CS 260 : Machine Learning Algorithm
Homework #5

Ronak Sumbaly
UID - 604591897

November 12, 2015

Bias-Variance Tradeoff

Canonical Setup
Consider the L_2 regularized linear regression as follows,

$$\beta_\lambda = arg\ min_\beta \left\{ \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 + \lambda ||\beta||_2^2) \right\}$$

Part a. Closed Form Solution.
In order to find the closed form solution of $\hat{\beta_\lambda}$ we differentiate the above equation w.r.t β and equate to 0.

$$\begin{aligned}
\frac{\partial \hat{\beta_\lambda}}{\partial \beta} &= \frac{2}{n} \beta\ X^T \cdot X - 2 \cdot X^T \cdot Y + 2\lambda \cdot \beta \\
&= 0 \\
\hat{\beta} &= (X^T \cdot X + \lambda)^{-1} X^T \cdot Y \\
&= (X^T \cdot X + \lambda)^{-1} X^T \cdot (X\beta^* + \epsilon)
\end{aligned}$$

We also know that the distribution for the noise is, $\epsilon \sim N(0, \sigma^2)$.
Applying properties of affine transformation we can write the distribution of y to be as follows,

$$Y \sim N(X\beta^*, \sigma^2 \mathbf{I})$$

Hence, the distribution of the closed form solution of $\hat{\beta_\lambda}$ can be written as,

$$\hat{\beta_\lambda} \sim N\Big(\big(X^T X + \lambda\big)^{-1} X^T X \beta^*, \big(X^T X + \lambda\big)^{-1} X^T X (XX^T + \lambda I)^{-1} \Big)$$

Part b. Bias Term Solution.
The bias solution can be calculated by removing the x^T term out from the $\mathbb{E}$ term since its a constant value and then replacing the $\mathbb{E}[\hat{\beta_\lambda}]$ with its mean value calculate in Part a.

$$\begin{aligned}
\mathbb{E}[\mathbf{x}^T \hat{\beta_\lambda}] - x^T \beta^* &= \mathbf{x}^T (\mathbb{E}[\hat{\beta_\lambda}] - \beta^*) \\
&= \mathbf{x}^T \Big(\big(X^T X + \lambda\big)^{-1} X^T X \beta^* - \beta^* \Big) \\
&= \mathbf{x}^T \Big(\big(X^T X + \lambda\big)^{-1} X^T X - \mathbf{I} \Big) \beta^*
\end{aligned}$$

Part c. Variance Term Solution.

While calculating the variance term we use the affine transformation theorem where we know that $[x^T(\hat{\beta_\lambda}) - \mathbb{E}[\hat{\beta_\lambda}]]$ has $mean = 0$ and variance equal to the value calculate in Part a.

$$\begin{aligned}\mathbb{E}\left[\left(x^T(\hat{\beta_\lambda} - \mathbb{E}[\hat{\beta_\lambda}])\right)^2\right] &= \mathbf{x}^T\left(X^TX + \lambda\right)^{-1}X^TX(XX^T + \lambda I)^{-1}\mathbf{x} \\ &= ||X(XX^T + \lambda)^{-1}x||_2^2\end{aligned}$$

Part d. From Part b. and Part c. of the bias and variance tradeoff we can see that if λ increases, the **bias** term also **increases** while the **variance** term **decreases**.

Kernelized Perceptron

Part a. Prove.

$\boldsymbol{w}$ is always a linear combination of feature vectors, $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i \phi(x_i)$

Proof.
Applying proof by induction we formulate the proceed as follows,
Base Case: When $k = 0$, we have, for weight vector w_0,

$$w_0 = 0 = \sum_{i=1}^{N} \alpha_i \phi(x_i)$$

where $\alpha_i = 0 \; \forall i$
Induction Hypothesis: For a value n where $k = n > 0$ we have following to be true,

$$w_n = \sum_{i=1}^{N} \alpha_n \phi(x_n)$$

Proving for (n+1): For $k = n + 1$ we have the weight vector w_{n+1} to be

$$\begin{aligned}w_{n+1} &= w_n + y_n\phi(x_n) \\ &= \sum_{i=1}^{N} \alpha_i \phi(x_i) + y_n\phi(x_n) \\ &= \sum_{i=1}^{N} (\alpha_i + y_i[i == n])\phi(x_i) \\ &= \sum_{i=1}^{N} \alpha_i' \phi(x_i)\end{aligned}$$

Hence by proof of induction $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i \phi(x_i)$

Part b. Prove.

The update rule for $\boldsymbol{w}$ for a kernelized Perceptron does depend on the explicit feature mapping $\phi(x)$

Proof.
We know that from Part a. that $\boldsymbol{w}$ can always be represented as a linear combination of feature vectors, the update rule for $\boldsymbol{w}$ can therefore be updated as follows,

$$\begin{aligned} y &= sign(\boldsymbol{w}^T \phi(x_j)) \\ &= sign((\sum_{i=1}^{N} \alpha_i \ \phi(x_i)^T).\phi(x_j)) \\ &= sign((\sum_{i=1}^{N} \alpha_i < \phi(x_i)^T.\phi(x_j) >) \end{aligned}$$

Therefore $< \phi(x_i)^T.\phi(x_j) >$ represents the inner products between nonlinear transformed features and hence the update rule is independent of $\boldsymbol{w}$.

Part c. Canonical Setup.

We do not need to explicitly store $\boldsymbol{w}$ at training or test time. Instead, we can implicitly use it by maintaining all the α_i

Solution.
From Part a. as proven by induction we can write the new weight vector as follows,

$$w_{i+1} = \sum_{i=1}^{N} (\alpha_i \ \phi(x_i)) + y_n \ \phi(x))$$

For each iteration the value of α would be updated using $\alpha_{i+1} = \alpha_i + y_i$. The revised algorithm without the weight vector can be written as follows,

Algorithm
$\alpha \leftarrow 0$
REPEAT
Pick a data point x_n
Make a prediction $y = sign(\sum_{i=1}^{N} \alpha_i < \phi(x_i)^T.\phi(x_n) >)$
IF $y = y_n$, do nothing ELSE,
$\quad \alpha_n \leftarrow \alpha_n + y_n$
UNTIL Converged.

Kernels

Part a. Canonical Setup.

$$K_3 = a_1 K_1 + a_2 K_2$$

where $a_1, a_2 \geq 0$ and K_1, K_2 are positive semi-definite.

Proof.
To show that K_3 is positive semi-definite we need to show for any $u \in \mathbb{R}^N$:

$$\begin{aligned} \boldsymbol{u}^T.K_3.\boldsymbol{u} &\geq 0 \\ \boldsymbol{u}^T.K_3.\boldsymbol{u} &= \boldsymbol{u}^T.(a_1 K_1 + a_2 K_2).\boldsymbol{u} \\ &= a_1.\boldsymbol{u}^T.K_1.\boldsymbol{u} + a_2.\boldsymbol{u}^T.K_2.\boldsymbol{u} \end{aligned}$$

By the facts given for $u \in \mathbb{R}^N$ both $\boldsymbol{u}^T.K_1.\boldsymbol{u}$ and $\boldsymbol{u}^T.K_2.\boldsymbol{u}$ are ≥ 0 (by definition of positive semi-definite). Thus, the sum of two non-negative terms is also non-negative.
Hence $u^T.K_3.u \geq 0 \Rightarrow K_3$ is positive semi-definite.

Part b. Canonical Setup.
K_4 is defined as

$$k_4(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}) f(\mathbf{x}')$$

where $f(.)$ is an arbitrary real valued function.

Proof.
We can represent the kernel K_4 as,

$$K_4 = \begin{bmatrix} f(x_1)f(x_1) & f(x_1)f(x_2) & ... & f(x_1)f(x_n) \\ ... & ... & ... & ... \\ f(x_n)f(x_1) & f(x_n)f(x_2) & ... & f(x_n)f(x_n) \end{bmatrix}$$

We consider $\boldsymbol{F}$ to be as,

$$\boldsymbol{F} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ . \\ . \\ f(x_n) \end{bmatrix}$$

Thus from above we get $K_4 = \boldsymbol{F}.\boldsymbol{F}^T$. Considering the condition for positive semi-definite we have for any $u \in \mathbb{R}^N$:

$$\begin{aligned} \boldsymbol{u}^T F F^T \boldsymbol{u} &= (F^T \boldsymbol{u})^T F^T \boldsymbol{u} \\ &= ||F^T \boldsymbol{u}||_2^2 \\ ||F^T \boldsymbol{u}||_2^2 &>= 0 \\ \boldsymbol{u}^T F F^T \boldsymbol{u} &>= 0 \end{aligned}$$

Hence K_3 is positive semi-definite.

Part c. Canonical Setup.

K_5 is defined as

$$k_5(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')\ k_2(\mathbf{x}, \mathbf{x}')$$

Proof.

K_5 is the Hadamard product of two positive semi-definite matrices $K_1, K2$, which is always positive semi-definite by Schur product theorem.

Soft Margin Hyperplane

Part a. Dual formulation of Soft Margin Hyperplane

The primal form of the soft margin hyperplane problem,

$$\begin{aligned} \min \quad & \frac{1}{2}\|\omega\|_2^2 + C\sum_i \xi^p \\ s.t. \quad & y_i(\omega x_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, \quad \forall i \end{aligned}$$

Can be represented as follows,

$$L(\omega, b, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) = C\sum_n \xi_n^p + \frac{1}{2}\|\omega\|_2^2 - \sum_n \lambda_n\xi_n + \sum_n \alpha_n\{1 - y_n[\omega^T x_n + b] - \xi_n\}$$

In order to derive the dual formulation we take partial derivatives w.r.t ω, b, ξ_n

$$\begin{aligned} \frac{\partial L}{\partial \omega} &= \omega - \sum_n y_n\alpha_n x_n = 0 \\ \frac{\partial L}{\partial b} &= \sum_n \alpha_n y_n = 0 \\ \frac{\partial L}{\partial \xi_n} &= C \cdot p\sum_n \xi_n^{p-1} - \lambda_n - \alpha_n = 0 \end{aligned}$$

Substituting the above results back in the Lagrangian of the primal form we get,

$$\begin{aligned} g(\{\alpha_n\}, \{\lambda_n\}) &= \sum_n C\xi_n^p - (\alpha_n + \lambda_n)\xi_n + \frac{1}{2}\|\sum_n y_n\alpha_n x_n\|_2^2 + \sum_n \alpha_n + (\sum_n \alpha_n y_n)b \\ &\quad - \sum_n \alpha_n y_n(\sum_m y_m\alpha_m x_m)^T x_n \\ &= \sum_n C\xi_n^p - (\alpha_n + \lambda_n)\xi_n + \sum_n \alpha_n - \frac{1}{2}\alpha_n\alpha_m y_m y_n x_m x_n \end{aligned}$$

Hence the dual formulation can be written as,

$$\begin{aligned} \max \quad & g(\{\alpha_n\}, \{\lambda_n\}) \\ s.t. \quad & \sum_n \alpha_n y_n = 0 \\ & C \cdot p\sum_n \xi_n^{p-1} - \lambda_n\alpha_n = 0 \\ & \alpha_n \geq 0 \\ & \lambda_n \geq 0, \forall n \end{aligned}$$

Part b. The general formulation for (p > 1) makes it more complex than the standard setting of (p = 1) as the slack ξ_n terms for (p > 1) cannot be estimated in the dual form.

Support Vector Machine Programming

Data Preprocessing

The mean and variance should be estimated from the training data and then applied to both the datasets because in formulation of a model we are initially only given a training data on the basis of which this model is created. This scaling process is a part of the model generated by the training data that we are going to use and thus for each new testing dataset we are applying the tests on both the generality of the model combined with the pre-processing.

Mean & Variance of 3^{rd} Feature in Testing Data

$$Mean = -0.0130$$
$$Variance = 1.0099$$

Mean & Variance of 10^{th} Feature in Testing Data

$$Mean = 0.0228$$
$$Variance = 0.9979$$

Cross Validation for Linear SVM

Linear SVM		
Accuracy	Time	Trade-Off
52.50	0.0360	0.0002
78.30	0.0400	0.0010
80.40	0.0440	0.0039
80.40	0.0440	0.0156
79.80	0.0640	0.0625
80.30	0.0700	0.2500
80.10	0.0580	1.0000
80.20	0.0700	4.0000
79.90	0.0740	16.0000

It can be seen that as the C value increases the accuracy increases upto a specific point and after that it decreases due to overfitting of the model.

The best accuracy for the training data was found for $\mathbf{C = 0.0039062}$. The corresponding test accuracy is,

$$Training\ Accuracy = 85.4253$$

Linear SVM in LibSVM

Linear SVM in LibSVM		
Accuracy	Time	Trade-Off
51.7000	0.0920	0.0002
78.8000	0.0720	0.0010
80.5000	0.0580	0.0039
79.7000	0.0480	0.0156
79.6000	0.0460	0.0625
79.4000	0.0640	0.2500
79.5000	0.1600	1.0000
79.3000	0.4020	4.0000
79.2000	1.3700	16.0000

The LibSVM and the linear SVM give the same accuracy for the same trade-off values.

Kernel SVM in LibSVM

Polynomial Kernel

Polynomial Kernel		
C	Accuracy	Time $(1.0e+04)$
4^{-4}	52	0.0000
4^{-3}	52	0.0000
4^{-2}	61	0.0000
4^{-1}	69	0.0000
4^{0}	77	0.0000
4^{1}	76	0.0000
4^{2}	77	0.0000
4^{3}	76	0.0001
4^{4}	76	0.0001
4^{5}	76	0.0003
4^{6}	76	0.0011
4^{7}	76	0.0021

Total Time taken by Polynomial Kernel = 116.31

RBF Kernel

RBF Kernel		
γ	Accuracy	Time$(1.0e+04)$
4^{-4}	63	0.0000
4^{-3}	65	0.0000
4^{-2}	65	0.0000
4^{-1}	66	0.0000
4^{0}	68	0.0000
4^{1}	68	0.0000
4^{2}	71	0.0000
4^{3}	71	0.0000
4^{4}	74	0.0000
4^{5}	71	0.0000
4^{6}	74	0.0000
4^{7}	74	0.0000

Total Time taken by RBF Kernel = 48.34

For more detailed accuracies for each degree and gamma value refer to console output of the code.

The best accuracy among the two Kernel is found in the Polynomial Kernel where $\mathbf{C = 0.25}$ and $\mathbf{degree = 1}$. For the corresponding values the test accuracy calculated is,

$$Training\ Accuracy = 85.5172$$