CS 5350/6350: Machine Learning Spring 2019

Homework 4

Handed out: 20 Mar, 2019
Due date: 11:59pm, 5 Apr, 2019

Author: Cade Parkison

Note: The file run.sh will run my test.py file which runs all the code needed to generate the results in this report. The svm_testing.ipynb file is a Jupyter Notebook file which can be ran interactively to generate the results in this report. There is also a pdf of the results of this notebook file, titled svm_testing.pdf.

I used the popular python package CVXOPT for the optimization problems in part 2 because I was already familiar with this package and not familiar with scipy. Unfortunately, after finishing the assignment, I realized this package is not installed on the CADE machines. I realize this is a requirement of the assignments, but due to time constraints I was unable to re-implement the code in scipy. My apologies, if you would like, I can come by your office and run this code on my machine to show you the results. Otherwise, a simple pip install cvxopt in the command line will install this package and allow my code to be ran.

1 Paper Problems [40 points]

1. [3 points] The learning of soft SVMs is formulated as the following optimization problem,

$$\begin{aligned} \min_{\mathbf{w},b,\{\xi_i\}} \quad & \frac{1}{2}\mathbf{w}^\top\mathbf{w} + C\sum_i \xi_i, \\ \text{s.t. } \forall 1 \le i \le N, \quad & y_i(\mathbf{w}^\top\mathbf{x}_i + b) \ge 1 - \xi_i, \\ & \xi_i \ge 0 \end{aligned}$$

where N is the number of the training examples. As we discussed in the class, the slack variables $\{\xi_i\}$ are introduced to allow the training examples to break into the margin so that we can learn a linear classifier even when the data is not linearly separable.

   (a) [1 point] What values ξ_i can take when the training example $\mathbf{x}_i$ breaks into the margin?

   The slack variable ξ_i is greater than zero when the training example $\mathbf{x}_i$ is inside the margin.

   (b) [1 point] What values ξ_i can take when the training example $\mathbf{x}_i$ stays on or outside the margin?

   The slack variable ξ_i is equal to zero when the training example $\mathbf{x}_i$ is inside the margin.

(c) [1 point] Why do we incorporate the term $C \cdot \sum_i \xi_i$ in the objective function? What will happen if we throw out this term?

This term allows the trade off between minimizing error and margin size. With this term, we can solve non-separable data at the cost of adding to the objective function for each example that is inside the margin. Without this term, the problem becomes a hard SVM and solutions would not be feasible for non-separable data.

2. [6 points] Write down the dual optimization problem for soft SVMs. Please clearly indicate the constraints, and explain how it is derived. (Note: do NOT directly copy slides content, write down your own understanding.)

In optimization, duality is a way to find bounds on an optimization problem. By augmenting the cost function, it is possible to turn a constrained problem into an unconstrained problem that gives lower bounds on the original constrained optimization problem. In the case of SVM, the dual problem solution is the same as the primal problem solution.

Primal:

$$\begin{aligned} \min_{\mathbf{w},b,\{\xi_i\}} \quad & \frac{1}{2}\mathbf{w}^\top\mathbf{w} + C\sum_i \xi_i \\ \text{s.t. } \forall 1 \le i \le N, \quad & y_i(\mathbf{w}^\top\mathbf{x}_i + b) \ge 1 - \xi_i, \\ & \xi_i \ge 0 \end{aligned}$$

The first step is to remove the constraints from the original primal problem and add them to the cost function with lagrangian multipliers α_i and β_i. We can also switch the order of the min-max.

$$\max_{\alpha_i \ge 0, \beta_i \ge 0} \min_{\mathbf{w},b,\{\xi_i\}} \quad \frac{1}{2}\mathbf{w}^\top\mathbf{w} + C\sum_i \xi_i - \sum_i \beta_i\xi_i - \sum_i \alpha_i(y_i(\mathbf{w}^\top\mathbf{x}_i + b) - 1 + \xi_i)$$

This is now our Lagrangian function, and we can take it's partial derivatives and set to zero to solve the inner minimization problem.

$$\begin{aligned} \frac{\delta L}{\delta \mathbf{w}} &= \mathbf{0} \\ \frac{\delta L}{\delta b} &= 0 \\ \frac{\delta L}{\delta \xi_i} &= 0 \end{aligned}$$

Using these partial derivatives and simplifying, we get the following equations.

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$
$$\sum_i \alpha_i y_i = 0$$
$$\alpha_i + \beta_i = C$$

The next step is to replace $\mathbf{w}$ with the value above, and add the contraints above.

$$\max_{\alpha_i \geq 0, \beta_i \geq 0} \quad -\frac{1}{2}\sum_i \sum_j y_i y_j \alpha_i \alpha_j {\mathbf{x}_i}^T \mathbf{x}_j + \sum_i \alpha_i$$
$$s.t. \sum_i \alpha_i y_i = 0$$
$$\forall i, \alpha_i + \beta_i = C$$

Finally, by constraining $\alpha_i \leq C$ we can remove β_i and add the constraints as the factors to minimize in the final form of the dual problem. Notice we changed the max to a min by negating the entire equation.

$$\min_{0 \geq \alpha_i \geq 0, \sum_i \alpha_i y_i = 0} \quad \frac{1}{2}\sum_i \sum_j y_i y_j \alpha_i \alpha_j {\mathbf{x}_i}^T \mathbf{x}_j - \sum_i \alpha_i$$

3. [8 points] Continue with the dual form. Suppose after the training procedure, you have obtained the optimal parameters.

   (a) [2 points] What parameter values can indicate if an example stays outside the margin?

   If an example stays outside the margin, it will have a zero $alpha_i$ value.

   (b) [3 points] What are common and what are different when we use these parameters and the slack variables $\{\xi_i\}$ to determine the relevant positions of the training examples to the margin?

   For both the α_i and ξ_i parameters, when either are zero, we know the training example is outside the margin. However, the slack variable is not upper bounded, but α_i must be less than the regularization term, C.

   (c) [3 points] Now if we want to find out which training examples just sit on the margin (neither inside nor outside), what shall we do? Note you are not allowed to examine if the functional margin (i.e., $y_i(\mathbf{w}^\top \mathbf{x}_i + b)$) is 1.

   One possible way to find these examples is to find the points where the following equation equals zero.

$$h(x) = \sum_i \alpha_i y_i K(x_i, x) + b = 0$$

4. [3 points] How can we use the kernel trick to enable SVMs to perform nonlinear classification? What is the corresponding optimization problem?

   The kernel trick takes our feature space to a larger dimensional space that can be linearly separable. The following equations define this optimization problem.

$$\min_{\alpha} \frac{1}{2}\alpha^T \mathbf{H} \alpha - \mathbf{1^T}\alpha$$

$$s.t.\ 0 \leq \alpha_i \leq C$$
$$y^T \alpha = 0$$

$$H_{i,j} = y_i y_j K(x_i, x_j)$$

5. [5 points] Prove that the primal objective function of the soft SVM is convex to $\mathbf{w}$ and b,

$$\frac{1}{2}\mathbf{w}^\top \mathbf{w} + C \sum_i \max\big(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)\big).$$

   The first part of the function,$\frac{1}{2}\mathbf{w}^\top\mathbf{w}$, is quadratic and therefore convex. The function $1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)$ is an affine function and is therefore convex. The 0 term in the max is also linear, which makes it convex. By using the fact that the piecewise maximum of convex functions is also convex, we can say that the max function is convex. Finally, the standard result that the sum of two or more convex functions is also convex can be used to prove that the primal objective function above is convex with respect to $\mathbf{w}$ and b.

6. [2 points] Illustrate how the Occam's Razor principle is reflected in the SVM objective and optimization in Problem 5.

   Occam's Razor principle generally states that we prefer simpler classification rules over complex ones. The SVM problem is a linear classifier, which is one of the simplest classes of classification rules.

7. [3 points] Suppose we have the training dataset shown in Table 1. We want to learn a SVM classifier. We initialize all the model parameters with 0. We set the learning rates for the first three steps to $\{0.01, 0.005, 0.0025\}$. Please list the sub-gradients of the SVM objective w.r.t the model parameters for the first three steps, when using the stochastic sub-gradient descent algorithm.

   Using the following paramaters: $N = 3$, $C = 1$, and augmenting the example vectors with a ones column, we can perform Stochastic sub-gradient descent on the above data.

   For step 1, $\gamma_1 = 0.01$, $\mathbf{w} = \begin{bmatrix}0 & 0 & 0 & 0\end{bmatrix}^T$, $\mathbf{x}_1 = \begin{bmatrix}0.5 & -1 & 0.3 & 1\end{bmatrix}^T$, and $y_1 = 1$:

x_1	x_2	x_3	y
0.5	-1	0.3	1
-1	-2	-2	-1
1.5	0.2	-2.5	1

Table 1: Dataset

$$\begin{aligned} y_1\mathbf{w}^T\mathbf{x}_1 &= 1\begin{bmatrix}0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}0.5\\-1\\0.3\\1\end{bmatrix} \\ &= 0 \\ y_1\mathbf{w}^T\mathbf{x}_1 &\leq 1 \end{aligned}$$

$$\begin{aligned} \mathbf{w} &= (1-\gamma_1)[\mathbf{w}_0;\mathbf{0}] + \gamma_1 CNy_i\mathbf{x}_i \\ &= (0.99)\begin{bmatrix}0\\0\\0\\0\end{bmatrix} + (0.01)(1)(3)(1)\begin{bmatrix}0.5\\-1\\0.3\\1\end{bmatrix} \\ &= \begin{bmatrix}0.015 & -0.03 & 0.009 & 0.03\end{bmatrix}^T \end{aligned}$$

For step 2, $\gamma_2 = 0.005$, $\mathbf{w} = \begin{bmatrix}0.015 & -0.03 & 0.009 & 0.03\end{bmatrix}^T$, $\mathbf{x}_2 = \begin{bmatrix}-1 & -2 & -2 & 1\end{bmatrix}^T$, and $y_2 = -1$

$$\mathbf{w} = \begin{bmatrix}0.029925 & 0.00015 & 0.038955 & -0.015\end{bmatrix}^T$$

Finally for step 3 we get the following using the same calculations as above.

$$\mathbf{w} = \begin{bmatrix}0.0411 & 0.001649 & 0.020108 & 0.0075\end{bmatrix}^T$$

8. [10 points] Let us derive a dual form for Perceptron. Recall, in each step of Perceptron, we add to the current weights $\mathbf{w}$ (including the bias parameter) $y_i\mathbf{x}_i$ for some misclassified example $(\mathbf{x}_i, y_i)$. We initialize $\mathbf{w}$ with $\mathbf{0}$. So, instead of updating $\mathbf{w}$, we can maintain for each training example i a mistake count c_i — the number of times the data point $(\mathbf{x}_i, y_i)$ has been misclassified.

- [2 points] Given the mistake counts of all the training examples, $\{c_1, \ldots, c_N\}$, how can we recover $\mathbf{w}$? How can we make predictions with these mistake counts?
  The following equation will give us $\mathbf{w}$ if we have the mistake counts c_i:

$$\mathbf{w} = \sum_i c_i y_i \mathbf{x}_i$$

- [3 points] Can you develop an algorithm that uses mistake counts to learn the Perceptron? Please list the pseudo code.
  Given a training set $D = \{(\mathbf{x}_i, y_i)\}, \mathbf{x}_i \in \mathbb{R}^n, y_i \in \{-1, 1\}$

```
(a) Initialize $\mathbf{w} = 0 \in \mathbb{R}^n$
(b) For epoch = 1 ... T:
      – Shuffle data
      – For each training example $(\mathbf{x}_i, y_i) \in D$:
```

- [5 points] Can you apply the kernel trick to develop an nonlinear Perceptron? If so, how do you conduct classification? Can you give the pseudo code for learning this kernel Perceptron?

2 Practice [60 points + 10 bonus]

1. [2 Points] Update your machine learning library. Please check in your implementation of Perceptron, voted Perceptron and average Perceptron algorithms. Remember last time you created the folders "Perceptron". You can commit your code into the corresponding folders now. Please also supplement README.md with concise descriptions about how to use your code to run these algorithms (how to call the command, set the parameters, etc). Please create a new folder "SVM" in the same level as these folders.

2. [28 points] We will first implement SVM in the primal domain with stochastic sub-gradient descent. We will reuse the dataset for Perceptron implementation, namely, "bank-note.zip" in Canvas. The features and labels are listed in the file "classification/data-desc.txt". The training data are stored in the file "classification/train.csv", consisting of 872 examples. The test data are stored in "classification/test.csv", and comprise of 500 examples. In both the training and test datasets, feature values and labels are separated by commas. Set the maximum epochs T to 100. Don't forget to shuffle the training examples at the start of each epoch. Use the curve of the objective function (along with the number of updates) to diagnosis the convergence. Try the hyperparameter C from $\{\frac{1}{873}, \frac{10}{873}, \frac{50}{873}, \frac{100}{873}, \frac{300}{873}, \frac{500}{873,} \frac{700}{873}\}$. Don't forget to convert the labels to be in $\{1, -1\}$.

   (a) [12 points] Use the schedule of learning rate: $\gamma_t = \frac{\gamma_0}{1+\frac{\gamma_0}{d}t}$. Please tune γ_0 and d to ensure convergence. For each setting of C, report your training and test error.
   After tuning the parameters, the best performance was achieved at values $\gamma_0 = 0.001$ and $d = 0.01$. The training and test errors for each setting of C are summarized in the table below.

C	Training Error (%)	Test Error (%)
$\frac{1}{873}$	5.046	7.20
$\frac{10}{873}$	4.128	4.80
$\frac{50}{873}$	5.046	6.40
$\frac{100}{873}$	4.014	4.80
$\frac{300}{873}$	4.817	5.60
$\frac{500}{873}$	5.505	6.80
$\frac{700}{873}$	4.817	7.00

Table 2: Percent Errors in soft SVM

(b) [12 points] Use the schedule $\gamma_t = \frac{\gamma_0}{1+t}$. Report the training and test error for each setting of C.

C	Training Error (%)	Test Error (%)
$\frac{1}{873}$	5.046	7.20
$\frac{10}{873}$	4.128	4.80
$\frac{50}{873}$	4.014	4.60
$\frac{100}{873}$	4.014	4.60
$\frac{300}{873}$	3.899	4.80
$\frac{500}{873}$	3.899	4.80
$\frac{700}{873}$	3.899	5.20

Table 3: Percent Errors in soft SVM

(c) [6 points] For each C, report the differences between the model parameters learned from the two learning rate schedules, as well as the differences between the training/test errors. What can you conclude?

For $C = \frac{1}{873}$, the training and test errors were identical for both learning rate schedules, and the weight parameters were as follows:

$$\mathbf{w}_a = \begin{bmatrix} -0.3749 & -0.1718 & -0.1461 & -0.0780 & 9.163\text{e}{-5} \end{bmatrix}$$

$$\mathbf{w}_b = \begin{bmatrix} -0.3735 & -0.1706 & -0.1454 & -0.078\,91 & 9.988\text{e}{-6} \end{bmatrix}$$

For $C = \frac{10}{873}$, the training and test errors were identical for both learning rate schedules, and the weight parameters were as follows:

$$\mathbf{w}_a = \begin{bmatrix} -0.7405 & -0.3731 & -0.3743 & -0.1979 & -0.0009 \end{bmatrix}$$

$$\mathbf{w}_b = \begin{bmatrix} -0.7286 & -0.3682 & -0.3698 & -0.1936 & 9.988\text{e}{-5} \end{bmatrix}$$

For $C = \frac{50}{873}$, the training errors were 1.032% higher for rate schedule A compared to schedule B. The test errors were 1.8% higher for rate schedule A compared to schedule B. The weight parameters were as follows:

$$\mathbf{w}_a = \begin{bmatrix} -1.1302 & -0.5553 & -0.6969 & -0.3104 & 0.0045 \end{bmatrix}$$

$$\mathbf{w}_b = \begin{bmatrix} -1.1044 & -0.5894 & -0.6338 & -0.2476 & 4.9943\text{e}{-4} \end{bmatrix}$$

For $C = \frac{100}{873}$, the training errors were identical for schedules A and B. The test errors were 0.2% higher for rate schedule A compared to schedule B.

The rest of the weight vectors were omitted for clarity and due to time constraints, however the trend seen above remains. These weight vectors can be seen in my Jupyter Notebook file if the grader wishes to see them.

The largest difference in errors between learning rate schedules in part A and B were seen for the C value of $C = \frac{500}{873}$. The training errors were 1.606% higher for rate schedule A compared to schedule B. The test errors were 2.0% higher for rate schedule A compared to schedule B.

Comparing the learned weight parameters, rate schedule A produced larger bias terms compared to rate schedule B. This can be seen above where the last term in the weight vectors is the bias term, b.

The lowest test errors for both rate schedules occurred at $C = \frac{100}{873}$.

Overall, the different values of the regularization term C made very little impact on the performance of this SVM algorithm. In addition the learning rate schedules also contributed little to the performance, as the majority of the errors for all tests were below 5%. An error of 5% is equivalent to predicting the wrong label only 25 out of the 500 test examples.

3. [30 points] Now let us implement SVM in the dual domain. We use the same dataset, "bank-note.zip". You can utilize existing constrained optimization libraries. For Python, we recommend to use "scipy.optimize.minimize", and you can learn how to use this API from the document at `https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.optimize.minimize.html`. For Matlab, we recommend to use the internal function "fmincon"; the document and examples are given at `https://www.mathworks.com/help/optim/ug/fmincon.html`. For R, we recommend to use the "nloptr" package with detailed documentation at `https://cran.r-project.org/web/packages/nloptr/nloptr.pdf`. In principle, you can choose any nonlinear optimization algorithm. But we recommend to use L-BFGS or CG for their robustness and excellent performance in practice.

   (a) [10 points] First, run your dual SVM learning algorithm with C in $\{\frac{100}{873}, \frac{500}{873}, \frac{700}{873}\}$. Recover the feature weights $\mathbf{w}$ and the bias b. Compare with the parameters learned with stochastic sub-gradient descent in the primal domain (in Problem 2) and the same settings of C, what can you observe? What do you conclude and why?

   For the SVM algorithm in the dual domain, I used the python package 'CVX-OPT' to run a Quadratic Programming solver. This solver takes an optimization problem of the following form:

$$\min_x \frac{1}{2} x^T P x - q^T x$$
$$s.t.\ Gx \leq h$$
$$and\ Ax = b$$

   In order to get the dual form SVM optimization problem into this form, a few matrix operations were required. Below is the dual form SVM problem in original form.

$$\min_\alpha \frac{1}{2} \sum_i \sum_j y_i y_j \alpha_i \alpha_j K(x_i, x_j) - \sum_i \alpha_i$$

$$s.t.\ 0 \leq \alpha_i \leq C$$
$$\sum_i \alpha_i y_i = 0$$

   The first step is to define a matrix H such that $H_{i,j} = y_i y_j K(x_i, x_j)$. The K in this equation is the kernel, where a linear kernel was used in part A and a Gaussian kernel was used in part B. Now we can plug this in and convert the sums into vectors as follows:

$$\min_\alpha \frac{1}{2} \alpha^T \mathbf{H} \alpha - \mathbf{1^T} \alpha$$

$$s.t.\ 0 \le \alpha_i \le C$$
$$y^T\alpha = 0$$

This form now matches the form required for the CVXOPT QP solver.

For $C = \frac{100}{873}$, the training error was 1.491% and the test error was 1.4%. The learned weight vector (bias is the last term) was the following:

$$\mathbf{w} = \begin{bmatrix} -0.9430 & -0.6515 & -0.6969 & -0.0410 & 1.5226 \end{bmatrix}$$

For $C = \frac{500}{873}$, the training error was 0.803% and the test error was 0.80%. The weight vector was:

$$\mathbf{w} = \begin{bmatrix} -1.5643 & -1.0138 & -1.1805 & -0.1562 & 1.9175 \end{bmatrix}$$

For $C = \frac{700}{873}$, the training error was 0.803% and the test error was 0.80%. The weight vector was:

$$\mathbf{w} = \begin{bmatrix} -2.0425 & -1.2801 & -1.5132 & -0.2483 & 2.1870 \end{bmatrix}$$

The training and test errors for the dual domain problem were 3 to 4 times better than for the primal problem. Comparing the learned weight vectors, the bias terms for the dual problem were much larger than the bias terms from the primal problem solutions. The other weight vector parameters were similar though.

The test error for $C = \frac{700}{873}$ was 5.2% for the primal problem while only 0.80% for the dual problem. I believe this large difference is mainly due to the use of the Quadratic Programming solver compared to the stochastic sub-gradient descent in the primal problem.

(b) [15 points] Now, use Gaussian kernel in the dual form to implement the non-linear SVM. Note that you need to modify both the objective function and the prediction. The Gaussian kernel is defined as follows:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\gamma}).$$

Test γ from $\{0.01, 0.1, 0.5, 1, 2, 5, 10, 100\}$ and the hyperparameter C from $\{\frac{100}{873}, \frac{500}{873}, \frac{700}{873}\}$. List the training and test errors for the combinations of all the γ and C values. What is the best combination? Compared with linear SVM with the same settings of C, what do you observe? What do you conclude and why?

Table 4 below lists the training and test errors for all combinations of parameters γ and C. C_i corresponds to the ith value from $\{0.01, 0.1, 0.5, 1, 2, 5, 10, 100\}$. As can be seen in the table, The C values had little effect on the errors, it was only for gamma values of 5,10, and 100 where any noticeable difference occurred. For the first 5 gamma values, all of the training errors were zero meaning no example was classified incorrectly. Even for the test data set, the errors were only 0.2%

for these 5 gamma values meaning only a single example out of 500 was classified incorrectly. As the gamma values increased, the errors started to increase, but only slightly. At theses high values of gamma, a trend began where the higher C values corresponded to the lowest errors.

The maximum test error of 0.6%, corresponding to three mislabeled examples, occurred at a gamma of 10 and a regularization value C of 100/873.

γ	Error (%)			Test Error (%)		
	C_1	C_2	C_3	C_1	C_2	C_3
0.01	0.0	0.0	0.0	0.2	0.2	0.2
0.1	0.0	0.0	0.0	0.2	0.2	0.2
0.5	0.0	0.0	0.0	0.2	0.2	0.2
1	0.0	0.0	0.0	0.2	0.2	0.2
2	0.0	0.0	0.0	0.2	0.2	0.2
5	0.8028	0.0	0.0	0.6	0.2	0.2
10	0.8028	0.0	0.0	0.6	0.2	0.2
100	0.344	0.0	0.0	0.4	0.0	0.0

Table 4: Percent Errors in dual form SVM

Overall, both dual forms of the SVM problem performed extremely well on the training and test data sets.

(c) [5 points] Following (b), for each setting of γ and C, list the number of support vectors. When $C = \frac{500}{873}$, report the number of overlapped support vectors between consecutive values of γ, i.e., how many support vectors are the same for $\gamma = 0.01$ and $\gamma = 0.1$; how many are the same for $\gamma = 0.1$ and $\gamma = 0.5$, etc. What do you observe and conclude? Why?

γ	C_1	C_2	C_3
0.01	872	872	872
0.1	869	868	864
0.5	825	730	689
1	805	555	519
2	693	389	359
5	442	208	193
10	316	130	114
100	290	116	98

Table 5: Number of support vectors

For $\gamma = 0.01$ and $\gamma = 0.1$, there were 868 overlap support vectors.

For $\gamma = 0.1$ and $\gamma = 0.5$, there were 730 overlap support vectors.

For $\gamma = 0.5$ and $\gamma = 1$, there were 552 overlap support vectors.

For $\gamma = 1$ and $\gamma = 2$, there were 379 overlap support vectors.

For $\gamma = 2$ and $\gamma = 5$, there were 194 overlap support vectors.

For $\gamma = 5$ and $\gamma = 10$, there were 122 overlap support vectors.

For $\gamma = 10$ and $\gamma = 100$, there were 64 overlap support vectors.

The gamma parameter defines how far the influence of a single training example reaches. Low values of gamma mean that the similarity measure has a high variance, or that two points far from each other will still be considered similar. This makes sense in the data above, as gamma increases, the number of support vectors decreases. Also as the gamma increases, the number of overlapping support vectors decreases faster than the decrease in the support vector count. This means that the support vectors are not similar between different gamma values. This makes sense because this means the variance in the Gaussian Kernel is getting small, points must be close to each other to be considered similar.

(d) [**Bonus**] [10 points] Implement the kernel Perceptron algorithm you developed in Problem 8 (Section 1). Use Gaussian kernel and test γ from $\{0.01, 0.1, 0.5, 1, 2, 5, 10, 100\}$. List the training and test errors accordingly. Compared with the nonlinear SVM, what do you observe? what do you conclude and why?