CMPSC 448: Machine Learning and AI
Homework 4

Instruction

This HW includes both theory and implementation problems. Please note,

- Your code must work with Python 3.7+
- You need to submit a report in PDF including all written deliverable and plots, and all implementation codes so one could regenerate your results.
- For non-linear classifier problem, you need to submit a Jupyter notebook for each algorithm and all complementary python codes so one could **reproduce** your results. Submit your code in Problem2*.py and replace * with the name of classifier.
- For clustering problem, you are **NOT** allowed to use the implementation from scikit-learn or import it and should submit your own implementation. Submit your code in Problem3.py.

Boosting

Problem 1. [20 points] Consider the AdaBoost algorithm we discussed in the class. AdaBoost is an example of ensemble classifiers where the weights in next round are decided based on the training error of the weak classifier learned on the current weighted training set. We wish to run the AdaBoost on the dataset provided in Table 1.

1. Assume we choose the following decision stump f_1 (a shallow tree with a single decision node), as the first predictor (i.e., when training instances are weighted uniformly):

   ```
   if(Color is Yellow):
       predict Edible = Yes
   else:
       predict Edible = No
   ```

   What would be the weight of f_1 in final ensemble classifier (i.e., α_1 in $f(\boldsymbol{x}) = \sum_{i=1}^{K} \alpha_i f_i(\boldsymbol{x})$)?

2. After computing f_1, we proceed to next round of AdaBoost. We begin by recomputing data weights depending on the error of f_1 and whether a point was (mis)classified by f_1. What is the weight of each instance in second boosting iteration, i.e., after the points have been re-weighted? Please note that the weights across the training set are to be uniformly initialized.

3. In AdaBoost, would you stop the iteration if the error rate of the current weak classifier on the weighted training data is 0?

Instance	Color	Size	Shape	Edible
D1	Yellow	Small	Round	Yes
D2	Yellow	Small	Round	No
D3	Green	Small	Irregular	Yes
D4	Green	Large	Irregular	No
D5	Yellow	Large	Round	Yes
D6	Yellow	Small	Round	Yes
D7	Yellow	Small	Round	Yes
D8	Yellow	Small	Round	Yes
D9	Green	Small	Round	No
D10	Yellow	Large	Round	No
D11	Yellow	Large	Round	Yes
D12	Yellow	Large	Round	No
D13	Yellow	Large	Round	No
D14	Yellow	Large	Round	No
D15	Yellow	Small	Irregular	Yes
D16	Yellow	Large	Irregular	Yes

Table 1: Mushroom data with 16 instances, three categorical features, and binary labels.

Experiment with non-linear classifiers

Problem 2. [40 points] For this problem, you will need to learn to use software libraries for the following non-linear classifier types:

- Boosted Decision Trees (i.e., boosting with decision trees as weak learner)
- Random Forests
- Support Vector Machines with Gaussian Kernel

All of these are available in scikit-learn, although you may also use other external libraries (e.g., XGBoost [1] for boosted decision trees and LibSVM for SVMs). You are welcome to implement learning algorithms for these classifiers yourself, but this is neither required nor recommended.

Use the non-linear classifiers from above for classification of Adult dataset. You can download the data from a9a in libSVM data repository. The a9a data set comes with two files: the training data file a9a with 32,561 samples each with 123 features, and a9a.t with 16,281 test samples. Note that a9a data is in LibSVM format. In this format, each line takes the form ⟨label⟩ ⟨feature-id⟩:⟨feature-value⟩ ⟨feature-id⟩:⟨feature-value⟩ ..... This format is especially suitable for sparse datasets. Note that scikit-learn includes utility functions (e.g., load_svmlight_file) for loading datasets in the LibSVM format.

For each of learning algorithms, you will need to set various hyperparameters (e.g., the type of kernel and regularization parameter for SVM; tree method, max depth, number of weak classifiers, etc for XG-Boost; number of estimators and min impurity decrease for Random Forests). Often there are defaults that make a good starting point, but you may need to adjust at least some of them to get good performance. Use hold-out validation or K-fold cross-validation to do this (scikit-learn has nice features to accomplish this, e.g., you may use train_test_split to split data into train and test data and sklearn.model_selection for K-fold cross validation). Do **not** make any hyperparameter choices (or any other similar choices) based

[1] A simple blog post on how to use XGBoost please check this.

on the test set! You should only compute the test error rates after you have settled on hyperparameter settings and trained your three final classifiers.

What to submit (in PDF file and Jupyter/python codes):

1. A brief description of each algorithm and how it works.
2. Description of your training methodology, with enough details so that another machine learning enthusiast can reproduce the your results. You need to submit all the codes (python and Jupyter notebooks) to reproduce your code. Please use prefix `Problem2*.py` where you need to replace `*` with the name of non-linear classifier for your coding files.
3. The list of hyperparameters and brief description of each hyperparameter you tuned in training, their default values, and the final hyperparameter settings you use to get the best result.
4. Training error rates, hold-out or cross-validation error rates, and test error rates for your final classifiers. You are also encouraged to report other settings you tried with the accuracy it achieved (please make a table with a column with each hyperparamter and accuracy of configuration of parameters).
5. Please do your best to obtain the best achievable accuracy for each classifier on given dataset. **Note: The amount of effort you put on tuning the parameters will be determined based on the discrepancy between the accuracy you get and the best achievable accuracy on a9a data for each algorithm.**

Parameters to be tuned for XGBoost:

1. n_estimators
2. max_depth
3. lambda
4. learning_rate
5. missing
6. objective

Parameters to be tuned for SVM:

1. kernel_type
2. gamma
3. C

Parameters to be tuned for Random Forests:

1. n_estimators
2. bootstrap
3. max_depth
4. min_impurity_decrease
5. min_samples_leaf

Clustering

Problem 3. [40 points] For this problem, you will implement the k-means++ algorithm in Python. You will then use it to cluster Iris dataset from the UCI Machine Learning Repository. The data is contained in the iris.data file, while the iris.names file contains a description of the data. The features $\boldsymbol{x}$ are given as the first four comma-separated values in each row in the data file. The labels y are the last entry in each row, but you do NOT need the class label for clustering.

1. sepal length (cm)
2. sepal width (cm)
3. petal length (cm)
4. petal width (cm)
5. class: {Iris Setosa, Iris Versicolour, Iris Virginica}

You need to,

- Create a new data set with two features by computing the ratio of raw features: $\boldsymbol{x} = (x_1, x_2)$ where $x_1 =$ (sepal length/sepal width) and $x_2 =$ (petal length/petal width). Plot the data to observe the clusters in data by yourself (use class label to color the data points for better illustration of clusters).
- Implement the k-means++ algorithm. You are provided with the skeleton of the code with main functions to be implemented (Problem3.py file in assignment directory). Submit the source code (documented!) of your implementation.
- Cluster the modified Iris dataset with with two features explained above. Run your algorithm 50 times over the data with different values of clusters $k = 1, 2, 3, 4, 5$ and plot the accuracy (x and y axes should be the number of clusters and the clustering objective, respectively).
- Based on the above plot, decide the number of final clusters and justify your answer. For the chosen number of clusters,
  1. Create a plot showing how objective changes with number of iterations.
  2. Create a plot with the data colored by assignment, and the cluster centers.