CSCI 5521: Introduction to Machine Learning (Fall 2019)[1]

Homework 5

1. (**40 points**) Given the ν-SVM (Section 13.4 in the book), consider a variation of its objective function defined as:

$$\min_{w,\xi,\rho} \frac{1}{2} w^T S w - \nu\rho + \sum_t C^t \xi^t, \tag{1}$$

where S is positive definite and $C^t > 0$ for $\forall t$. This variation is a combination of using ρ to rescale the margin and factor C for penalizing the soft margin. In some real applications, we have samples of different importance, in which small or no training error is allowed for important samples. This can be implemented with a different penalty factor C^t for each (x^t, r^t) instead of a uniform C for all. Start from the primal form and show each step to derive the Lagrange dual of the SVM variation. (Hint: Read Section 13.3 carefully to see how Equation 13.15 is derived from Equation 13.10).

2. (**60 points**) Perceptron algorithm can be kernelized to allow non-linear mapping. A simple version of kernel perceptron algorithm is to simply replace coefficients w with sample weight α as $f(x) =< w, x > +b = \sum_i \alpha^i y^i < x^i, x > +b$ (note: you can also stack 1 as the last feature in x to represent the bias term b within w). The kernel perceptron algorithm be equivalently trained as the original perceptron algorithm as follows,

$\alpha = \{0\}^N$;
$b = 0$;
while not converged **do**
  for each sample (x^t, y^t) **do**
    if $(\sum_{i=1}^N \alpha^i y^i < x^i, x^t > +b) y^t \leq 0$ **then**
      $\alpha^t = \alpha^t + 1$
      $b = b + y^t$
    end if
  end for
end while

[1]Instructor: Rui Kuang (kuang@umn.edu). TA: Rachit Jas (jas00001@umn.edu) and Tianci Song (song0309@umn.edu)

(a) Implement this kernel perceptron using a polynomial kernel (try different degree) and train it on non-linearly separable data given by the following Python code:

```
import numpy as np
np.random.seed(1) # For reproducibility
r1 = np.sqrt(np.random.rand(100, 1)) # Radius
t1 = 2*np.pi*np.random.rand(100, 1) # Angle
data1 = np.hstack((r1*np.cos(t1), r1*np.sin(t1))) # Points
np.random.seed(2) # For reproducibility
r2 = np.sqrt(3*np.random.rand(100, 1)+2) # Radius
t2 = 2*np.pi*np.random.rand(100, 1) # Angle
data2 = np.hstack((r2*np.cos(t2), r2*np.sin(t2))) # Points
```

Data from class 1 is contained in the matrix `data1` and data from class 2 is contained in the matrix `data2`. Use the following code to visualize the data:

```
import matplotlib.pyplot as plt
plt.scatter(data[np.where(labels==1)[0],0],
data[np.where(labels==1)[0],1], c='r')
plt.scatter(data[np.where(labels==-1)[0],0],
data[np.where(labels==-1)[0],1], c='b')
```

Combine the data into a single matrix and assign class labels $\{1, -1\}$ to use to test your kernel perceptron algorithm using the following code:

```
data3 = np.vstack((data1, data2))
labels = np.ones((200, 1))
labels[0:100, :]  = -1
```

All the data is contained in matrix `data3` and the corresponding labels are in vector `theclass`.

Test your implementation of the kernel perceptron algorithm on the above data and plot the decision boundary (hint: follow the explanations in footnote [2] (matlab code), and then use similar functions provided in Python: `meshgrid` function from numpy and `contour` function from matplotlib to draw the boundary, and footnote [3] shows a demo to do contour plotting by using these two functions) and report the error rate.

[2] https://www.mathworks.com/help/stats/support-vector-machines-for-binary-classification.html

[3] https://matplotlib.org/3.1.1/gallery/images_contours_and_fields/contour_demo.html

(b) Apply the `svc` function in svm module from sklearn package [4] using the same kernel to the same data and plot the decision boundary obtained by the SVM in the same figure with the decision boundary of your kernel perceptron. Compare the results with your kernel perceptron implementation. Play with the `C` penalty parameter and explain how it changes the margin obtained.

(c) Once you have tested that your kernel perceptron works, train and evaluate your implementation using the given subset of the optdigits dataset. The first training and test datasets `digits49_train.txt` and `digits49_test.txt` consists of only digits 4 and 9. The second training and test datasets `digits79_train.txt` and `digits79_test.txt` consists of only digits 7 and 9. Report the training and test error rates on both datasets.

Instructions

- Solutions to all questions must be presented in a report which includes result explanations, and all error rates.
- All programming questions must be written in Python, no other programming languages will be accepted. The code must be able to be executed from either command line or PyCharm window on the cselabs machines. Each function must take the inputs in the order specified and print/display the required output to either terminal or PyCharm console. For each part, you can submit additional files/functions (as needed) which will be used by the main functions specified below. Put comments in your code so that one can follow the key parts and steps. **Please follow the rules strictly. If we cannot run your code, you will receive no credit.**
- **Question 2:**
  - Train the kernel perceptron: *kernPercGD*(*train_data*: training data matrix, *train_label*: training label vector). The function must return in variables the outputs (α: $n \times 1$ vector, b: a scalar).
- For the optdigit datasets, the last column is the label $\{-1, 1\}$.

[4] `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`

Submission

- **Things to submit:**
  1. hw5_sol.pdf: A PDF document which contains the report with solutions to all questions.
  2. kernPercGD.py: The Python code of the *kernPercGD* function in Question 2.
  3. hw5_Q2.py: Main script for Question 2 which will learn the model and print the error rates for each of the three datasets.
  4. Any other files, except the data, which are necessary for your code.
- **Submit**: Upload hw5_sol.pdf as a separate file and a zip file which is the compression of all other files. All material must be submitted electronically via Canvas.