Programming Assignment 1

CS6510: Applied Machine Learning
IIT-Hyderabad
Aug-Nov 2017

Max Marks: 35 (+ 4 bonus)
Due: 11 Sep 2017 11:59 pm

This homework is intended to cover programming exercises in the following topics:

- k-NN, Decision Trees, Naive Bayes Classifier, Support Vector Machines

Instructions

- Please use Google Classroom to upload your submission by the deadline mentioned above. Your submission should comprise of a single file (PDF/ZIP), named `<Your_Roll_No>_PA1`, with all your solutions.
- For late submissions, 10% is deducted for each day (including weekend) late after an assignment is due. Note that each student begins the course with 6 grace days for late submission of assignments. Late submissions will automatically use your grace days balance, if you have any left. You can see your balance on the CS6510 Marks and Grace Days document under the course Google drive (soon to be shared).
- You should use PYTHON for the programming assignments.
- Please read the department plagiarism policy. Do not engage in any form of cheating - strict penalties will be imposed for both givers and takers. Please talk to instructor or TA if you have concerns.

1 Questions

1. **K-Nearest Neighbours:** *(2+4+2 = 8 marks)*

   - Build your own dataset (with unique data-points) using Python libraries. The dataset must consist of at least 1000 data-points in the form (ID,A,B,C,D,E,F,Class) where

   $$0 \leq A, B, C, D, E, F \leq 100 \tag{1}$$

   and ID is the sequentially generated number of that data-point. The value of Class is 0 or 1 (randomly assigned). Create this dataset just once since you will be using it for the next steps.

- Create a random 80-20 split of the data into train and test respectively. Store them in two different csv files. Plot the train data and test data with different labels on the same plot. (Note : You will need to submit the csv files. Adhere to the format mentioned above)
- Write your own implementation of K-Nearest Neighbours. Input to the program is the value K. Run this implemention for different values of K (1 to 21 inclusive). Do the same using the library (sklearn) implementation of K-NN.
- Compare and plot the accuracy and runtime for different values of K between your implementation and sklearn. What do you observe? Mention in the report. Mention any other inferences you can make.
- If the input to the K-NN is an even number, what do you expect, and what do you observe? Mention it in the report.
- Deliverables:
  (a) Three csv files with the dataset, the train dataset and the test dataset in the given format. *(2 marks)*
  (b) Well-documented/verbatose codes for generating the dataset(with split), your own implementation and the implementation with the library. *(4 marks)*
  (c) Report with the required statistics, plots and inferences. *(2 marks)*
- Allowed Libraries: sklearn, numpy, pandas, matplotlib.

2. **Naive Bayes Classifier:** *(4+4=8 marks)*

- Download the Email dataset from here. This dataset is from Lingspam dataset. There are two types of emails, "spam" and "non-spam". You need to implement your own Naive Bayes classifier to categorize emails as "spam" or "non-spam". To convert the text data into vectors, you can use TfidfVectorizer.
- Train your Naive Bayes on data given in folder "nonspam-train" and "spam-train". Test it on emails given in folder "nonspam-test" and "spam-test". Report accuracy, F1-score, confusion matrix and area under ROC curve (AUC). Also, report your observations from the result.
- Use sklearn's inbuilt Naive Bayes classifier to train on the same dataset. Compare the performance on test dataset with your implementation.
- *Deliverables:*
  – Source file(s) containing your implementation of the Naive Bayes Classifier. *(4 marks)*
  – Report containing the accuracy, F-1 score, confusion matrix and AUC on the test set, along with the observations asked above. *(4 marks)*

3. **Decision Tree:** *(6+4=10 marks)*
In this problem, you will be implementing a Decision Tree classifier. You can download the training dataset from this link. You should implement your classifier as a class using the template below:

```
class DecisionTree:
  def predict(self, test_file):
    pass
```

You may want to add other methods for reading the data, training, etc. But you must necessarily implement `predict(self, test_file)` method which takes exactly one argument - a string that denotes the name of the file containing test samples, and returns the model's predictions as a list of class labels. The test file will have the exact same format as the training, except (surprisingly) for the last column corresponding to true labels that will be missing. (Please refer to the `README` file accompanying the training set for the exact format.) Please ensure that the order of predicted class labels in the list returned by the `predict` method corresponds exactly to that of the data samples in the test file. For example, if the test file contains 5 data points, x_1, x_2, x_3, x_4, x_5, in that order, and your predictions are y_1, y_2, y_3, y_4, y_5, respectively, then the same order must be maintained in the returned list. You should use the `pickle` module to save your trained model to a file. You may want to do something like this:

```
>>> model = DecisionTree()
>>> model.train('train.csv') #assuming you have a train method
>>> import pickle
>>> with open('your_roll.model', 'w') as f:
        pickle.dump(model, f)
```

Your model will be run against a test dataset which is (unfortunately) not provided to you! Grading will be based on the performance of your model on this test set, as well as your implementation. (All the models will be tested on the same dataset.)

A note on libraries. You are free to use `numpy`, and any other module from the standard library. Your models will be tested in an environment wherein only `numpy` and the standard library will be available. Please add whatever module you are using as an attribute of the model. For example, if you are using `numpy`, your constructor may look like,

```
def __init__(self):
   import numpy
   self.np = numpy
   ...
```

so that whenever you want to use some attribute of the `numpy` module, you can access it through `self.np`. In other words, you must not expect a module to be available in the global scope. Any such attempt will lead to a `NameError`.

Deliverables:

- Source file(s) containing your implementation of `DecisionTree` class (you can choose any splitting criterion of your choice that performs well). Please stick to Python 2.7 for this problem. *(6 marks)*
- A trained model. Please name the file containing your model exactly as ⟨`your_roll_no`⟩`.model`. *(4 marks)*

4. **Support Vector Machine (SVM):** *(6+3=9 marks)*
Consider a sample of N independent and identically distributed training instances $(\mathbf{x}_i, y_i)_{i=1}^N$ where $\mathbf{x}_i$ is a d-dimensional input vector and $y_i \in (-1, +1)$ is its class label. SVM finds the linear discriminant with the maximum margin in the feature space induced by the mapping function Φ. The discriminator function can be written as:

$$f(\mathbf{x}) = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle + b$$

or the dual form:

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b$$

(that you saw in class), here $k(\mathbf{x}_i, \mathbf{x})$ is *kernel function* and α is the vector of dual variables corresponding to each separation constraint.

- There are several kernel functions used in literature, such as linear kernel (k_{LIN}), the polynomial kernel (k_{POL}), and the Gaussian kernel (k_{GAU}):

$$K_{LIN}(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

$$K_{POL}(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + 1)^q, q \in \mathbb{N}$$

$$K_{GAU}(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{s^2}), s \in \mathbb{R}_{++}$$

  - Write your own function to implement each of (k_{LIN}),(k_{POL}), and (k_{GAU}). Please visit this sklearn page, and, this StackOverflow page, or, this page to get an idea of how to use custom kernels. Use 5-fold stratified cross-validation on this dataset, and report *the best* kernel function and *best choice* of parameters (i.e. q and s).

- In recent years, Multiple Kernel Learning (MKL) methods have been proposed, where we use multiple kernels instead of selecting one specific kernel function and its corresponding parameters. There are different learning methods for determining the kernel combination function:
  - *Fixed rules* are functions without any parameters (e.g., summation or multiplication of the kernels) and do not need any training. You should implement your MKL as a class like below:

```
class MultiKernelfixedrules(object):
    def __init__(self, kernels, X=None):
        pass
```

    Within this class, define a function that will take a **convex sum** of all three above mentioned kernels in order to make MKL. Please give more weight to the kernel that provided the best performance in the previous question. You may visit this github page implementation to write your own class. Report your observations; does this give better performance on the same 5-fold cross validation setup?
  - **(Bonus Question)** *Heuristic approaches* use a parameterized combination function and find the parameters of this function generally by looking at some measure obtained from each kernel function separately. Moguerza et al. (2004) and de Diego et al. (2010a) propose a matrix functional form of combining kernels:

$$k_\eta(\mathbf{x}_i, \mathbf{x}_j) = \sum_{m=1}^{P} \eta_m k_m(\mathbf{x}_i, \mathbf{x}_j)$$

where $\eta_m(.,.)$ assignes a weight according to $k_m(\mathbf{x}_i, \mathbf{x}_j)$. Cristianini et al. (2002) proposed a method to choose η_m using *kernel alignment*.

$$\eta_m = \frac{A(K_m, yy^T)}{\sum_{h=1}^{P} A(K_h, yy^T)} \forall m$$

where $A(K, yy^T)$ is:

$$A(K, yy^T) = \frac{\langle K, yy^T \rangle_{Frobenius}}{(\#samples)\sqrt[2]{\langle K, K \rangle_{Frobenius}}}$$

* You should implement your MKL as a class like below:

```
class MultiKernelheuristic(object):
    def __init__(self, kernels, X=None):
        pass
```

* Within `MultiKernelheuristic` class define $A(K, yy^T)$ and η_m.

Deliverables:

- Source file(s) containing your implementation of (k_{LIN}),(k_{POL}), (k_{GAU}) python functions. Report the *best value* of q and s after applying 5-fold cross-validation. Compare **(a)** Classification accuracy **(b)** Training time on the above mentioned dataset. *(6 marks)*
- Source file(s) containing your implementation of `MultiKernelfixedrule` class. Compare **(a)** Training time **(b)** Classification accuracy between `MultiKernelfixedrule` and the individual kernels on the above mentioned dataset. *(3 marks)*
- **(OPTIONAL - BONUS)** Source file(s) containing your implementation of `MultiKernelheurist` class. Compare **(a)** Training time **(b)** Classification accuracy between `MultiKernelfixedrule` and `MultiKernelheuristic` on the above mentioned dataset. *(4 marks)*