CS 5350/6350: Machine Learning Fall 2022

Homework 4

Handed out: 8 Nov, 2022
Due date: 11:59pm, 22 Nov, 2022

- You are welcome to talk to other members of the class about the homework. I am more concerned that you understand the underlying concepts. However, you should write down your own solution. Please keep the class collaboration policy in mind.
- Feel free to discuss the homework with the instructor or the TAs.
- Your written solutions should be brief and clear. You do not need to include original problem descriptions in your solutions. You need to show your work, not just the final answer, but you do *not* need to write it in gory detail. Your assignment should be **no more than 15 pages**. Every extra page will cost a point.
- Handwritten solutions will not be accepted.
- *Your code should run on the CADE machines.* **You should include a shell script, `run.sh`, that will execute your code in the CADE environment. Your code should produce similar output to what you include in your report.**

  You are responsible for ensuring that the grader can execute the code using only the included script. If you are using an esoteric programming language, you should make sure that its runtime is available on CADE.
- Please do not hand in binary files! We will *not* grade binary submissions.
- The homework is due by **midnight of the due date**. Please submit the homework on Canvas.

1 Paper Problems [40 points + 10 bonus]

1. [9 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) [3 point] What values ξ_i can take when the training example $\mathbf{x}_i$ breaks into the margin?

(b) [3 point] What values ξ_i can take when the training example $\mathbf{x}_i$ stays on or outside the margin?

(c) [3 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?

Solution:
(a)If the example is correctly classified and break into the margin, ξ_i will be within (0,1). If the example is incorrectly classified, it will cross the margin and ξ_i will be more than 1.
(b)That means ξ_i is 0.
(c)This term is used to minimize the total slack, i.e., allow as few examples as possible to violate the margin. Otherwise the training error would be increased.

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.) Solution:
The primal optimization problem is to calculate
$\underset{\mathbf{w},b,\{\xi_i\}}{min} \frac{1}{2}\mathbf{w}\top\mathbf{w} + C\sum\limits_i \xi_i$ subject to $y_i(\mathbf{w}\top\mathbf{x}_i + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for any i.
So the dual form can be written as
$\underset{\alpha_i \geq 0, \beta_i \geq 0}{max} \underset{\mathbf{w},b,\{\xi_i\}}{min} = \frac{1}{2}\mathbf{w}\top\mathbf{w} + C\sum\limits_i \xi_i - \sum\limits_i \beta_i \xi_i - \sum\limits_i \alpha_i(y_i(\mathbf{w}\top\mathbf{x}_i + b) - 1 + \xi_i)$.
We know that $dual \leq primal$.
In SVM dual = primal according to Slater's condition for convex optimization.
For convex optimization, we know that $\frac{\partial L}{\partial \mathbf{w}} = 0, \frac{\partial L}{\partial b} = 0, \frac{\partial L}{\partial \xi_i} = 0$ to get the optimal of inner level. That is to say:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}} &= \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0 \\ \frac{\partial L}{\partial b} &= \sum_i \alpha_i y_i = 0 \\ \frac{\partial L}{\partial \xi_i} &= C - \beta_i - \alpha_i = 0 \end{aligned} \tag{1}$$

Substitute equation (1) to the original dual form, we can eliminate $\mathbf{w}$ and the optimization problem becomes $\underset{\alpha_i \geq 0, \beta_i \geq 0}{max} -\frac{1}{2}\sum\limits_i\sum\limits_j \alpha_i y_i \mathbf{x}_i^\top \mathbf{x}_j \alpha_j y_j + \sum\limits_i \alpha_i$. In this problem α_i, β_i and y_i should satisfy that $\sum\limits_i \alpha_i y_i = 0$and $\forall i, \alpha_i + \beta_i = C$. We know that $\beta_i \geq 0$, so the constraints can be further simplified by eliminating β_i in it. That is to say, the final dual problem can be written to, $\underset{\{0 \leq \alpha_i \leq C\}, \sum\limits_i \alpha_i y_i = 0}{min} \frac{1}{2}\sum\limits_i\sum\limits_j y_i y_j \alpha_i \alpha_j \mathbf{x}_i^\top \mathbf{x}_j - \sum\limits_i \alpha_i$.

The constraints are $0 \leq \alpha_i \leq C, \sum\limits_i \alpha_i y_i = 0$.

3. [10 points] Continue with the dual form. Suppose after the training procedure, you have obtained the optimal parameters.

   (a) [4 points] What parameter values can indicate if an example stays outside the margin?

   (b) [6 points] 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.
   Solution:
   (a)$a_j = 0$ indicates the examples are outside the margin.
   (b)If $0 < a_j^* < C$, it indicates the examples just sit on the margin.

4. [6 points] How can we use the kernel trick to enable SVMs to perform nonlinear classification? What is the corresponding optimization problem?
   Solution: By defining a dot product in the high dimensional space we can play a kernel trick: $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^\top \phi(\mathbf{z})$.
   We know $\mathbf{w} = \sum\limits_i \alpha_i y_i \mathbf{x}_i$, thus we can plug the kernel into the dual form as:
   $\underset{\{0\le\alpha_i\le C\},\sum\limits_i \alpha_i y_i = 0}{min} \frac{1}{2}\sum\limits_i\sum\limits_j y_i y_j \alpha_i \alpha_j \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) - \sum\limits_i \alpha_i$, which is the corresponding optimization problem.

5. [9 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.

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

   Solution:
   N=3,$C = \frac{1}{3}$, $\mathbf{w}^0 = [\mathbf{w}_0; b] = [0, 0, 0, 0; 0]\top$,
   For epoch 1, we randomly choose an example 1 and calculate the prediction $\mathbf{w}\top\mathbf{x}_i$, which is 0. So gradient updating for this step is $\mathbf{w}^1 = \mathbf{w}^0 - \gamma_1[\mathbf{w}_0^0; 0] + \gamma_1 \cdot C \cdot N y_1 \mathbf{x}_1 = [0.005, -0.01, 0.003, 0.01]\top$.
   For epoch 2,we randomly choose an example 3 and calculate the $y_i\mathbf{w}\top\mathbf{x}_i = 1*0.008 \le 1$. So $\mathbf{w}^2 = \mathbf{w}^1 - \gamma_2[\mathbf{w}_0^1; 0] + \gamma_2 \cdot C \cdot N y_3 \mathbf{x}_3 = [0.0125, -0.009, -0.01, 0.015]\top$.
   For epoch 3, we choose example 2. Similarly we get $\mathbf{w}^3 = \mathbf{w}^2 - \gamma_3[\mathbf{w}_0^2; 0] + \gamma_3 \cdot C \cdot N y_2 \mathbf{x}_2 = [0.015, -0.004, -0.005, 0.0125]\top$.

6. [**Bonus**][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?
- [3 points] Can you develop an algorithm that uses mistake counts to learn the Perceptron? Please list the pseudo code.
  Solution:

```
Algorithm: Dual Perceptron
initialize $\mathbf{a} = 0, b = 0, \gamma = 0.1$
repeat
    for i=1 to N do
        if $y_i(\sum_{j=1}^{l} \alpha_j y_j \mathbf{x}_j^\top \mathbf{x}_i + b) \leq 0$ then
            $a_i = a_i + \gamma$;
            $b = b + \gamma y_i$;
        end
    end
until no mistakes are made in the for loop;
return $\mathbf{a}, b$ to define h;
$h(x) = sign(\sum_{j=1}^{N} \alpha_j y_j x_j \mathbf{X} + b)$.
```

- [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 fo 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 subgradient 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 $\{\frac{100}{873}, \frac{500}{873,} \frac{700}{873}\}$. Don't forget to convert the labels to be in $\{1, -1\}$.

(a) [12 points] Use the schedule of learning rate: $\gamma_t = \frac{\gamma_0}{1+\frac{\gamma_0}{a}t}$. Please tune $\gamma_0 > 0$ and $a > 0$ to ensure convergence. For each setting of C, report your training and test error.
Solution:
The C sets and their corresponding errors are listed in table 2. $\gamma_0 = 0.05$ and $a = 0.1$ are used.

Table 2: Results of Stochastic sub-gradient descent for SVM($\gamma_t = \frac{\gamma_0}{1+\frac{\gamma_0}{a}t}$).

C	Training error	Testing error
100/873	0.0539	0.058
500/873	0.0449	0.054
700/873	0.0528	0.052

(b) [12 points] Use the schedule $\gamma_t = \frac{\gamma_0}{1+t}$. Report the training and test error for each setting of C.
Solution:
Use $\gamma_0 = 0.01$ to ensure convergence and reduce the error.The C sets and their corresponding errors are listed in table 3.

Table 3: Results of Stochastic sub-gradient descent for SVM($\gamma_t = \frac{\gamma_0}{1+t}$).

C	Training error	Testing error
100/873	0.068	0.074
500/873	0.080	0.086
700/873	0.056	0.074

(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?

Solution:
If taking average from every model,I can see from my results that the errors using $\gamma_t = \frac{\gamma_0}{1+\frac{\gamma_0}{a}t}$ is smaller than the errors when using $\gamma_t = \frac{\gamma_0}{1+t}$. I got the results for the former one using $\gamma_0 = 0.05$ and $\gamma_0 = 0.01$ for the later errors. The errors in table 2 are obviously smaller than table 3 despite this.
In all cases the testing error is bigger than the training error, which is reasonable. By comparing errors between different C, there is no significant difference if taking average from results in every model. But generally speaking, the probability of a good outcome

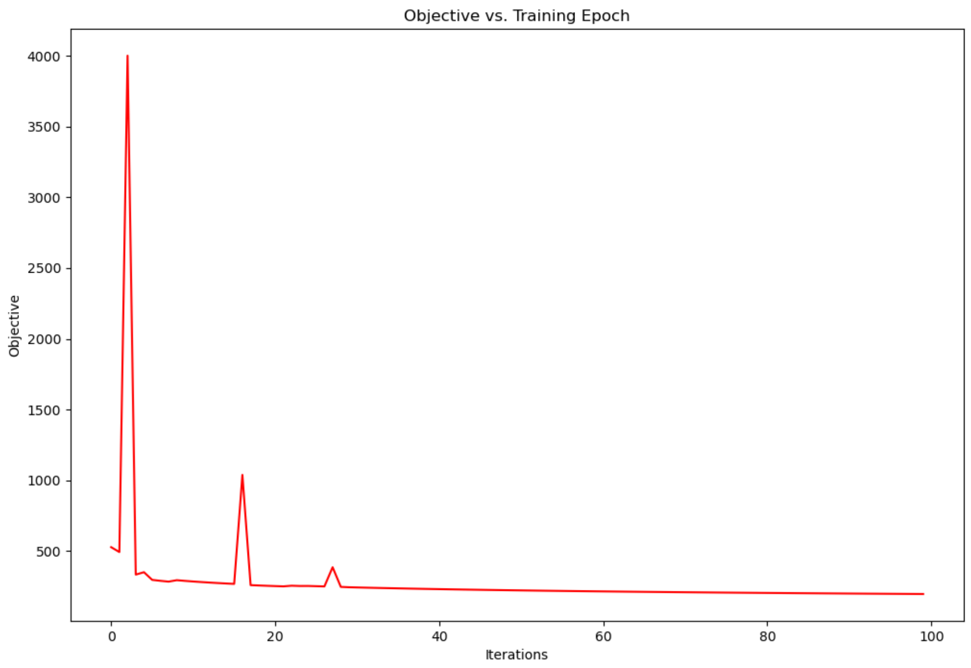

Figure 1: Objective vs iterations

is bigger when C is smaller, and vice versa.
Besides, when using stochastic sub-gradient descent for SVM, we can diagnosis convergence by the objective function. If we use the same method in calculating sub-gradient to calculate the objective function, i.e. we use an example to calculate the objective by repeating it N times. The objective function along with iterations can be like fig 1. Singularity occurs. And if we use the original loss objective function that will be more obvious. 2

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 using "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`. We recommend using SLSQP to incorporate the equality constraints. For Matlab, we recommend using the internal function "fmincon"; the document and examples are given at `https://www.mathworks.com/help/optim/ug/fmincon.html`. For R, we recommend using the "nloptr" package with detailed documentation at `https://cran.r-project.org/web/packages/nloptr/nloptr.pdf`.

   (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

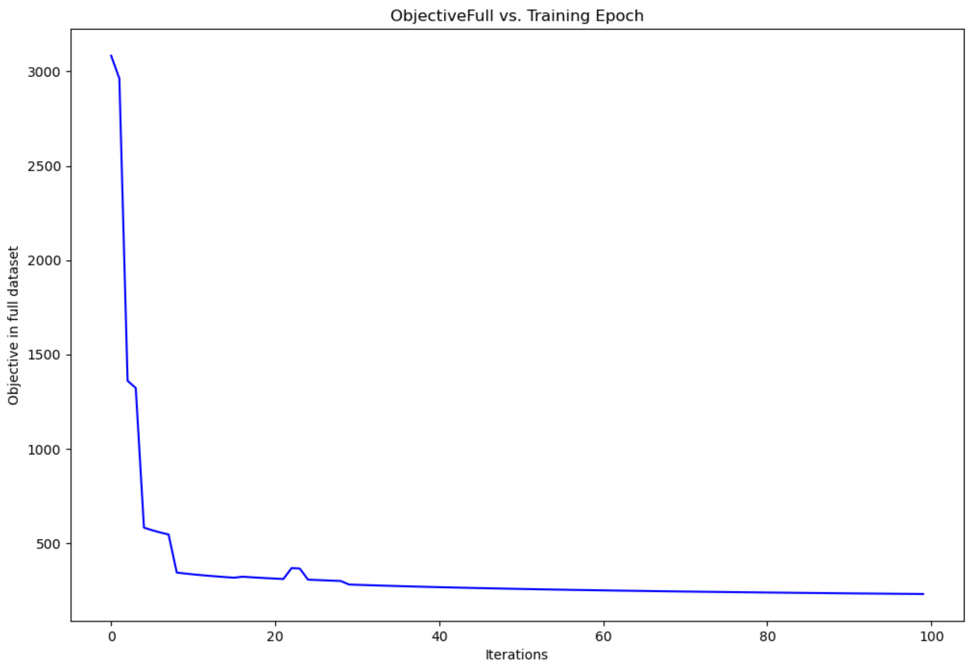

Figure 2: Objective in full dataset vs iterations

2) and the same settings of C, what can you observe? What do you conclude and why? Note that if your code calculates the objective function with a double loop, the optimization can be quite slow. To accelerate, consider writing down the objective in terms of the matrix and vector operations, and treat the Lagrange multipliers that we want to optimize as a vector! Recall, we have discussed about it in our class.
Solution:
To avoid using double for loop, we can derive the calculation of

$$\begin{aligned}\underset{\{0\leq\alpha_i\leq C\},\sum_i \alpha_i y_i=0}{min} \frac{1}{2}\sum_i\sum_j y_i y_j \alpha_i \alpha_j \mathbf{x}_i \top \cdot \mathbf{x}_j - \sum_i \alpha_i \\ = \underset{\{0\leq\alpha_i\leq C\},\sum_i \alpha_i y_i=0}{min} \frac{1}{2}\boldsymbol{\beta}\top\mathbf{M}\boldsymbol{\beta} - \sum_i \alpha_i. \\ \boldsymbol{\beta} = (\alpha_i * y_i)_{[N,1]}, \mathbf{M} = [\mathbf{x}_i \cdot \mathbf{x}_j^\top]_{[N,N]}.\end{aligned} \quad (2)$$

In scipy.optimize.minimize I chose a tolerance of 1e-3, and the optimized weight vectors and bias are shown in table 4 Comparing with parameters learned in table !5, we can see that solving dual form using Scipy is much more robust than stochastic sub-gradient descent solution. But the stochastic one is an efficient way for SVM, which is accurate and computationally cheap. Scipy.optimize takes more time even using matrix form. Parameters learned by these two methods are

different, but symbols of every weight factor learned are the same.
Besides, the smaller the C is, the smaller the testing error is. We concluded in problem 2(c) that the probability of a good outcome is bigger when C is smaller, which is exactly the same as the conclusion here.

Table 4: Results of dual SVM using Scipy.

C	Weights	Bias	Testing error
100/873	[-0.95,-0.65,-0.73,-0.04]$^\top$	3.34	0.058
500/873	[-1.56,-1.01,-1.18,-0.16]$^\top$	6.27	0.088
700/873	[-2.03,-1.27,-1.50,-0.24]$^\top$	8.431	0.106

Table 5: $[b; \mathbf{W}]$ learned in primal SVM.

C	$\gamma_t = \frac{\gamma_0}{1+\frac{\gamma_0}{a}t})$	$\gamma_t = \frac{\gamma_0}{1+t}$
100/873	[-2.87,-16.91,-5.94,-7.75,0.32]$^\top$	[0.18,-5.30,-4.52,-0.84,-0.79]$^\top$
500/873	[21.37,-59.90,-33.04,-38.89,-13.95]$^\top$	[2.15,-17.97,-8.73,-5.95,-10.47]$^\top$
700/873	[19.37,-177.10,-87.03,-77.00,-36.20]$^\top$	[-5.20,-39.14,-22.83,-13.68,-8.16]$^\top$

(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 $\{0.1, 0.5, 1, 5, 100\}$ 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?
Solution:
Similarly we can write the function as matrix form as below,

$$\begin{aligned}
&\underset{\{0\le\alpha_i\le C\},\sum_i \alpha_i y_i=0}{min} \frac{1}{2}\sum_i\sum_j y_i y_j \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) - \sum_i \alpha_i \\
&= \underset{\{0\le\alpha_i\le C\},\sum_i \alpha_i y_i=0}{min} \frac{1}{2}\boldsymbol{\beta}^\top \mathbf{A}\boldsymbol{\beta} - \sum_i \alpha_i \\
&\boldsymbol{\beta} = (\alpha_i * y_i)_{[N,1]}, \mathbf{A} = [\exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\gamma})]_{[N,N]} \\
&= [\exp(-\frac{\mathbf{x}_i\mathbf{x}_i^\top + \mathbf{x}_j\mathbf{x}_j^\top - 2\mathbf{x}_i\mathbf{x}_j^\top}{\gamma})]_{[N,N]}.
\end{aligned} \tag{3}$$

Results are show in table 6. We can see the testing error is very small especially when γ is small. Zero error occurs when γ is small and C gets bigger. The best combination here is $\gamma = 0.1, C = \frac{700}{873}$. We can get bigger errors when C gets bigger in linear SVMs, which is different from nonlinear SVM.
And we can see nonlinear SVM is much more robust and accurate. In Gaussian kernel, the parameter γ is smaller, the easier to separate the dataset in mapped space.

Table 6: Gaussian kernel for dual SVM.

C	γ	Errors	
		Training Error	Testing Error
100/873	0.1	0.003	0.004
	0.5	0.018	0.018
	1	0.023	0.02
	5	0.071	0.088
	100	0.168	0.156
500/873	0.1	0.0	0.0
	0.5	0.003	0.004
	1	0.013	0.014
	5	0.022	0.02
	100	0.133	0.144
700/873	0.1	0.0	0.0
	0.5	0.0	0.0
	1	0.01	0.008
	5	0.02	0.02
	100	0.13	0.142

(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?
Solution: Numbers of support vectors are shown in table 7. We can see that when the accuracy of the trained model is high, the numbers of support vectors is relatively small.
When C=$\frac{500}{873}$, the number of support vectors are shown in table 8

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

Table 7: Support vectors for nonlinear SVM.

C	γ	Support vectors
		Training data
100/873	0.1	267
	0.5	344
	1	441
	5	696
	100	796
500/873	0.1	109
	0.5	150
	1	192
	5	409
	100	786
700/873	0.1	96
	0.5	128
	1	164
	5	342
	100	783

Table 8: Support vectors for nonlinear SVM.

C	γ	Support vectors
		Training data
500/873	0.01	333
	0.05	121
	0.1	109
	0.2	123
	0.5	150