Assignment (5)

Artificial Intelligence and Machine Learning (24-787 Fall 2019)

Due Date: 2019/11/23 (Sat) @ 11:59 pm EST
All the files must be put into one directory **andrewID-HW5/**. 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. Zip the directory **andrewID-HW5/** and submit the zip on canvas. 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 Pizza for any questions on the assignment.

andrewID-HW5/

q1.ipynb
q1.pdf
q2.ipynb
q2.pdf
q3.ipynb
q3.pdf

Submission file structure

PROBLEM 1

Support Vector Machines **[40 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

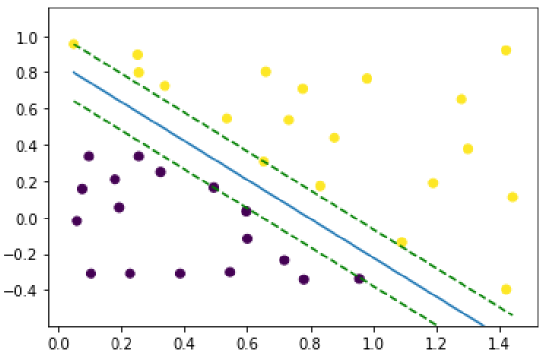

Fig. 1: Expected decision boundary for the linearly separable dataset.

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:

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

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.

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.

d) (Theoretical problem) SVM can also be used to perform non-linear classification with a kernel trick. Recall the hard-margin SVM in (2), which is the primal form of SVM. The dual of this primal problem can be specified as a procedure to learn the following linear classifier:

$$f(x) = \sum_i \alpha_i y_i ({x^{(i)}}^T x) + b \tag{4}$$

Here $x^{(i)}$ and y_i are training data and x is the new sample to be classified. Note that now we can replace ${x^{(i)}}^T x$ with a kernel $K(x^{(i)}, x)$, and have non-linear decision boundary.

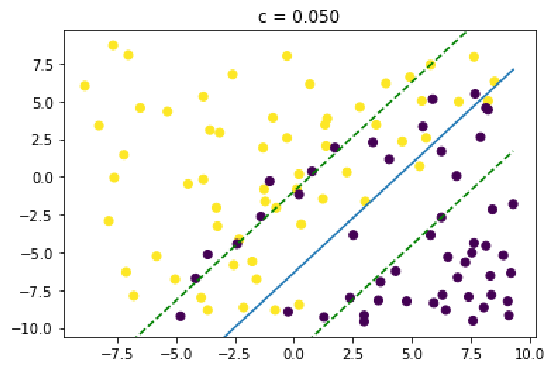

Fig. 2: Decision boundary for the linearly non-separable dataset.

In Fig. 3, there are different SVMs with different shapes/patterns of decision boundaries. The training data is labeled as $y_i \in \{-1, 1\}$, represented as the shape of circles and squares respectively. Support vectors are drawn in solid circles. Match the scenarios described below to one of the 6 plots (note that one of the plots does not match to anything). Each scenario should be matched to a unique plot. Explain **briefly** why it is the case for each scenario.

1. A soft-margin linear SVM with $C = 0.02$.
2. A soft-margin linear SVM with $C = 20$.
3. A hard-margin kernel SVM with $K(u, v) = u \cdot v + (u \cdot v)^2$.
4. A hard-margin kernel SVM with $K(u, v) = \exp(-5||u - v||^2)$.
5. A hard-margin kernel SVM with $K(u, v) = \exp(-\frac{1}{5}||u - v||^2)$.

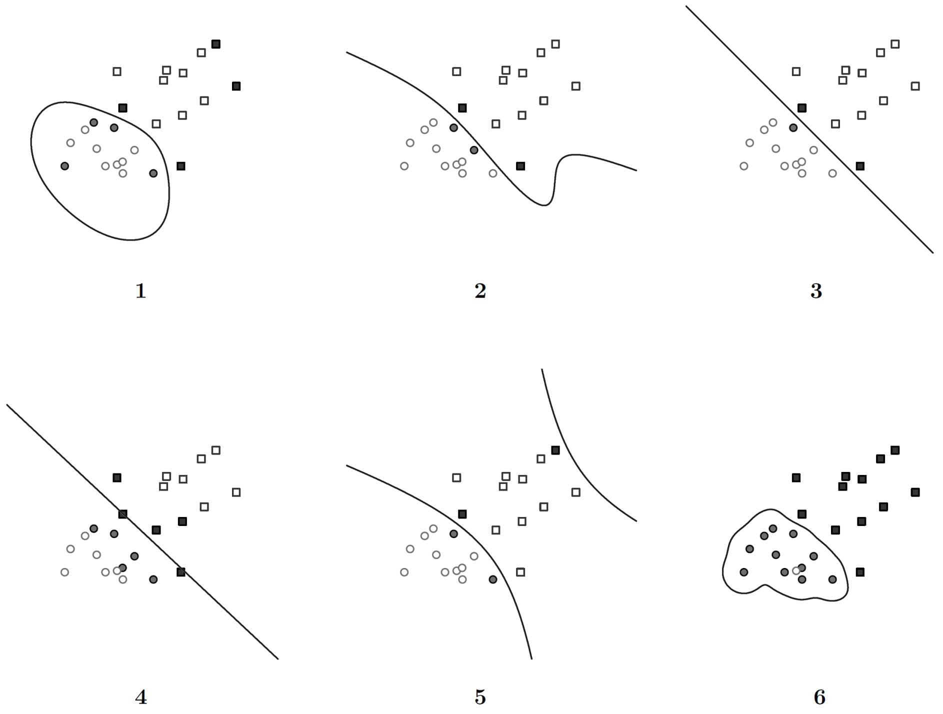

Fig. 3: SVM Kernel boundaries.

PROBLEM 2

Independent Component Analysis (ICA) **[20 points]**

This problem is designed to help you implement the **FastICA** algorithm. In the q2.ipynb file, we have created 3 different signals and mixed them together in **X** of size (3, n), meaning we have n=2000 observations and 3 components.

The goal of the FastICA algorithm is to compute a weighting matrix **W** (3, 3) such that the true signal **S** of size (3, n) can be recovered by $\mathbf{S} = \mathbf{W}^T\mathbf{X}$. The algorithm described by the following procedures:

1. Center **X** by substracting the mean: $\mathbf{X} = \mathbf{X} - \text{mean}(\mathbf{X})$;
2. Whiten the data: $\tilde{\mathbf{X}} = \mathbf{U}\mathbf{D}^{-1/2}\mathbf{U}^T\mathbf{X}$, where **U** and **D** are the orthogonal matrix of eigenvectors and the diagonal matrix of eigenvalues;
3. Choose a random initial value for **W**: $\mathbf{W} = \text{rand}(3, 3)$;
4. For each column $\mathbf{w}_i$ $(i = 1, 2, 3)$ in **W**:
4.1 Update: $\mathbf{w}_i^+ = \frac{1}{n}\sum_{p=1}^{n}\mathbf{x}_p g(\mathbf{w}_i^T\mathbf{x}_p) - \frac{1}{n}\sum_{p=1}^{n} g'(\mathbf{w}_i^T\mathbf{x}_p)\mathbf{w}_p$. Here $g(u) = \tanh(u), g'(u) = 1 - \tanh^2(u)$. $\mathbf{x}_p$ is the p-th observation and is of size (3,1).
4.2 Normalize: $\mathbf{w}_i^+ = \frac{\mathbf{w}_i^+}{||\mathbf{w}_i^+||}$
4.3 Decorrelation: $\mathbf{w}_i^+ = \mathbf{w}_i^+ - \sum_{j=1}^{i-1}(\mathbf{w}_i^{+T}\mathbf{w}_j)\mathbf{w}_j$. Note that j starts at 1 and ends at $i - 1$.
4.4 Check convergence: if $|\mathbf{w}_i^T\mathbf{w}_i^+ - 1| < 10^{-5}$, converged.
4.5 Update **W** with $\mathbf{w}_i = \mathbf{w}_i^+$
4.6 If not converged in 4.4, go to 4.1 and repeat.
5. return **W** and $\mathbf{S} = \mathbf{W}^T\mathbf{X}$

That's it, Hooray! This is basically what is described in the section 5 and 6 of the paper 'Independent Component Analysis: Algorithms and Applications' by Aapo Hyvärinen and Erkki Oja. You may use the paper as a knowledge reference but the above procedures should help you to implement FastICA. Additionally, please make sure you understand what the above procedures are doing before implementation. Finally, please follow the template in q2.ipynb to implement this algorithm.

If your implementation is correct, you should be able to see the reconstructed signals in the final plot similar to Fig. 4, where different colors indicate different signals.

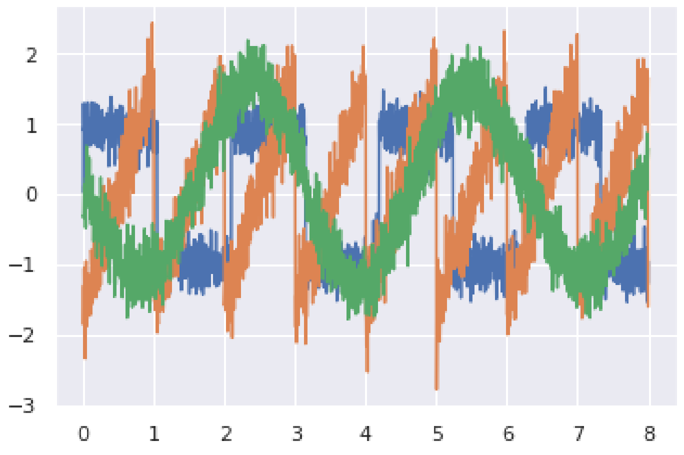

Fig. 4: Reconstructed signals **S**.

PROBLEM 3

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)
