CSE512 - Machine Learning - Homework 2

Yang Wang, 110014939
Computer Science Department
Stony Brook University
yang.wang@stonybrook.edu

1. Question 1 – Naive Bayes and Logistic Regression

1.1. Naive Bayes with both continuous and boolean variables

The general Bayes classifier is

$$Y = \arg\max_Y P(Y|X) = \arg\max_Y P(X|Y)P(Y)$$

Assuming conditional independence between X_1 and X_2 given Y, the Naive Bayes classifier become

$$Y = \arg\max_Y P(X_1|Y)P(X_2|Y)P(Y)$$

$P(Y)$ has only 1 parameter λ, since Y is boolean.

$$P(Y=0) = \lambda$$
$$P(Y=1) = 1-\lambda$$

$P(X_1|Y)$ has 2 parameters λ_1 and λ_2, since X_1 is also boolean.

$$P(X_1=0|Y=0) = \lambda_1$$
$$P(X_1=1|Y=0) = 1-\lambda_1$$
$$P(X_1=0|Y=1) = \lambda_2$$
$$P(X_1=1|Y=1) = 1-\lambda_2$$

$P(X_2|Y)$ has 4 parameters μ_1, μ_2, σ_1, σ_2, because X_2 is continuous.

$$P(X_2=x_2|Y=0) = \frac{1}{\sqrt{2\pi}\sigma_1} e^{-\frac{(x_2-\mu_1)^2}{2\sigma_1^2}}$$
$$P(X_2=x_2|Y=1) = \frac{1}{\sqrt{2\pi}\sigma_2} e^{-\frac{(x_2-\mu_2)^2}{2\sigma_2^2}}$$

In total, we need 7 parameters to define our Naive Bayes classifier.
Ignoring the normalizer, we can compute $P(Y|X)$ as follows,

$$\begin{aligned} P(Y=0|X) &\propto P(X_1|Y=0)P(X_2|Y=0)P(Y=0) \\ &= \lambda\lambda_1^{1-X_1}(1-\lambda_1)^{X_1}\frac{1}{\sqrt{2\pi}\sigma_1}e^{-\frac{(X_2-\mu_1)^2}{2\sigma_1^2}} \end{aligned}$$

$$\begin{aligned} P(Y=1|X) &\propto P(X_1|Y=1)P(X_2|Y=1)P(Y=1) \\ &= (1-\lambda)\lambda_2^{1-X_1}(1-\lambda_2)^{X_1}\frac{1}{\sqrt{2\pi}\sigma_2}e^{-\frac{(X_2-\mu_2)^2}{2\sigma_2^2}} \end{aligned}$$

1.2. Naive Bayes and Logistic Regression with Boolean variables

Suppose Y is boolean and $\mathbf{X} = (X_1, \cdots, X_d)$ is a vector of boolean variables,

$$\begin{aligned} P(Y=1) &= \lambda \\ P(X_i=1|Y=1) &= \lambda_i \\ P(X_i=1|Y=0) &= \beta_i \end{aligned}$$

Then the Naive Bayes classifier follows the same form as Logistic Regression, proved below.

$$\begin{aligned} & P(Y=1|\mathbf{X}) \\ =& \frac{P(\mathbf{X}|Y=1)P(Y=1)}{P(\mathbf{X}|Y=1)P(Y=1)+P(\mathbf{X}|Y=0)P(Y=0)} \\ =& \frac{P(X_1|Y=1)\cdots P(X_d|Y=1)P(Y=1)}{P(X_1|Y=1)\cdots P(X_d|Y=1)P(Y=1)+P(X_1|Y=0)\cdots P(X_d|Y=0)P(Y=0)} \\ =& \frac{\lambda_1^{X_1}(1-\lambda_1)^{1-X_1}\cdots\lambda_d^{X_d}(1-\lambda_d)^{(1-X_d)}\lambda}{\lambda_1^{X_1}(1-\lambda_1)^{1-X_1}\cdots\lambda_d^{X_d}(1-\lambda_d)^{(1-X_d)}\lambda+\beta_1^{X_1}(1-\beta_1)^{1-X_1}\cdots\beta_d^{X_d}(1-\beta_d)^{(1-X_d)}(1-\lambda)} \\ =& \frac{1}{1+(\frac{\beta_1}{\lambda_1})^{X_1}(\frac{1-\beta_1}{1-\lambda_1})^{1-X_1}\cdots(\frac{\beta_d}{\lambda_d})^{X_d}(\frac{1-\beta_d}{1-\lambda_d})^{1-X_d}\frac{1-\lambda}{\lambda}} \\ =& \frac{1}{1+exp(log((\frac{\beta_1}{\lambda_1})^{X_1}(\frac{1-\beta_1}{1-\lambda_1})^{1-X_1}\cdots(\frac{\beta_d}{\lambda_d})^{X_d}(\frac{1-\beta_d}{1-\lambda_d})^{1-X_d}\frac{1-\lambda}{\lambda}))} \\ =& \frac{1}{1+exp(\sum_{i=1}^{d}(\ X_i log(\frac{\beta_i}{\lambda_i})+(1-X_i)log(\frac{1-\beta_i}{1-\lambda_i})+log\frac{1-\lambda}{\lambda}\ )} \\ =& \frac{1}{1+exp(-(\sum_{i=1}^{d}\theta_i X_i+\theta_{d+1}))} \end{aligned}$$

2. Question 2 – Support Vector Machines

2.1. Linear case

In order to compute the LOOCV error, we need to remove each data point in turn, re-train the SVM using the remaining data and test on the removed point. Consider two cases: (1) When we remove a non-support vector (on the correct side of the margin), the SVM classifier would stay the same and make no error. (2) When we remove a support vector (on the margin or the wrong side of the margin), the SVM could change and possibly introduce a wrong prediction. Thus the number of wrong predictions is no more than m and the LOOCV is bounded above by $\frac{m}{n}$.

2.2. General case

Yes, the bound of LOOCV error still holds for general kernel SVM. Because the previous reasoning does not depend on the linearity of the SVM, and still holds for general kernel SVM.

3. Question 3 – Implementation of SVMs

3.1. Implement Kernel SVM using Quadratic Programming (QP)

3.1.1 write the SVM dual objective as a quadratic program

We can reformulate the dual objective of kernel SVM as follows,

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}[Diag(Y)\alpha]^T K[Diag(Y)\alpha] - \mathbf{1}_n^T\alpha \\ s.t. \quad & Y^T\alpha = 0 \\ & \mathbf{0}_n \le \alpha \le C\mathbf{1}_n \end{aligned}$$

By comparing with the standard form of quadratic programming, we have,

$$
\begin{aligned}
H &= Diag(Y) \cdot K \cdot Diag(Y) \\
f &= -\mathbf{1}_n \\
Aeq &= Y^T \\
beq &= 0 \\
lb &= \mathbf{0}_n \\
ub &= C\mathbf{1}_n
\end{aligned}
$$

3.1.2

I've implemented a matlab function called **KernelSVM** to solve α of the dual SVM.

3.1.3

I've implemented a matlab function called **KernelSVM_wrap** to compute w and b of the primal SVM, only for linear kernel.

3.1.4 $C = 0.1$

```
When C is 0.100:
The classification accuracy is 90.74%.
The objective value is 24.76.
The number of support vectors is 340.00.
The confusion matrix is:
|-------|
|181| 32|
|  2|152|
|-------|
```

3.1.5 $C = 100$

```
When C is 100.000:
The classification accuracy is 97.82%.
The objective value is 112.15.
The number of support vectors is 125.00.
The confusion matrix is:
|-------|
|179|  4|
|  4|180|
|-------|
```

3.2. Implement Linear SVM using Stochastic Gradient Descent (SGD)

3.2.1 SGD update rules for w and b

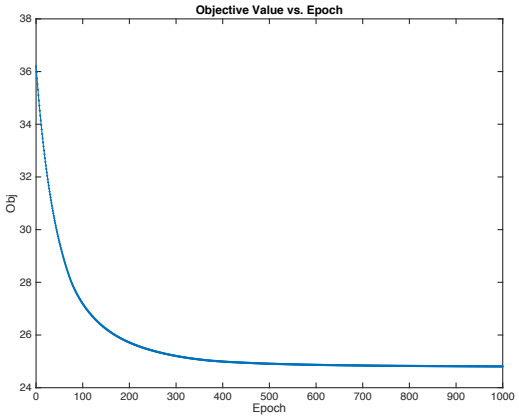
$$\begin{aligned}\mathbf{w}^{new} &= \mathbf{w}^{cur} - \eta\frac{\partial L_j}{\partial \mathbf{w}} \\ &= \mathbf{w}^{cur} - \eta(\frac{1}{n}\mathbf{w} + C\frac{\partial L(\mathbf{w}, b, \mathbf{x}_j, y_j)}{\partial \mathbf{w}}) \\ &= \begin{cases}(1-\frac{\eta}{n})\mathbf{w}^{cur}, & \text{if } y_j(\mathbf{w}^T\mathbf{x}_j + b) \geq 1 \\ (1-\frac{\eta}{n})\mathbf{w}^{cur} + \eta C y_j \mathbf{x}_j, & \text{if } y_j(\mathbf{w}^T\mathbf{x}_j + b) < 1\end{cases}\end{aligned}$$

$$\begin{aligned}b^{new} &= b^{cur} - \eta\frac{\partial L_j}{\partial b} \\ &= \begin{cases}b^{cur}, & \text{if } y_j(\mathbf{w}^T\mathbf{x}_j + b) \geq 1 \\ b^{cur} + \eta C y_j, & \text{if } y_j(\mathbf{w}^T\mathbf{x}_j + b) < 1\end{cases}\end{aligned}$$

3.2.2

I've implemented a matlab function called **LinearSVM_SGD** to solve α of the linear SVM.

3.2.3 $C = 0.1$



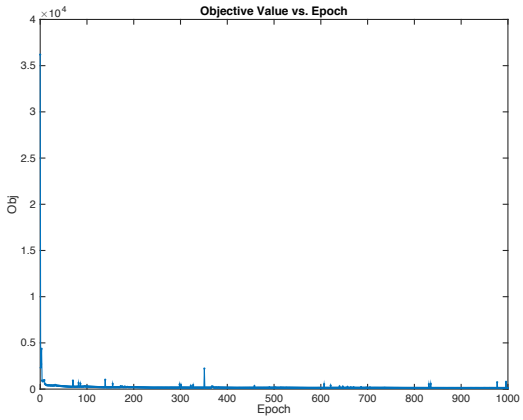
When $C = 0.1$, the objective we obtained using SGD is 24.81, where QP algorithm got 24.76.

3.2.4 $C = 0.1$

```
When C is 0.100:
The training error is 7.18%.
The test error is 10.63%.
The |w| is 4.00.
The objective value is 24.81.
The confusion matrix is:
|-------|
|181| 37|
|  2|147|
|-------|
```

3.2.5 $C = 100$

When $C = 100$, I still use $\eta_0 = 1$, $\eta_1 = 100$, results are reported below.

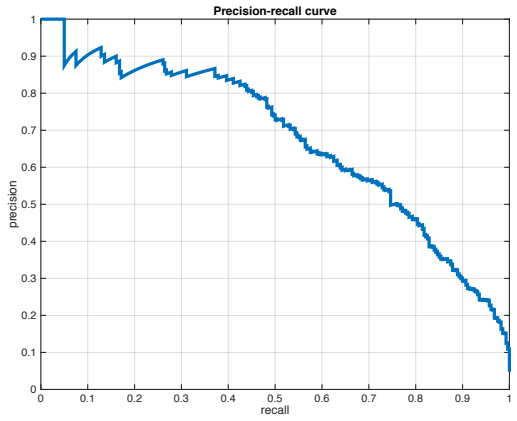



```
When C is 100.000:
The training error is 0.00%.
The test error is 1.63%.
The |w| is 15.64.
The objective value is 122.29.
The confusion matrix is:
|-------|
|178|  1|
|  5|183|
|-------|
```

4. SVM for object detection

4.1. Random Negative Data

Training with random negative data, we obtained $AP = 68.08\%$ on validation data.

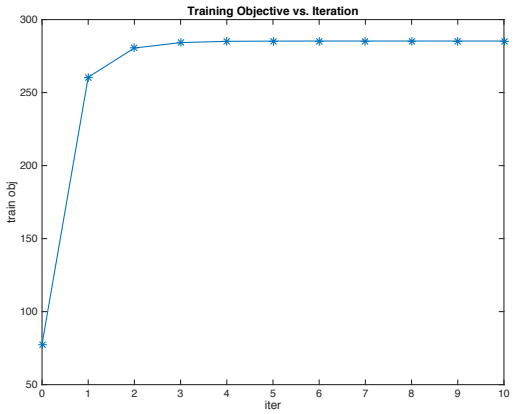


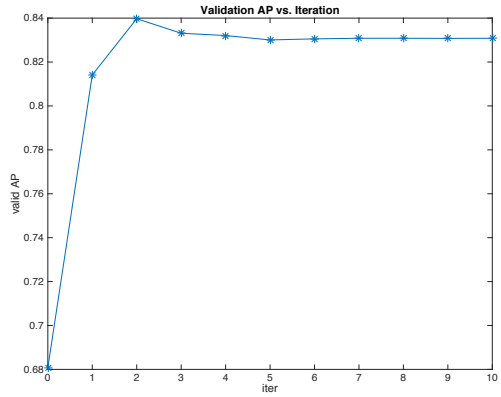
4.2. Hard Negative Mining

I have implemented the Hard Negative Mining algorithm in file **upperbody.m**

4.3. Hard Negative Mining

Training with hard negative mining, we obtained $AP = 83.08\%$ on validation data.





4.4. Competition

For the competition, I get $AP = 76.06\%$ on the test data.