)_j) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{m} = (\varphi_1(x_1)_i \varphi_2(x_1)_j) * (\varphi_1(x_2)_i \varphi_2(x_2)_j) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{m} \varphi_3(x_1)_{i,j} * \varphi_3(x_2)_{i,j} \\ &= \varphi_3(x_1)^T \varphi_3(x_2) \text{ ; here } \varphi_3(.) \text{ is a (nm)x1 dimensional vector} \end{aligned}$$

Thus we can write $k_5(\cdot, \cdot)$ as an innerproduct of two vectors i.e. $k_5(x_1, x_2) = \varphi_3(x_1)^T \varphi_3(x_2)$. Now consider $\boldsymbol{v}^T K_5 \boldsymbol{v}$

$$\begin{aligned} \boldsymbol{v}^T K_5 \boldsymbol{v} &= \sum_{i,j} \boldsymbol{v}_i < \varphi_3(x_i), \varphi_3(x_j) > \boldsymbol{v}_j \\ &=< \sum_i \boldsymbol{v}_i \varphi_3(x_i), \sum_j \varphi_3(x_j)\boldsymbol{v}_j >= || \sum_j \varphi_3(x_j)\boldsymbol{v}_j ||_2^2 \\ &>> 0 \end{aligned}$$

Hence, K_5 is positive semidefinite.

4 Soft Margin Hyperplanes

4.1 Part A

Lagrangian of the modified optimization problem is given by

$$\mathcal{L}_p = C\sum_n \xi_n + \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} - \sum_n \lambda_n\xi_n + \sum_n \alpha_n[1 - y_n(\boldsymbol{w}^T\phi(x_n) + b) - \xi_n]$$
$$\frac{\partial \mathcal{L}_p}{\partial \xi_n} = pC\xi_n^{p-1} - \lambda_n - \alpha_n = 0$$
$$\frac{\partial \mathcal{L}_p}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_n y_n\alpha_n\phi(x_n) = 0$$
$$\frac{\partial \mathcal{L}_p}{\partial b} = \sum_n \alpha_n y_n = 0$$

Substituting the conditions back into lagrangian we get

$$\mathcal{G}_p = \sum_n \{C\xi_n^{p-1} - \lambda_n - \alpha_n\}\xi_n + \frac{1}{2}||\sum_n y_n\alpha_n\phi(x_n)||_2^2 + \sum_n \alpha_n - \sum_n y_n\alpha_n b - \sum_n y_n\alpha_n(\sum_m y_m\alpha_m\phi(x_m))^T\phi(x_n)$$

$$\mathcal{G}_p = \sum_n (1-p)C\xi_n^p + \sum_n \alpha_n - \frac{1}{2}\sum_{n,m} y_m y_n\alpha_n\alpha_m\phi(x_n)^T\phi(x_m)$$
$$= \underbrace{\frac{1-p}{C^{\frac{1}{p-1}}}\sum_n (\frac{\lambda_n + \alpha_n}{p})^{\frac{p}{p-1}}}_{S_p} + \sum_n \alpha_n - \frac{1}{2}\sum_{n,m} y_m y_n\alpha_n\alpha_m\phi(x_n)^T\phi(x_m)$$

Hence, the dual formation of the problem is given as

$$\boldsymbol{\alpha} = \max_\alpha \mathcal{G}_p$$
$$\alpha_n, \lambda_n \geq 0$$
$$\sum_n y_n\alpha_n = 0$$

4.2 Part B

The general formulation is more complex than the one discussed in class because of the additional S_p term. Also, the number of variables involved is double than that of the simpler formulation, as we have the additional λ_n terms.

5 Programming

5.1 Data Preprocessing

Mean and standard deviation before pre-processing

1. 3^{rd} feature: Mean = 2.5260, Standard Deviation = 1.0856

2. 10^{th} feature: Mean = 2.5526, Standard Deviation = 1.1222

Mean and standard deviation after pre-processing

1. 3^{rd} feature: Mean = -0.013, Standard Deviation = 1.0099
2. 10^{th} feature: Mean = 0.0228, Standard Deviation = 0.9979

5.2 Cross validation for Linear SVM

5.2.1 A

Variation of Cross validation accuracy and training time are shown below. After a certain value of C, accuracy flattens out as can be observed in the graph, while the training time generally increases with value of C.

5.2.2 B

As it can be observed from the graphs, maximum cross-validation accuracy is obtained at C = 4^{-3} with an accuracy of 80.9%

5.2.3 C

Test accuracy if the model is trained using C = 4^{-3} is 84.64%.

5.3 Linear SVM using LibSVM

Plots for variation of cross-validation accuracy and training time with C are shown below

As can be observed, the accuracy plot is similar to the one we obtained using quadprog, while the time plot follows a similar trend of increasing training time with value of C. In general, training time of LibSVM is slightly more than the one observed from our implementation.

The maximum cross-validation accuracy acheived was 80.4% with value of C as 4^{-5}. This value is slightly different from the one obtained using our implementaion. This might due to differing optimization algorithm used by quadprog and LibSVM or because of different forms(primal and dual) being optimized by them or a combination of the above two reasons.

5.4 SVM using Kernels

Cross validation and training times for Polynomial kernel function are shown below for different degrees. Please note that the X-axis is in log scale.

Cross validation and training times for RBF kernel function are shown below for different values of gamma. Please note that the X-axis is in log scale.

As can be observed, the maximum cross-validation accuracy is obtained with RBF kernel, with C = 4 and gamma = 4^{-3}. The corresponding accuracy = 88%.

The test accuracy for these parameters is 90.4368%.