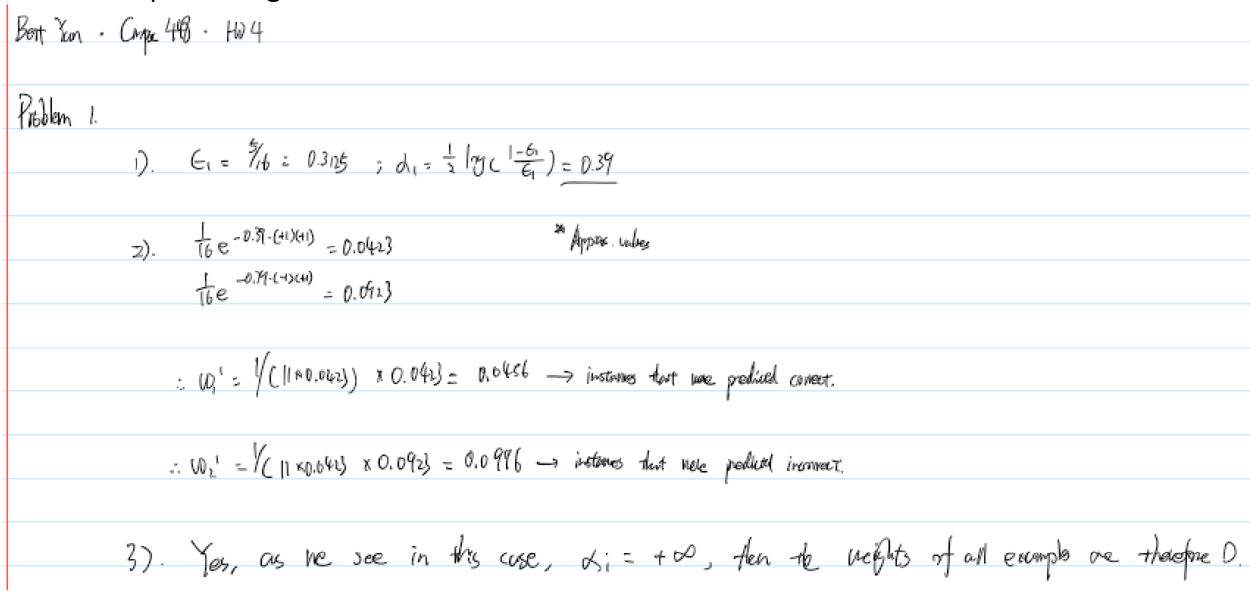
CMPSC 448

Assignment 4 | Due: 11/11/22

Bert Yan

Original work done by me.

Problem 1 | Boosting



Bert Yan · Cmpsc 448 · HW4

Problem 1.

1). $\epsilon_1 = 5/16 = 0.3125$; $\alpha_1 = \frac{1}{2} \log (\frac{1-\epsilon_1}{\epsilon_1}) = \underline{0.39}$

2). $\frac{1}{16} e^{-0.39 \cdot (+1)(+1)} = 0.0423$ * Approx. values

$\frac{1}{16} e^{-0.39 \cdot (-1)(+1)} = 0.0923$

$\therefore w_1' = 1/(11 \times 0.0423) \times 0.0423 = 0.0456 \rightarrow$ instances that were predicted correct.

$\therefore w_2' = 1/(11 \times 0.0423 \times 0.0923 = 0.0996 \rightarrow$ instances that were predicted incorrect.

3). Yes, as we see in this case, $\alpha_i = +\infty$, then the weights of all example are therefore 0.

Problem 2 | Experiment with non-linear classifiers

1 | Boosted Decision Trees

**Please refer to Problem2BoostedDecisionTrees.ipynb for coding.*

Boosted Decision Trees | Algorithm description

Boosted decision trees use an efficient implementation of the MART gradient boosting algorithm. Gradient boosting is a machine learning technique for regression problems. It builds each regression tree in a stepwise fashion, using a predefined loss function to measure the error in each step and correct for it in the next. Gradient boosting works by building simpler (weak) prediction models sequentially where each model tries to predict the error left over by the previous model. Because of this, the algorithm tends to overfit rather quickly.

Boosted Decision Trees | Training methodology

Load the prepared data -> load_svmlight_file('a9a.txt')
Train the XGBoost Model -> split the training data and use the K-fold CV
Tune the hyperparameters -> use grid search
Decide the final classifiers
Make Predictions with XGBoost Model -> model.score(X_test, y_test)

Boosted Decision Trees | List of hyperparameter

eta, gamma, max_depth, min_child_weight, max_delta_step, subsample, sampling_method, colsample_bytree, colsample_bylevel, colsample_bynode, lambda, alpha, tree_method, sketch_eps, scale_pos_weight, updater, refresh_leaf, process_type, grow_policy, max_leaves, max_bin, predictor, num_parallel_tree, monotone_constraints, interaction_constraints

Boosted Decision Trees | Hyperparameter tuned

tree method [default = auto]: For training boosted tree models
max_depth [default = 6]: Maximum depth of a tree.
min_child_weight [default = 1]: Minimum sum of instance weight (hessian) needed in a child.
eta [default = .3]: Step size shrinkage used in update to prevent overfitting.
colsample_bytree [default = 1]: is the subsample ratio of columns when constructing each tree.

Boosted Decision Trees | Final hyperparameter settings

tree method [default = auto]
max_depth = 7
min_child_weight [default = 1]
eta [default = .3]
colsample_bytree = 5

Training error rates: 0.1465
Cross-validation error rates: 0.162694513
test error rates: 0.1444

2 | Random Forests

**Please refer to Problem2RandomForests.ipynb for coding.*

Random Forests | Algorithm description

Random Forest: As in bagging, we build several decision trees on bootstrapped training samples. But when building these decision trees, each time a split in a tree is considered, a random sample of p < d features (predictors) is chosen as split candidates from the full set of all d features. The split is allowed to use only one of those p features. A fresh sample of features is taken at each split, and typically we choose $p < \sqrt{d}$

Random Forests | Training methodology

Load the prepared data -> load_svmlight_file('a9a.txt')
Train the RandomForest Model -> split the training data and use the K-fold CV clf.fit(X_train, y_train)
Tune the hyperparameters -> use grid search
Decide the final classifiers
Make Predictions with RandomForest Model -> clfl.score(X_test, y_test)

Random Forests | List of hyperparameter

{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': None, 'verbose': 0, 'warm_start': False}

Random Forests | Hyperparameter tuned

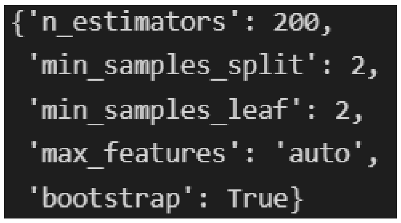
n_estimators [default = 100]: number of trees in the foreset
max_features [default = auto] = max number of features considered for splitting a node
max_depth [default = None] = max number of levels in each decision tree
min_samples_split [default = 2] = min number of data points placed in a node before the node is split min_samples_leaf [default = 1] = min number of data points allowed in a leaf node
bootstrap [default = True] = method for sampling data points (with or without replacement)

Random Forests | Final hyperparameter settings

{'n_estimators': 200,
'min_samples_split': 2,
'min_samples_leaf': 2,
'max_features': 'auto',
'bootstrap': False}*

Training error rates: 0.1547
Cross-validation error rates: 0.156592957
test error rates: 0.1556

*Note here that bootstrap is set to False, however running model_random.best_params_ outputted that:



```
{'n_estimators': 200,
 'min_samples_split': 2,
 'min_samples_leaf': 2,
 'max_features': 'auto',
 'bootstrap': True}
```

Checking setting bootstrap to True while other settings remain the same:

Training error rates: 0.1531
Cross-validation error rates: 0.1566
test error rates: 0.1526

We observe differences in the matter of one hundredth, one thousandth for cross-validation error rates.

3 | Support Vector Machines

**Please refer to Problem2SVM.ipynb for coding.*

Support Vector Machines | Algorithm description

Support Vector Machines with Gaussian Kernel: In SVM, kernels are used for solving nonlinear problems in higher dimensional where linear separation is not possible. Generally, SVM is a simple dot product operation. Therefore, we should choose a kernel such that, K(x,y)=K(x).K(y). Gaussian is one such kernel giving good linear separation in higher dimension for many nonlinear problems.

Support Vector Machines | Training methodology

Load the prepared data -> load_svmlight_file('a9a.txt')
Train the SVM Gaussian Model -> split the training data and use the K-fold CV clf.fit(X_train, y_train)
Tune the hyperparameters -> use grid search
Decide the final classifiers
Make Predictions with SVM Gaussian Model -> clfl.score(X_test, y_test)

Support Vector Machines | List of hyperparameter

{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'tol': 0.001, 'verbose': False}

Support Vector Machines | Hyperparameter tuned

Kernel[rbf]: The function of kernel is to take data as input and transform it into the required form.

Support Vector Machines | Final hyperparameter settings

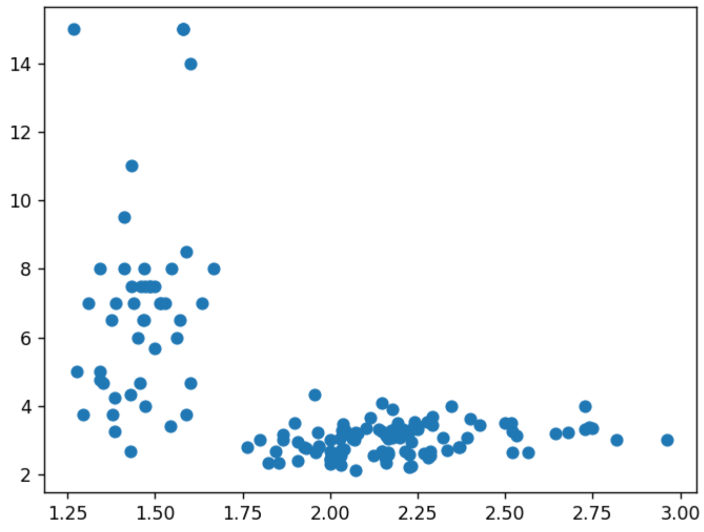
kernel = linear

Training error rates: 0.
 Cross-validation error rates: 0.1530713
test error rates: 0.1502

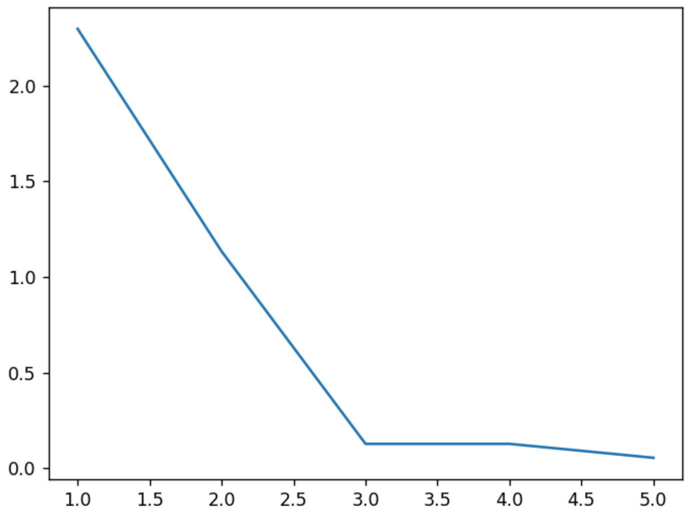
Problem 3 | Clustering

Justification are concluded within the graphs as they indicate the selection of k=5.

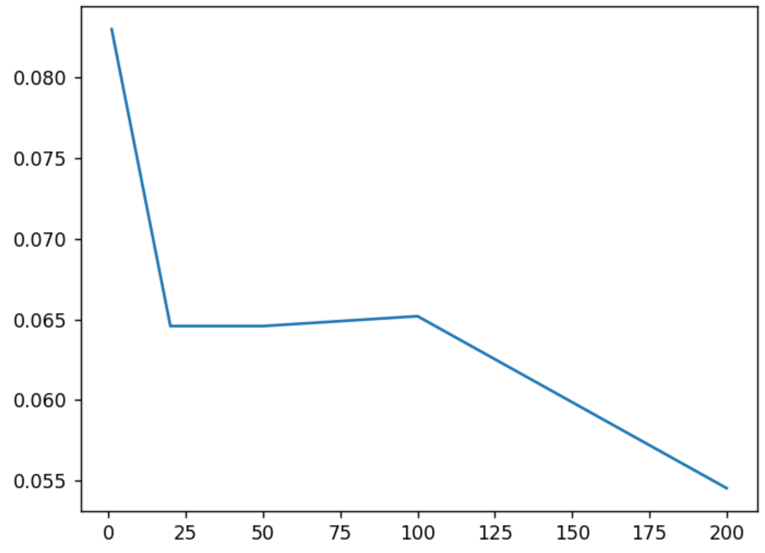
Scatter Plot for x1 and x2:



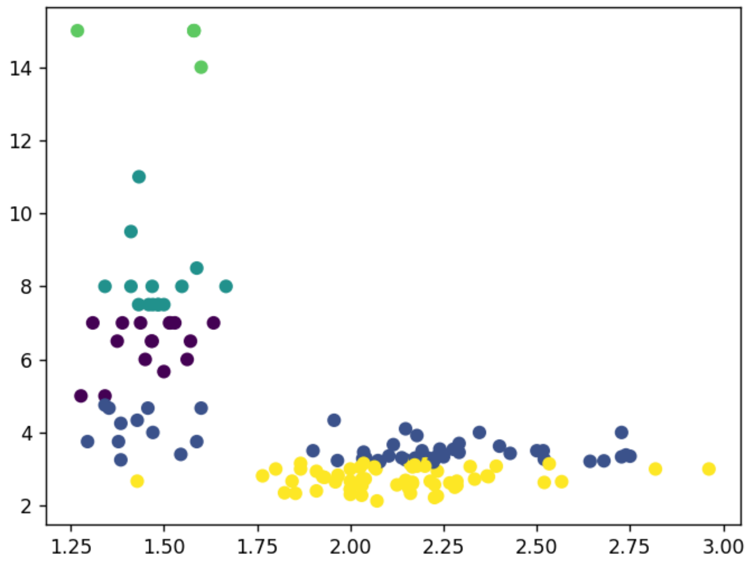
Plot for running k-means with k = 1,2,3,4,5:

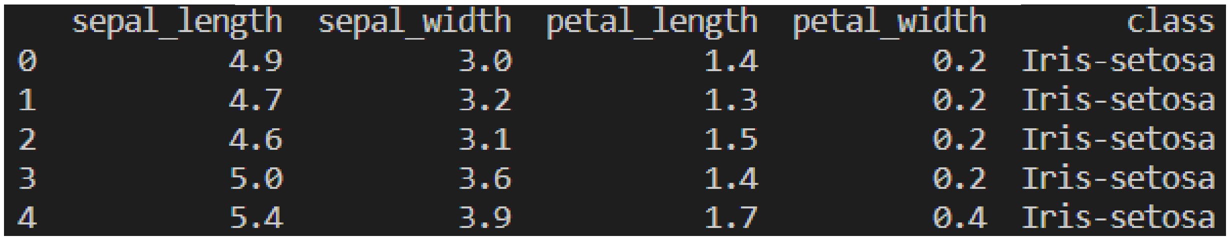


Plot for the corresponding results for using k=5 as the best choice for number of iterations:



Plot with the data colored by assignment, and the cluster centers:





```
   sepal_length  sepal_width  petal_length  petal_width        class
0           4.9          3.0           1.4          0.2  Iris-setosa
1           4.7          3.2           1.3          0.2  Iris-setosa
2           4.6          3.1           1.5          0.2  Iris-setosa
3           5.0          3.6           1.4          0.2  Iris-setosa
4           5.4          3.9           1.7          0.4  Iris-setosa
```