Assignment (5)

Artificial Intelligence and Machine Learning (24-787 Fall 2021)

Due Date: 11/20/2021 (Friday) @ 11:59 pm EST

In case a problem requires programming, it should be programmed in Python. In Programming, you should use plain Python language, unless otherwise stated. For example, if the intention of a Problem is familiarity with numpy library, it will be clearly noted in that problem to use numpy. Please submit your homework through Gradescope.

Submissions: There are two steps to submitting your assignment on Gradescope:

1. HW05 Writeup: Submit a combined pdf file containing the answers to theoretical questions as well as the pdf form of the FILE.ipynb notebooks.

- Convert the jupyter notebook to a pdf file. Ensure that the submitted notebooks have been run and the cell outputs are visible - **Hint:** Restart and Run All option in the Kernel menu. Make sure all plots are visible in the pdf.
- If an assignment has theoretical and mathematical derivation, scan your handwritten solution and make a PDF file.
- Then concatenate them all together in your favorite PDF viewer/editor. The file name (FILE)should be saved as **HW-assignmentnumber-andrew-ID.pdf**. For example for assignment 1, my FILE = HW-5-andrewid.pdf
- Submit this final PDF on Gradescope, and **make sure to tag the questions correctly!**

2. HW05 Code: Submit a ZIP folder containing the FILE.ipynb notebooks for each of the programming questions. The ZIP folder containing your iPython notebook solutions should be named as HW-assignmentnumber-andrew-ID.zip

You can refer to Numpy documentation while working on this assignment. **ANY** deviations from the submission structure shown below would attract penalty to the assignment score. Please use Piazza for any questions on the assignment.

PROBLEM 1

Support Vector Machines **[30 points]**

In this problem, you'll practice to solve a classification problem using SVM. In class, we have seen how to formulate SVM as solving a constrained quadratic optimization problem. Now, you will implement an SVM in the primal form. Conveniently, the **cvxopt** module in python provides a solver for constrained quadratic optimization problem, which does essentially all of the work for you. This solver can solve arbitrary constrained quadratic optimization problems of the following form:

$$\begin{aligned} \arg\min_{\mathbf{z}} \quad & \frac{1}{2}\mathbf{z}^T\mathbf{Q}\mathbf{z} + \mathbf{p}^T\mathbf{z} \\ \text{s.t.} \quad & \mathbf{G}\mathbf{z} \leq \mathbf{h} \end{aligned} \tag{1}$$

a) (Programming problem) You are given a data file **clean_lin.txt**. The first two columns are coordinates of points, while the third column is label.

Now let's implement the following quadratic optimization problem:

$$\begin{aligned} \arg\min_{\mathbf{w},b} \quad & \frac{1}{2}||\mathbf{w}||^2 \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1, \quad (i = 1, 2, \cdots) \end{aligned} \tag{2}$$

To do this, you have to find how to turn this constrained optimization problem into the standard form shown in (1). Things you should keep in mind: which variable(s) does **z** need to represent? What do you need to construct **Q**, **p**, **G** and **h**?

Hint: **z** should be 3×1. **G** should be $n \times 3$, where n is the number of training samples.

Train the linear SVM on the data using ***cvxopt.solvers.qp(Q,p,G,h)***. Plot the decision boundary and margin boundaries. You should have a plot similar to Fig. 1

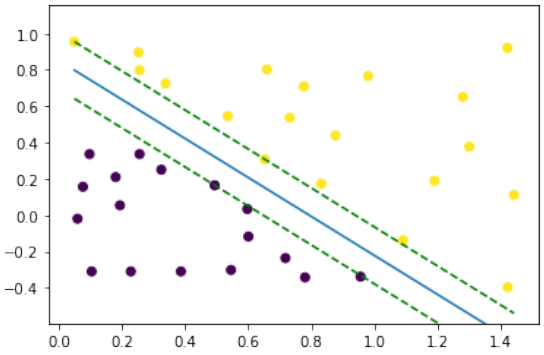

Fig. 1: Expected decision boundary for the linearly separable dataset.

b) (Programming problem) Now let's go ahead to solve a linearly non-separable case using SVM. Load the training data in **dirty_nonlin.txt**.

As discussed in the lecture, introducing slack variables for each point to have a soft-margin SVM can be used for non-separable data. The soft-margin SVM, which has following form:

$$\begin{aligned} \arg\min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - \xi_i, \quad (i = 1, 2, \cdots) \\ & \xi_i \geq 0, \quad (i = 1, 2, \cdots) \end{aligned} \tag{3}$$

At this point, use $C = 0.05$ in your code, but keep that as a variable because you will be asked to vary C in the subsequent questions. Again, you want to think about what the terms in the standard formulation represent now.

Hint: The problem is still quadratic in the design variables. So your solution, if found, will be the global minimum. **z** should be $(n + 3) \times 1$. **G** should be $2n \times (n + 3)$. If you construct your design vector as $[w_1, w_2, b, \xi_1, \cdots, \xi_n]$, you shall see that **G** can be constructed by putting together four sub matrices (upper left 3×3, lower right $n \times n$ etc.).

Finally, plot the decision boundary and margin boundaries. You should expect to have a plot similar to Fig. 2.

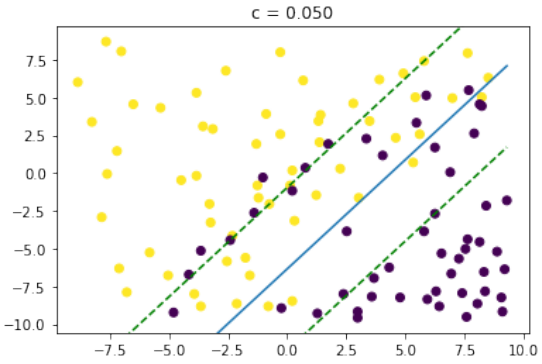

Fig. 2: Decision boundary for the linearly non-separable dataset.

c) (Programming problem) Use your code in **b)** to draw 4 plots, each corresponding to different values of $C = [0.1, 1, 100, 1000000]$. Discussion your observations of the decision margins.

(Problem 2 on next page)

PROBLEM 2

House price prediction **[40 points]**

In this problem, you'll get some hands-on experience on a real-world dataset. Assume you already have a model that could predict the house price, you are going to predict the residual error of this model. Specifically, your target is the error in logarithmic scale (**logerror**), and your inputs are some features that may contribute to the house price, like the number of bathrooms, location, etc.

Data Description

You are given three .csv files, namely **train.csv**, **properties.csv** and **data_dictionary.csv**. The **train.csv** includes transaction id, log error, and transaction date. The **properties.csv** includes the transaction id and a list of features. The **data_dictionary.csv** gives some explanation of what each feature means.

a) (Programming problem) Firstly, let's load and visualize data.

1. Use pandas to load **train.csv** and **properties.csv** into two DataFrame. Merge them into a new DataFrame based on the transaction id.
2. Since there are some outliers at both ends of log error, replace these outliers with proper maximum/minimum value. Specifically, replace log error from 0% to 1% with value at 1%, and 99% to 100% with value at 99%. (HINT: You can use np.percentile)
3. Make a scatter plot and histogram of **logerror**. (HINT: You should find logerror follows a nice normal distribution)

b) (Programming problem) Usually, a dataset will have a lot of missing values (NaN), so we'll first do data cleaning before moving to the next part.

1. Build a new DataFrame that has two columns: **"column_name"** and **"missing_count"**. The **"column_name"** contains every column in merged DataFrame. The **"missing_count"** counts how many missing data that certain column has.
2. Adding a new column called **"missing_ratio"** to this new DataFrame. This new column stores the ratio of number of missing data to the number of total data.
3. Fill the missing data of each feature in merged DataFrame (the df you got from **a**) by its mean value.

c) (Programming problem) At this point, we can do some univariate analysis. For this problem, we will look at the correlation coefficient of each of these variables.

1. For each variable, compute the correlation coefficient with logerror. After that, sort and make a bar chart of these coefficients.
2. If your bar chart is right, there are few variables at the top of this chart without any correlation values. Explain why.

d) (Programming problem) Now it's time to apply some non-linear regression models.

1. To simply, in this problem we only use float value features. Therefore, drop the categorical features, "id" and "transactiondate" in your merged dataset.
2. Split your data into train and test following the 70/30 ratio. Use **sklearn.ensemble.RandomForestRegressor** to train your data.
3. Report the importance of each feature using bar chart. Also, report the Mean Square Error (MSE) of test set.

e) (Programming problem) Cross-validation is a useful way to avoid overfitting. In problem **e**, only use the first 500 samples of your dataset in **d.2**.

1. Use **sklearn.model_selection.KFold** to implement KFold cross validation with fold=5. Print the overall MSE and compare with your result in **d.2**.
2. Run your algorithm in **d.2** 100 times with random seeds from 0 to 99. Use the same train/test ratio. Report the MSE of each prediction. Explain your findings and what's the advantage of cross-validation. (HINT: Randomly running your algorithm in d.2 means you should pass random seed into both your "train_test_split" process and random forest model)

(Problem 3 on next page)

PROBLEM 3

Flower Classification using CNN **[30 points]**

In this question, you will apply Convolution Neural Networks (CNN) to classify different flowers. Specifically, you will use Pytorch to build CNN models and evaluate the accuracy. The dataset is included in **Image.zip**, and sample code is also provided for you to help develop the model. Follow the instructions in the sample code and answer the questions below

a) Data Preprocessing and visualization - 5 points: Firstly, you need to load the dataset and plot it to make sure you are using the dataset correctly. The data is stored in a file called **Image.zip**. The zip file needs to be unzipped first. You can load the data using the built-in Pytorch dataset called ***ImageFolder*** and visualize the image using the code provided in the template. Each RGB image has (3,224,224) pixels, and the label denotes the type of flowers for this image. There are 4317 images in total and five types of flowers. You should have a similar image to this

Please design your own image augmentation method and visualize one image. Print the type of flower for this image

b) Training the CNN model with Cross Entropy Loss - 15 points: With the data loaded above, follow the instructions provided in the sample code to build your own CNN model. We recommend using **Colab** in this question for faster training. Colab provides a free GPU for you to train a CNN model. A sample CNN model is built in the python script for reference. You should refer to the python file for more detail. Here is a sample architecture:

```
CNNModel(
  (cnn1): Conv2d(3, 16, kernel_size=(5, 5), stride=(1, 1))
  (relu1): ReLU()
  (maxpool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=193600, out_features=5, bias=True)
)
```

You should build your own model and achieve at least 70% accuracy to get a full mark. The estimated training time for a shadow CNN model (two convolution layers) is around 10 mins using a Colab GPU.
You need to do the following:

- Plot your accuracy versus iterations
- Plot your loss versus iterations
- Make predictions for a batch of sample images
- Visualize all your filters in the first convolution layer