**Required packages for the program**

These Python (version 3.8.5) packages are used for the program:

pandas

os (default package in Python)

sys (default package in Python)

numpy

matplotlib

scikit-learn (imported as sklearn)

datetime (default package in Python)

To install non-default packages with **pip**, these commands should be executed:

pip install pandas

pip install numpy

pip install matplotlib

pip install scikit-learn

**or**

pip3 install -U numpy

pip3 install -U pandas

pip3 install -U matplotlib

pip3 install -U scikit-learn

To install non-default packages with **conda**, these commands should be executed:

conda install -c conda-forge numpy

conda install -c conda-forge pandas

conda install -c conda-forge matplotlib

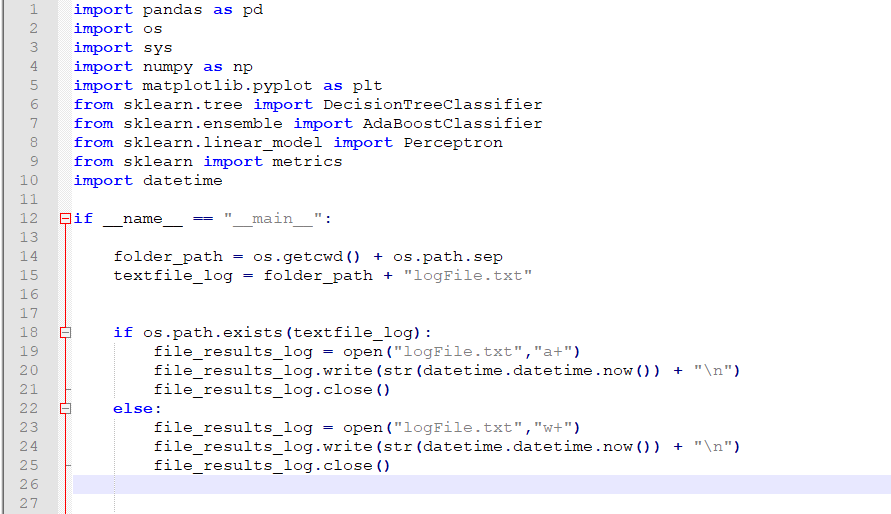
conda install -c conda-forge scikit-learn

**Training and testing files for the program**

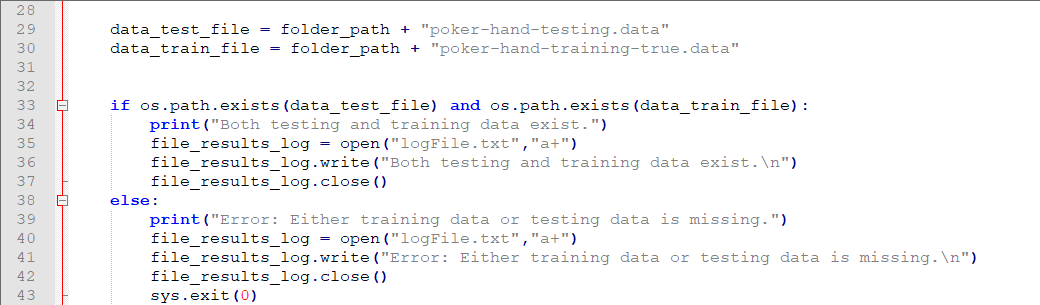
To be able to use, the training file (*poker-hand-training-true.data*) and testing file (*poker-hand-testing.data*) must be in the same folder with the program. Both files are downloaded from <https://archive.ics.uci.edu/ml/machine-learning-databases/poker/>

**How program works**

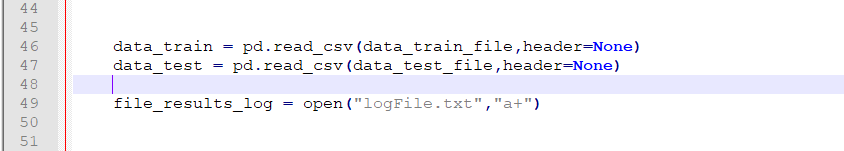
1) The program creates the log file (if doesn’t exist), and writes the date and time.



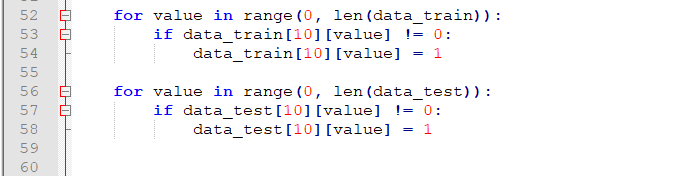
2) The program checks the existence of training and test files. If one of them is absent, the program finishes.



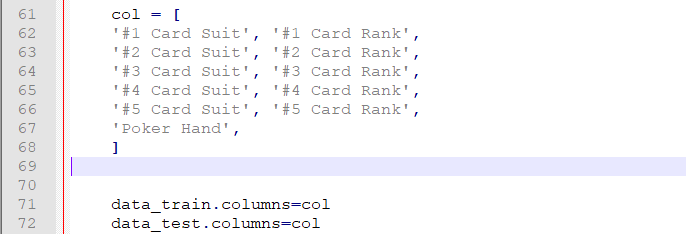
3) The program reads the .data files.



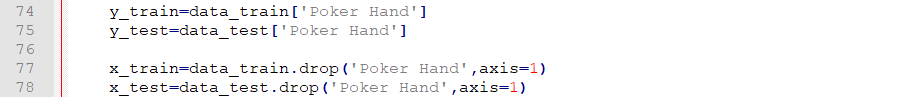
4) Due to the fact that in the homework we need only 2 values (0 – None in the hand and 1 – Something in the hand.), the program turns all non-zero values in the last column to 1. Since there are million values in the training data, this part is the most time consuming.



5) The program assigns the names to the variables.

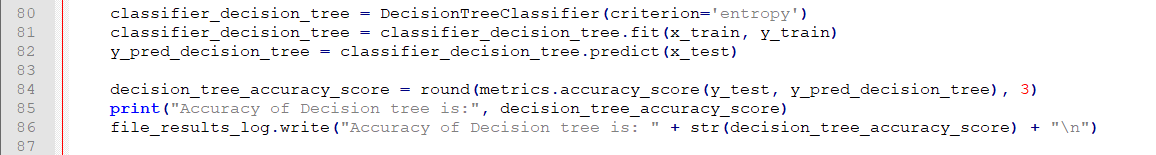


6) The program creates the variables with testing and training values (*y\_train* and *y\_test* , respectively) and the variables to be trained and tested (*x\_train* and *x\_test* , respectively).



7) In the first stage, the classification is done by Decision Tree Classifier. Decision Tree Classification is done by DecisionTreeClassifier() got *from sklearn.tree* package. More information can be read here: <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

Only changed default parameter for the homework purpose is **criterion='entropy'** , because by the default, DecisionTreeClassifier() uses **criterion='gini'** . After creating the variable, it builds the decision tree from the training set (*x\_train* and *y\_train* , respectively) by DecisionTreeClassifier.fit( *x\_train* , *y\_train* ) method in the next line. Then, it tries to predict the values from the test set by DecisionTreeClassifier.predict( *x\_test* ) method. The accuracy is rounded till 3 digits after zero by metrics.accuracy\_score(*y\_test, y\_pred\_decision\_tree*) from *sklearn* package (More information can be read here: [https://scikit-learn.org/stable/modules/classes.html?highlight=metrics#module-sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html?highlight=metrics" \l "module-sklearn.metrics)) and is printed on the screen.



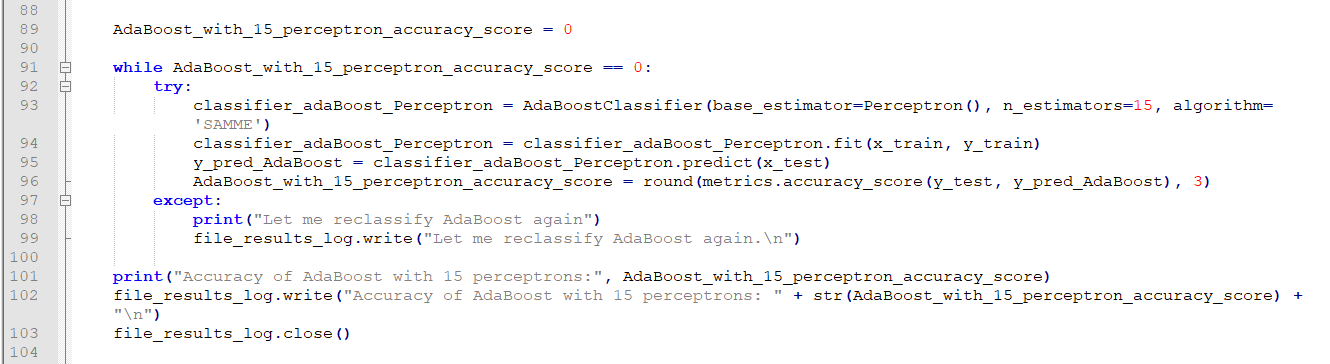
7) In the second stage, the classification is done by AdaBoost Classifier. AdaBoost Classification is done by AdaBoostClassifier() got *from sklearn.ensemble* package. More information can be read here: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html>

The AdaBoost Classifier is loaded as **AdaBoostClassifier(base\_estimator=Perceptron(), n\_estimators=15, algorithm='SAMME')**. The **base\_estimator** by default is DecisionTreeClassifier with max\_depth=1, so according to our requirements, we have to define the weak classifier as **Perceptron()**. The Perceptron() is loaded from *sklearn.linear\_model* package with default values (More information can be read here: <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html>). The **n\_estimators** is the number of estimators by which AdaBoost is classified. Since we need 15 perceptrons according to our requirements, **n\_estimators** made equal to 15. **algorithm='SAMME'** is used as discrete boosting algorithm, since our weak classifier doesn’t support probability values, which doesn’t allow us to use default algorithm **'SAMME.R'** .

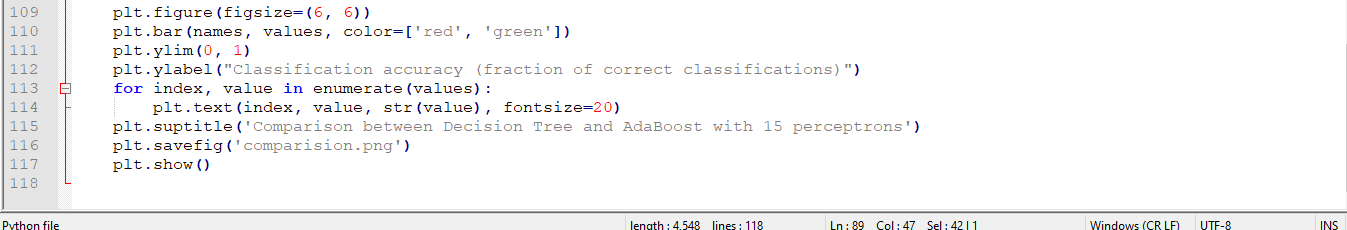
Other problem is that if the values of perceptron is worse than the random values,

it provides an error **ValueError: BaseClassifier in AdaBoostClassifier ensemble is worse than random, ensemble can not be fit.** To avoid it, AdaBoost\_with\_15\_perceptron\_accuracy\_score = 0 variable, while loop with try and except block are created. The loop executes until the accuracy score remains 0.

After creating the variable, it builds the AdaBoost classifier from the training set (*x\_train* and *y\_train* , respectively) by AdaBoostClassifier.fit( *x\_train* , *y\_train* ) method in the next line. Then, it tries to predict the values from the test set by AdaBoostClassifier.predict( *x\_test* ) method. The accuracy is rounded till 3 digits after zero by metrics.accuracy\_score(*y\_test, y\_pred\_decision\_tree*) from *sklearn* package (More information can be read here: [https://scikit-learn.org/stable/modules/classes.html?highlight=metrics#module-sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html?highlight=metrics" \l "module-sklearn.metrics)) and is printed on the screen.



8) In the final stage, the image is created, labeled, saved (as “*comparision.png”* file) and opened by the program. This done by **matplotlib.pyplot** package. The program finishes only if the image window is closed:



For example, the resulting image below describes that the classification accuracy (in the graph, the fraction of correct classifications. To get percents, simply multiply every value to 100) of Decision Tree Classifier is significantly higher than the AdaBoost Classifier with 15 perceptrons. Despite the accuracy of AdaBoost fluctuates from execution to execution, the difference in percents usually between 6-17%. In other words, Decision Tree Classifier is 6-17% more accurate.

