UNIVERSITY OF SOUTHERN CALIFORNIA

Homework 3

Peehoo Dewan
Collaberated with Coco Chengliangdong, Pooja Voladoddi, Satakshi Rana
October 17, 2014

1 KERNELIZED PERCEPTRON

1.1 LINEAR COMBINATION

(a) Given, x has a nonlinear feature mapping s.t

$$x \rightarrow \varphi(x)$$

Using perceptron algorithm, we iterate over all training examples and try to learn weights that predicts by checking $y = sgn(w^T \varphi(x))$

Algorithm :

$$\text{If} \quad y = sgn(w^T \varphi(x)) \quad \text{we do nothing}$$
$$\text{otherwise,} \quad w^{t+1} = w^t + y_i \varphi(x_i) \quad \text{where} \quad i = 1,2 \ldots N \quad t = \text{iteration}$$

We see above that each training example may / may not contribute to the weights. If the training example gets classified correctly then we do not take its contribution. Nonetheless, some training examples can be misclassified multiple times and they would contribute to weight more than once. Let β_i be the number of times $\varphi(x_i)$ is misclassified. Then w can be expressed as,

$$w^{t+1} = w^t + \beta_i y_i \varphi(x_i) \quad \text{where} \quad i = 1,2 \ldots N \quad t = \text{iteration}$$

where,

$$w^t = \sum_{j=1}^{N} \beta_j y_j \varphi(x_j) \qquad t = \text{iteration}$$

Therefore, the weight vector can be expressed as a linear combination of the features.

$$\mathbf{w} = \sum_{i=1}^{N} \beta_i y_i \varphi(x_i)$$

Since $y \in \{-1, 1\}$, above statement can be written as,

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \varphi(x_i)$$

where,

$$\alpha_i = y_i * \beta_i$$

1.2 INNER PRODUCT BETWEEN NONLINEAR TRANSFORMED FEATURE VECTORS

(b) Substituting the value of w obtained in part (a) in the below equation we get,

$$\begin{aligned} y_j &= \text{sign}(w^T \varphi(x_j)) \\ &= \text{sign}((\sum_{i=1}^{N} \alpha_i \varphi(x_i))^T \varphi(x_j)) \\ &= \text{sign}(\sum_{i=1}^{N} \alpha_i \varphi(x_i)^T \varphi(x_j)) \\ &= \text{sign}(\sum_{i=1}^{N} \alpha_i < \varphi(x_i), \varphi(x_j) >) \end{aligned}$$

This proves that the prediction can be expressed as an inner product of the transformed feature vectors

1.3 ALGORITHM FOR MAINTAINING ALPHA

Step 1 : Initialize $\vec{\alpha}$ to a zero vector of size N where N is the number of training examples.
Step 2 : Run step 3 for a fixed number of iterations or until a stopping criteria like $(y_j = \hat{y}_j)$ is met.
Step 3 : For each training example (x_j, y_j)
Step 3.1 : Calculate $\hat{y}_j = \text{sign}(\sum_{i=1}^{N} \alpha_j < \varphi(x_i), \varphi(x_j) >$
Step 3.2 : If $\hat{y}_j \neq y_j$ then we update α_j
Step 3.3 : $\alpha'_j = \alpha_j + 1$
Step 3.4 : $\alpha_j = \alpha'_j$

2 Support Vector Machine without Bias

2.1 Slack Variables

(a)

$$\begin{aligned} f(x) &= \mathbf{w}^T \varphi(\mathbf{x}) \\ y &= sign(\mathbf{w}^T \varphi(\mathbf{x})) \end{aligned} \tag{2.1}$$

$$l^{\text{hinge}} = max(0, 1 - yf(x))$$

Let L = Total hinge loss on all training examples

$$L = \sum_{n=1}^{N} \max(0, 1 - y_n f(x_n))$$

Minimizing the overall hinge loss we get,

$$\begin{aligned} L &= \min_{w} \ \sum_{n} \max(0, 1 - y_n f(x_n)) + \frac{\lambda}{2}||\mathbf{w}||_2^2 \\ &= \min_{w} \ \sum_{n} \max(0, 1 - y_n(\mathbf{w}^T \varphi(\mathbf{x}_n))) + \frac{\lambda}{2}||\mathbf{w}||_2^2 \\ &= \min_{w} \ C\sum_{n} \max(0, 1 - y_n(\mathbf{w}^T \varphi(\mathbf{x}_n))) + \frac{1}{2}||\mathbf{w}||_2^2 \end{aligned}$$

Where,

$$C = \frac{1}{\lambda}$$

Let the slack variable ξ_n be defined as

$$\xi_n = \max(0, 1 - y_n(\mathbf{w}^T \varphi(\mathbf{x}_n)))$$

Therefore we can write,

$$L = \min_{w, \{\xi_n\}} \ C\sum_{n} \xi_n + \frac{1}{2}||\mathbf{w}||_2^2$$

Where,

$$\xi_n \geq 0 \ \text{ and } \ \xi_n \geq 1 - y_n(\mathbf{w}^T \varphi(\mathbf{x}_n)) \ \ \forall n$$

2.2 Lagrangian Primal form

(b) From the above constraints, we can write the lagrangian as,

$$L(w, \{\xi_n\}, \{\alpha_n\}, \{\lambda_n\}) = \min_{w, \{\xi_n\}} \ C\sum_{n=1}^{N} \xi_n + \frac{1}{2}||\mathbf{w}||_2^2 + \sum_{n=1}^{N} \lambda_n(-\xi_n) + \sum_{n=1}^{N} \alpha_n(1 - y_n(\mathbf{w}^T \varphi(\mathbf{x}_n)) - \xi_n) \tag{2.2}$$

such that,

$$\alpha_n \geq 0 \text{ and } \lambda_n \geq 0$$

2.3 Lagrangian Primal form - Link to Dual form

Differentiating w.r.t w we get,

$$\frac{\partial L(w,\{\xi_n\},\{\alpha_n\},\{\lambda_n\})}{\partial w} = \mathbf{w} - \sum_{n=1}^{N} \alpha_n(y_n\varphi(\mathbf{x}_n)) := 0$$

This gives,

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n(y_n\varphi(\mathbf{x}_n)) \quad \forall n \tag{2.3}$$

Differentiating w.r.t ξ_n we get,

$$\frac{\partial L(w,\{\xi_n\},\{\alpha_n\},\{\lambda_n\})}{\partial \xi_n} = C - \lambda_n - \alpha_n := 0$$

This gives,

$$C - \lambda_n - \alpha_n = 0 \quad \forall n \tag{2.4}$$

2.4 Lagrangian Primal variables substituted

Rearranging and Substituting (1.3) and (1.4) in (1.2) we get,

$$\begin{aligned} L(w,\{\xi_n\},\{\alpha_n\},\{\lambda_n\}) &= \min_{w,\{\xi_n\}} \sum_{n=1}^{N}(C-\lambda_n-\alpha_n)\xi_n + \frac{1}{2}\mathbf{w}\mathbf{w}^T + \sum_{n=1}^{N}\alpha_n(1-y_n(\mathbf{w}^T\varphi(\mathbf{x}_n))) \\ &= \min_{w,\{\xi_n\}} \frac{1}{2}\sum_{m=1}^{N}\alpha_m(y_m\varphi(\mathbf{x}_m))(\sum_{n=1}^{N}\alpha_n(y_n\varphi(\mathbf{x}_n)))^T + \sum_{n=1}^{N}\alpha_n(1-y_n((\sum_{m=1}^{N}\alpha_m(y_m\varphi(\mathbf{x}_m)))^T\varphi(\mathbf{x}_n))) \\ &= \min_{w,\{\xi_n\}} \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{m=1}^{N}\sum_{n=1}^{N}\alpha_m\alpha_n(y_m y_n\varphi(\mathbf{x}_m)^T(\varphi(\mathbf{x}_n)) \end{aligned}$$

2.5 Lagrangian Dual form

$$\max_{\alpha} g(\{\alpha_n\},\{\lambda_n\}) = \max_{\alpha} \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{m=1}^{N}\sum_{n=1}^{N}\alpha_m\alpha_n(y_m y_n\varphi(\mathbf{x}_m)^T(\varphi(\mathbf{x}_n)) \tag{2.5}$$

such that,

$$\begin{aligned} 0 \le \alpha_n \le C, \quad \forall n \\ \lambda_n \ge 0 \forall n \end{aligned}$$

2.6 Difference between the dual form with/without bias term

Due to the absence of the bias term, the current dual form lacks the constraint $\sum_{n=1}^{N}\alpha_n y_n = 0$

2.7 Convex optimization

The second term in equation (1.5) is a sum of square terms. By the properties of convex functions, a linear combination of convex functions is also a convex function. The first term in the equation is a constant. Subtracting the second term from the first makes the dual function a concave function and therefore the problem reduces to maximization of concave function.

3 Sample question for quiz

3.1 Regularization

Let

$$N = \text{number of training examples}$$
$$p = \text{number of features}$$

Then we know that x and θ are p dimensional vectors denoted by,

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix} \qquad \vec{\theta} = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_p \end{pmatrix}$$

For training data set D, the likelihood of data L(D) can be written as,

$$L(D) = p(D;\theta)$$

In order to prevent overfitting, we create a modified log likelihood function also called the objective function, which penalizes the weights θ such that,

$$\theta = arg\max_{\theta} \ \ln p(D;\theta) - \frac{\lambda}{2}||\theta||_2^2 \qquad \text{where, } \lambda > 0$$

We maximize this function. If we change out likelihood function into cross entropy function by taking negative of likelihood, then we minimize it such that,

$$\theta = arg\min_{\theta} \ -\ln p(D;\theta) + \frac{\lambda}{2}||\theta||_2^2 \qquad \text{where, } \lambda > 0$$

3.2 Logistic Regression

Let

$$N = \text{number of training examples}$$
$$p = \text{number of features}$$

Then we know that x, w_1 and w_0 are p dimensional vectors denoted by,

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix} \qquad \vec{w}_0 = \begin{pmatrix} w_{01} \\ w_{02} \\ \vdots \\ w_{0p} \end{pmatrix} \qquad \vec{w}_1 = \begin{pmatrix} w_{11} \\ w_{12} \\ \vdots \\ w_{1p} \end{pmatrix}$$

3.2.1 Posterior Probability

(a) The posterior probabilities using softmax are given as,

$$p(y=0|\vec{x}) = \frac{e^{\vec{w_0}^T\vec{x}}}{e^{\vec{w_0}^T\vec{x}} + e^{\vec{w_1}^T\vec{x}}}$$

$$p(y=1|\vec{x}) = \frac{e^{\vec{w_1}^T\vec{x}}}{e^{\vec{w_0}^T\vec{x}} + e^{\vec{w_1}^T\vec{x}}}$$

3.2.2 Adding constant vector

(b) Let $\vec{b}$ be a p dimentional vector which we add to both the weight vectors The posterior probabilities using softmax are then given as,

$$\begin{aligned}
p(y=0|\vec{x}) &= \frac{e^{(\vec{w_0}+\vec{b})^T\vec{x}}}{e^{(\vec{w_0}+\vec{b})^T\vec{x}} + e^{(\vec{w_1}+\vec{b})^T\vec{x}}} \\
&= \frac{e^{(\vec{w_0}^T+\vec{b}^T)\vec{x}}}{e^{(\vec{w_0}^T+\vec{b}^T)\vec{x}} + e^{(\vec{w_1}^T+\vec{b}^T)\vec{x}}} \\
&= \frac{e^{\vec{w_0}^T\vec{x}}e^{\vec{b}^T\vec{x}}}{e^{\vec{w_0}^T\vec{x}}e^{\vec{b}^T\vec{x}} + e^{\vec{w_1}^T\vec{x}}e^{\vec{b}^T\vec{x}}} \\
&= \frac{e^{\vec{w_0}^T\vec{x}}e^{\vec{b}^T\vec{x}}}{(e^{\vec{w_0}^T\vec{x}} + e^{\vec{w_1}^T\vec{x}})e^{\vec{b}^T\vec{x}}}
\end{aligned}$$

Cancelling the common term in numerator and denominator we get the original expression for posterior probability.

$$p(y=0|\vec{x}) = \frac{e^{\vec{w_0}^T\vec{x}}}{e^{\vec{w_0}^T\vec{x}} + e^{\vec{w_1}^T\vec{x}}}$$

Similarly doing the same for $p(y=1|\vec{x})$

$$\begin{aligned}
p(y=1|\vec{x}) &= \frac{e^{(\vec{w_1}+\vec{b})^T\vec{x}}}{e^{(\vec{w_0}+\vec{b})^T\vec{x}} + e^{(\vec{w_1}+\vec{b})^T\vec{x}}} \\
&= \frac{e^{(\vec{w_1}^T+\vec{b}^T)\vec{x}}}{e^{(\vec{w_0}^T+\vec{b}^T)\vec{x}} + e^{(\vec{w_1}^T+\vec{b}^T)\vec{x}}} \\
&= \frac{e^{\vec{w_1}^T\vec{x}}e^{\vec{b}^T\vec{x}}}{e^{\vec{w_0}^T\vec{x}}e^{\vec{b}^T\vec{x}} + e^{\vec{w_1}^T\vec{x}}e^{\vec{b}^T\vec{x}}} \\
&= \frac{e^{\vec{w_1}^T\vec{x}}e^{\vec{b}^T\vec{x}}}{(e^{\vec{w_0}^T\vec{x}} + e^{\vec{w_1}^T\vec{x}})e^{\vec{b}^T\vec{x}}}
\end{aligned}$$

Cancelling the common term in numerator and denominator we get the original expression for posterior probability.

$$p(y=1|\vec{x}) = \frac{e^{\vec{w_1}^T \vec{x}}}{e^{\vec{w_0}^T \vec{x}} + e^{\vec{w_1}^T \vec{x}}}$$

Therefore, this shows that the posterior probability doesn't change even on adding a constant vector to the weights.

3.3 UNIQUE SOLUTION

As seen above, the posterior probability does not change on adding a constant vector. The objective function can be written as,

$$\begin{aligned} \ln P(D) &= \sum_n \ln P(y_n|x_n) \\ &= \sum_{n=1}^{N} \sum_{k=0}^{1} y_{nk} \ln P(C_k|x_n) \end{aligned}$$

where,

$$y_{nk} = \begin{cases} 1 & if \quad y_n = k \\ 0 & otherwise \end{cases}$$

Therefore cross entropy error function which is negative of likelihood would be written as,

$$\begin{aligned} \varepsilon(w_0, w_1) &= -\sum_{n=1}^{N} \sum_{k=0}^{1} y_{nk} \ln P(C_k|x_n) \\ &= -\sum_{n=1}^{N} y_n \ln P(y_n = 1|x_n) + (1 - y_n) \ln P(y_n = 0|x_n) \end{aligned}$$

The above cross entropy function is a convex function as proved in lecture notes. We arrived at this conclusion by applying rules of convex functions and showing that the Hessian is positive semidefinite.
After adding the regularization terms, the new objective function looks like,

$$\varepsilon(w_0, w_1, \lambda_0, \lambda_1) = -\sum_{n=1}^{N} y_n \ln P(y_n = 1|x_n) + (1 - y_n) \ln P(y_n = 0|x_n) + \quad \lambda_0 ||w_0||_2^2 + \quad \lambda_1 ||w_1||_2^2$$

Differentiating the second and third parts of the above equation gives

$$\frac{\partial^2(\lambda_0 ||w_0||_2^2)}{\partial w_0 \partial w_0^T} = 2\lambda_0 \quad \text{and} \quad \frac{\partial^2(\lambda_1 ||w_1||_2^2)}{\partial w_1 \partial w_1^T} = 2\lambda_1$$

As λ_0 and λ_1 are both greater than 0. Therefore their Hessian is positive definite and so they are strictly convex function. On adding these convex function to the cross entropy function, we get that a linear combination of a convex function to a strictly convex function is a strictly convex function. Therefore it would have a unique solution and this is how we can take care of the invariability.

4 PROGRAMMING

4.1 DATA PREPROCESSING

Complete

4.2 IMPLEMENT LINEAR SVM

Complete

4.3 CROSS VALIDATION FOR LINEAR SVM

4.3.1 CROSS-VALIATION ACCURACY AND TIMES

C	Accuracy	Time in seconds
4^{-6}	51.6040%	0.429738
4^{-5}	78.7566%	0.315864
4^{-4}	80.4606%	0.327161
4^{-3}	81.1606%	0.340787
4^{-2}	80.4576%	0.429502
4^{-1}	80.0596%	0.452572
1	80.2606%	0.446431
4^{1}	80.4616%	0.464316
4^{2}	80.4616%	0.508090

We see that when we increase the value of C, the accuracy increases uptil a certain point and then starts decreasing. C is the weight on the slack variables which basically account for the hinge loss. Increasing C to a high value results in λ going towards 0 which increases w whereas decreasing C to a very low value results in high values of λ therefore reducing w. Therefore when we increase C, value of w increases which inturn increases the computational time.

4.3.2 CHOOSING C BASED ON CROSS VALIDATION RESULTS

Looking at the cross validation accuracies, I would choose C= 4^{-3}

4.3.3 REPORTING TEST ACCURACY FOR CHOSEN C

For C= 4^{-3}, test accuracy = 84.64%

4.4 USE LINEAR SVM IN LIBSVM

4.4.1 CROSS-VALIATION ACCURACY AND TIMES

C	Accuracy	Time in seconds
4^{-6}	51.7%	0.0957
4^{-5}	78.8%	0.0902
4^{-4}	80.5%	0.0704
4^{-3}	79.7%	0.0607
4^{-2}	79.6%	0.0592
4^{-1}	79.4%	0.0849
1	79.5%	0.2164
4^{1}	79.3%	0.5712
4^{2}	79.2%	1.9648

The accuracies are closer to what we got in 4.3(a) but they are not exactly the same.

4.4.2 EARLIER IMPLEMENTATION VS LIBSVM

It is a little faster compared to earlier implementation but only for lower values of C. For higher values of C, the earlier implementation is better.

4.5 USE KERNEL SVM IN LIBSVM

4.5.1 POLYNOMIAL KERNEL

C	d	Accuracy	Time in seconds
4^{-4}	1	51.7%	0.0941
4^{-3}	1	51.7%	0.0930
4^{-2}	1	78.7%	0.0845
4^{-1}	1	80.4%	0.0709
1	1	79.5%	0.0583
4^{1}	1	79.2%	0.0603
4^{2}	1	79.5%	0.0910
4^{3}	1	79.4%	0.2173
4^{4}	1	79.3%	0.6499
4^{5}	1	79.2%	2.3403
4^{6}	1	79.3%	9.5658
4^{7}	1	78.8%	18.6672
4^{-4}	2	51.7%	0.1085
4^{-3}	2	51.7%	0.0989
4^{-2}	2	51.7%	0.1082
4^{-1}	2	65.4%	0.0974
1	2	72.8%	0.0951
4^{1}	2	70.9%	0.1119
4^{2}	2	70.9%	0.1093
4^{3}	2	70.9%	0.1103
4^{4}	2	70.9%	0.1115
4^{5}	2	70.9%	0.1235
4^{6}	2	70.9%	0.1165
4^{7}	2	70.9%	0.1137
4^{-4}	3	51.7%	0.1122
4^{-3}	3	51.7%	0.1028
4^{-2}	3	51.7%	0.1092
4^{-1}	3	62.1%	0.0996
1	3	79.2%	0.1028
4^{1}	3	79.0%	0.1076
4^{2}	3	79.2%	0.1105
4^{3}	3	79.2%	0.1085
4^{4}	3	79.2%	0.1087
4^{5}	3	79.2%	0.1083
4^{6}	3	79.2%	0.1080
4^{7}	3	79.2%	0.1088

4.5.2 RBF KERNEL

C	gamma	Accuracy	Time in seconds
4^{-4}	4^{-7}	51.7%	0.0879
4^{-3}	4^{-7}	51.7%	0.0876
4^{-2}	4^{-7}	51.7%	0.0894
4^{-1}	4^{-7}	51.7%	0.0875
1	4^{-7}	51.7%	0.0878
4^{1}	4^{-7}	77.0%	0.0902
4^{2}	4^{-7}	79.5%	0.0732
4^{3}	4^{-7}	79.8%	0.0596
4^{4}	4^{-7}	79.0%	0.0537
4^{5}	4^{-7}	79.4%	0.0632
4^{6}	4^{-7}	81.8%	0.1026
4^{7}	4^{-7}	85.3%	0.2682
4^{-4}	4^{-6}	51.7%	0.0876
4^{-3}	4^{-6}	51.7%	0.0873
4^{-2}	4^{-6}	51.7%	0.0874
4^{-1}	4^{-6}	51.7%	0.0873
1	4^{-6}	76.1%	0.0911
4^{1}	4^{-6}	79.7%	0.0728
4^{2}	4^{-6}	79.7%	0.0594
4^{3}	4^{-6}	79.6%	0.0537
4^{4}	4^{-6}	81.7%	0.0616
4^{5}	4^{-6}	85.4%	0.1082
4^{6}	4^{-6}	87.8%	0.2798
4^{7}	4^{-6}	87.4%	0.3629
4^{-4}	4^{-5}	51.7%	0.0875
4^{-3}	4^{-5}	51.7%	0.0878
4^{-2}	4^{-5}	51.7%	0.0884
4^{-1}	4^{-5}	73.1%	0.0889
1	4^{-5}	79.9%	0.0737
4^{1}	4^{-5}	80.0%	0.0596
4^{2}	4^{-5}	81.1%	0.0533
4^{3}	4^{-5}	85.7%	0.0669
4^{4}	4^{-5}	88.1%	0.1096
4^{5}	4^{-5}	87.4%	0.1456
4^{6}	4^{-5}	87.4%	0.1447
4^{7}	4^{-5}	87.4%	0.1450
4^{-4}	4^{-4}	51.7%	0.0877
4^{-3}	4^{-4}	51.7%	0.0896
4^{-2}	4^{-4}	54.5%	0.0876
4^{-1}	4^{-4}	80.3%	0.0787
1	4^{-4}	81.7%	0.0620
4^{1}	4^{-4}	84.7%	0.0565
4^{2}	4^{-4}	88.0%	0.0659
4^{3}	4^{-4}	87.2%	0.0923
4^{4}	4^{-4}	87.2%	0.0932
4^{5}	4^{-4}	87.2%	0.0925
4^{6}	4^{-4}	87.2%	0.0923
4^{7}	4^{-4}	87.2%	0.0914

C	gamma	Accuracy	Time in seconds
4^{-4}	4^{-3}	51.7%	0.0877
4^{-3}	4^{-3}	51.7%	0.0881
4^{-2}	4^{-3}	61.2%	0.0880
4^{-1}	4^{-3}	83.5%	0.0774
1	4^{-3}	87.2%	0.0688
4^{1}	4^{-3}	88.6%	0.0892
4^{2}	4^{-3}	88.2%	0.0927
4^{3}	4^{-3}	88.2%	0.0893
4^{4}	4^{-3}	88.2%	0.0886
4^{5}	4^{-3}	88.2%	0.0894
4^{6}	4^{-3}	88.2%	0.0892
4^{7}	4^{-3}	88.2%	0.0894
4^{-4}	4^{-2}	51.7%	0.0909
4^{-3}	4^{-2}	51.7%	0.0921
4^{-2}	4^{-2}	51.7%	0.0930
4^{-1}	4^{-2}	51.7%	0.0965
1	4^{-2}	76.6%	0.0978
4^{1}	4^{-2}	77.9%	0.0994
4^{2}	4^{-2}	77.9%	0.0991
4^{3}	4^{-2}	77.9%	0.1001
4^{4}	4^{-2}	77.9%	0.1005
4^{5}	4^{-2}	77.9%	0.0991
4^{6}	4^{-2}	77.9%	0.1003
4^{7}	4^{-2}	77.9%	0.1034
4^{-4}	4^{-1}	51.7%	0.0914
4^{-3}	4^{-1}	51.7%	0.0952
4^{-2}	4^{-1}	51.7%	0.0939
4^{-1}	4^{-1}	51.7%	0.0965
1	4^{-1}	57.3%	0.1028
4^{1}	4^{-1}	57.9%	0.1007
4^{2}	4^{-1}	57.9%	0.1016
4^{3}	4^{-1}	57.9%	0.1002
4^{4}	4^{-1}	57.9%	0.1008
4^{5}	4^{-1}	57.9%	0.1016
4^{6}	4^{-1}	57.9%	0.1013
4^{7}	4^{-1}	57.9%	0.1032

Based on the above observation, I would choose a RBF Kernel with C = 4 and gamma = 4^{-3}
Accuracy = 90.4368 %