COGS 118A - Final Project Report

March 18, 2021

1 Abstract

This paper is my attempt of describing a mini-scale replication of Caruana and Niculescu-Mizil's 2006 paper (hereafter referred to as CNM06) by comparing the performance of three supervised machine-learning algorithms (Logistic Regression, Random Forersts (RFs), and KNN) on binary classification problems across four datasets (LETTER1, LETTER2, COVTYPE, CONNECT4) by using three scoring metrics (accuracy, F1 score and ROC). The emperical results of this analysis seem to corroborate the findings of the original CNM06 paper.

Keywords: CNM06 - Caruana and Niculescu-Mizil's 2006 paper, Binary Classification, k-Nearest Neighbours, Supervised Models, Logistic Regression, Perceptron, COV - Cover Type Data, LETTER1 - Letter Data "O" positive classification, LETTER2 - Letter Data "AM" positive classification, CONNECT4 - Connect-4 Outcome Type dataset

2 Introduction

STATLOG (King et al., 1995) was the most well-known and comprehensive study of emperically comparing supervised machine learning algorithms when it was performed. Caruana and Niculescu-Mizil conducted the CNM06 study to carry out an extensive emperical evaluation of new learning algorithms that have subsequently emerged since the STATLOG study (e.g. KNN, SVM, random forests, decision trees, ANN, NB and LOGREG).

This project focuses on replicating CNM06 using three supervised machine learning algorithms (Logistic Regression (LOGIT), Random Forests (RFs), and KNN) constrained by four of the original data sets (COVTYPE, CONNECT4, LETTER1, LETTER2). Since two of these datasets are imbalanced (CONNECT4, LETTER1), accuracy as the sole metric is not optimal to observe all the differences between these classifiers. Therefore, compensating for this imbalance, to compare model performance in my paper I use other metrics in addition to accuracy which include the F1 score and ROC.

My results seem to replicate the findings of CNM06. Random Forests is the best overall classifier by a significant difference.

3 Methods

The selection of the hyperparameters of the four algorithms I have decided to analyze in this report follow those chosen in the CNM06 while the performance metrics are slightly different.

3.1 Learning Algorithms

Logistic Regression (LOGIT): Initialized using default solver set to Limited Memory BFGS (*lbfgs*). To encourage convergence, max iterations is set to 1000. We train the model by varying the ridge (regularization) parameter by factors of 10 ranging from 10^{-8} and 10^{5}.

Random Forests (RFs): All my Random Forest classifiers use 1024 total tree estimators with the following space of max number of features per tree split: {1, 2, 4, 6, 8, 12, 16, or 20}.

KNN: I use 25 different values of k between 1 and 500 equally spaced by a step of 20. During multi-metric evaluation using grid-search, weights searched were {*uniform, distance*} and Euclidean distance is used as a metric.

3.2 Performance Metrics

I used accuracy, F1 score and ROC as the performance metrics for this anaylsis.

Accuracy: Used as the base measure of different algorithm performances across different datasets with values ranging between [0,1]. However, accuracy as the sole metric is extremely poor at capturing true algorithm performance on imbalanced datasets. Poor classifiers can produce high accuracies on extremely imbalanced datasets by simply predicting the over-represented classification. Therefore, to compensate for this flaw, we compare the algorithm performances using other metrics as well.

F1 score: F-1 score incorporates the ideas of Precision and Recall.

ROC: ROC measures the efficiency with which classifiers order positive cases before negative cases, and operates irrespective of class distribution (Caruana,2006).

All of these metrics will allow us to fully capture the true performance of an algorithm on a dataset even if accuracy is falsely high (in case of imbalanced datasets)

3.3 Data Sets

I trained my algorithms and tested them on four datasets from the UCI Machine Learning Repository: COVTYPE, CONNECT4, LETTER1, LETTER2. For more information on the datasets, please refer to table 1. Before training/testing each model, I applied standardization to a mean of 0 and unit variance to the features of each data set. Additionally, I converted the categorical features using one-hot encoding for each data set.

For the COVTYPE dataset, by assigning the largest class (Lodgepole Pine) to 1 and the remaining classes to 0, I converted the target variable to binary.

For the LETTER1 dataset, I converted the target variable to binary by mapping "O" to 1 and the remaining letters were mapped to 0.

For the LETTER2 dataset, I converted the target variable to binary by mapping "A-M" to 1, inherintly mapping the remaining letters to 0.

For the CONNECT4 dataset, the target variable was converted to binary classification through assigning the mapping (> '1.5')–>1 and (<= '1.5')–>0.

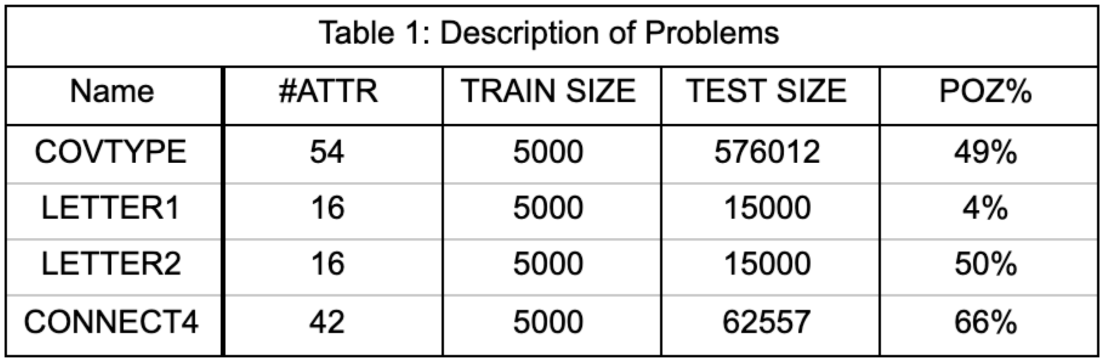

Table 1: Description of Problems

Name	#ATTR	TRAIN SIZE	TEST SIZE	POZ%
COVTYPE	54	5000	576012	49%
LETTER1	16	5000	15000	4%
LETTER2	16	5000	15000	50%
CONNECT4	42	5000	62557	66%

4 Training and Testing Framework

For each machine learning algorithm and dataset combination, randomly sampled data is split into a training set (train_size = 5000) and a testing set (test_size = #samples - train_size) in each iteration of five trials.

For each trial's train/test split, the optimal hyperparameters per performance metric for each algorithm are found via 5-fold-cv multi-metric gridsearch.

For each metric, we obtain a different set of optimal hyperparameters for a single algorithm. Therefore, since we are using three metrics, we obtain three different sets of hyperparameters for each algorithm in a single trial.

Within a trial, the model is trained using these optimal hyperparameters by fitting the entire training set and then used to make predictions on the test set. This happens three times using each set of optimal hyperparameters.

Each time, to evaluate performance of the model on the test set, we only record the performance metric which is responsible for choosing the set of hyperparameters being used.

To see this process in action, please refer to the CODE in appendix.

5 Performance

Each performance metric of each classifier averaged over all four datasets is summarized in table 2 below. The rows list the classifiers and the columns list their averaged metric performance over all four data sets. The best performing algorithm in each column is boldfaced. An * is supposed to denote a non-significant difference between a classifier and the best performing algorithm. However, in my analysis, it's clear that all differernces are significant.

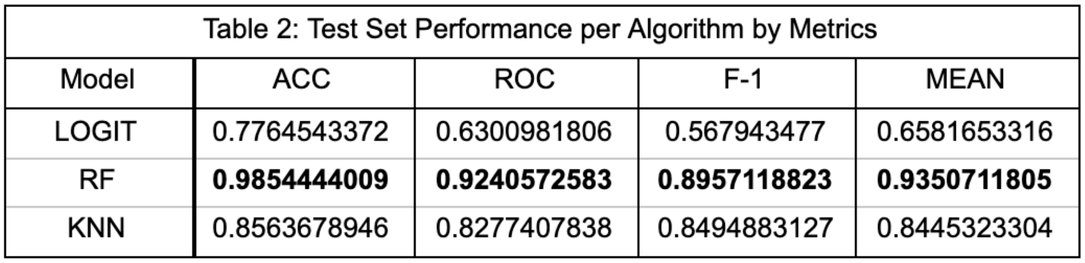

Table 2: Test Set Performance per Algorithm by Metrics

Model	ACC	ROC	F-1	MEAN
LOGIT	0.7764543372	0.6300981806	0.567943477	0.6581653316
RF	**0.9854444009**	**0.9240572583**	**0.8957118823**	**0.9350711805**
KNN	0.8563678946	0.8277407838	0.8494883127	0.8445323304

The performance of each classifier averaged over all metrics on each dataset is summarized in table 3 below. The rows list the classifiers and the columns list their performance averaged over all the metrics for each dataset. The best performing algorithm on specific dataset in a column is boldfaced. An * denotes a non-significant difference between a classifier and the best performing algorithm.

Table 3: Test Set Performance per Algorithm by Dataset

Model	COVTYPE	LETTER1	LETTER2	CONNECT4	MEAN
LOGIT	0.7536584605	0.4873431397	0.7237598674	0.6678998587	0.6581653316
RF	**0.8198716014**	**1.178077091**	0.9481177691*	**0.7942182607**	**0.9350711806**
KNN	0.7756400106*	0.9335403975	**0.9567020188**	0.7122468944	0.8445323303

6 Discussion

Table 2 tells us that across all datasets, the classifier with the highest performance metrics is Random Forests (RFs) by a significant difference. Inherently, Random Forests is the classifier with the highest mean performance metric score.

Table 3 tells us that the best classifier on COVTYPE, LETTER1 and CONNECT4 data sets is Random Forests (RFs) by a significant difference (except for KNN scoring non-significantly lower than RFs COVTYPE dataset). However, on the LETTER2 dataset, KNN has the highest test performance with RF following close behind with a non-significant difference. Overall, the best classifier on all the datasets is RF by a high margin.

Table 4 (in appendix) tells us that Random Forests and KNN perform perfect classification on the validation (training) dataset. Logistic Regression performs slightly better on the validation set than on the testing set.

With regards to time complexity of each classifier, increase in sample size of the datasets resulted in a longer prediction time for each classifier. KNN took slightly longer than RF which took significantly longer than Logistic Regression when comparing prediction times.

7 Conclusion

Overall, my analysis more or less corroborates the findings from the study conducted by Caruana and Niculescu-Mizil in their 2006 paper (CNM06) paper. Random Forests performs consistently well over all the datasets scoring the highest metrics, therefore proving to be the most optimal classifier overall. While KNN performs the best on LETTER2 dataset, RF is non-significantly lower which allows the classifier to maintain a significantly overall higher position when it comes to classifier ranking.

8 Acknowledgments

I would like to thank the staff of COGS 118A for providing support throughout the process of this analysis. I would also like to thank my roommates who provided me with the environment to complete this assignment over several late nights without disturbance. Finally, I would like to thank Professor Jason Fleischer for instilling within me a strong foundation of machine learning through this quarter, as well as extending support during times I felt overwhelmed or stressed. As always, I thank all my classmates who asked questions, and replied to my posts on piazza which allowed me to move past obstacles through this project.

9 Appendix

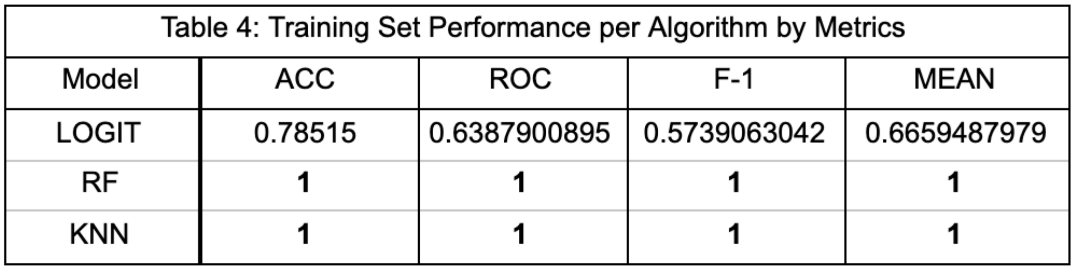

Table 4: Training Set Performance per Algorithm by Metrics

Model	ACC	ROC	F-1	MEAN
LOGIT	0.78515	0.6387900895	0.5739063042	0.6659487979
RF	**1**	**1**	**1**	**1**
KNN	**1**	**1**	**1**	**1**

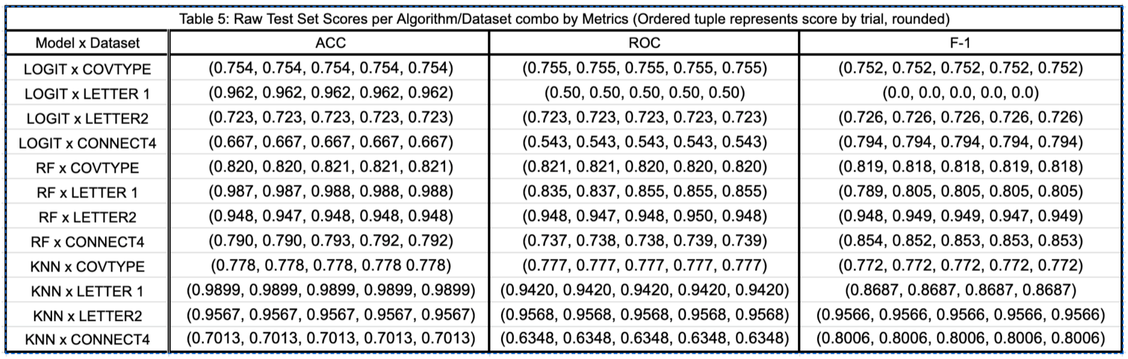

Table 5: Raw Test Set Scores per Algorithm/Dataset combo by Metrics (Ordered tuple represents score by trial, rounded)

Model x Dataset	ACC	ROC	F-1
LOGIT x COVTYPE	(0.754, 0.754, 0.754, 0.754, 0.754)	(0.755, 0.755, 0.755, 0.755, 0.755)	(0.752, 0.752, 0.752, 0.752, 0.752)
LOGIT x LETTER 1	(0.962, 0.962, 0.962, 0.962, 0.962)	(0.50, 0.50, 0.50, 0.50, 0.50)	(0.0, 0.0, 0.0, 0.0, 0.0)
LOGIT x LETTER2	(0.723, 0.723, 0.723, 0.723, 0.723)	(0.723, 0.723, 0.723, 0.723, 0.723)	(0.726, 0.726, 0.726, 0.726, 0.726)
LOGIT x CONNECT4	(0.667, 0.667, 0.667, 0.667, 0.667)	(0.543, 0.543, 0.543, 0.543, 0.543)	(0.794, 0.794, 0.794, 0.794, 0.794)
RF x COVTYPE	(0.820, 0.820, 0.821, 0.821, 0.821)	(0.821, 0.821, 0.820, 0.820, 0.820)	(0.819, 0.818, 0.818, 0.819, 0.818)
RF x LETTER 1	(0.987, 0.987, 0.988, 0.988, 0.988)	(0.835, 0.837, 0.855, 0.855, 0.855)	(0.789, 0.805, 0.805, 0.805, 0.805)
RF x LETTER2	(0.948, 0.947, 0.948, 0.948, 0.948)	(0.948, 0.947, 0.948, 0.950, 0.948)	(0.948, 0.949, 0.949, 0.947, 0.949)
RF x CONNECT4	(0.790, 0.790, 0.793, 0.792, 0.792)	(0.737, 0.738, 0.738, 0.739, 0.739)	(0.854, 0.852, 0.853, 0.853, 0.853)
KNN x COVTYPE	(0.778, 0.778, 0.778, 0.778 0.778)	(0.777, 0.777, 0.777, 0.777, 0.777)	(0.772, 0.772, 0.772, 0.772, 0.772)
KNN x LETTER 1	(0.9899, 0.9899, 0.9899, 0.9899, 0.9899)	(0.9420, 0.9420, 0.9420, 0.9420, 0.9420)	(0.8687, 0.8687, 0.8687, 0.8687)
KNN x LETTER2	(0.9567, 0.9567, 0.9567, 0.9567, 0.9567)	(0.9568, 0.9568, 0.9568, 0.9568, 0.9568)	(0.9566, 0.9566, 0.9566, 0.9566, 0.9566)
KNN x CONNECT4	(0.7013, 0.7013, 0.7013, 0.7013, 0.7013)	(0.6348, 0.6348, 0.6348, 0.6348, 0.6348)	(0.8006, 0.8006, 0.8006, 0.8006, 0.8006)

10 References

1. Caruana, R. and Niculescu-Mizil, A.,2020. An Empirical ComparisonOf Supervised Learning Algorithms.[online] Cs.cornell.edu. Available at: https://www.cs.cornell.edu/~caruana/ctp/ct.papers/caruana.icml06.pdf

2. Archive.ics.uci.edu. 2020. UCI Ma-chine Learning Repository. [online] Available at: https://archive.ics.uci.edu/ml/index.php [Accessed 15 March 2021]

3. Scikit-learn.org.2020.Sklearn.Neighbors.Kneighborsclassifier— Scikit-Learn 0.23.2 Documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html [Accessed 15 March 2021]

4. Scikit-learn.org.2020.Sklearn.LinearModel.Logisticregression—Scikit-Learn 0.23.2 Documentation.[online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html [Accessed 15 March 2021]

5. Scikit-learn.org.2020.Sklearn.Ensemble.Randomforestclassifier—Scikit-Learn 0.23.2 Documentation.[online] Available at: https://scikit-

learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html [Accessed 15 March 2021]

COGA 118A - Final Project Code (COVTYPE DATASET)

March 18, 2021

1 Algorithm testing for COVTYPE dataset

```
[ ]: import numpy as np
     import matplotlib.pyplot as plt
     from sklearn import datasets
     from sklearn.linear_model import LogisticRegression
     from sklearn.model_selection import GridSearchCV
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import accuracy_score, roc_auc_score, make_scorer, f1_score
     from sklearn.preprocessing import StandardScaler
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.neighbors import KNeighborsClassifier
```

1.1 Loading the Covertype Dataset

```
[ ]: # Covertype dataset.
     cov = datasets.fetch_covtype()   #Load Covertype (COVTYPE) dataset.

     X = cov.data                     # The shape of X is (581012, 54) which means
                                          # there are 581012 data points, each data␣
      ↪point
                                          # has 54 features.

     # Here, we convert the target variable to binary by assigning the largest class␣
      ↪to 1 and the rest to 0
     # The largest class in this dataset is Lodgepole Pine (Pinus contorta) - Type 2
     # Y = 0 (or False): Spruce/Fir (original value 1) / Ponderosa Pine (Original␣
      ↪value 3) /
     #                    Cottonwood/Willow (original value 4) / Aspen (Original␣
      ↪value 5) /
     #                    Douglas-fir (Original value 6) / Krummholz (Original value␣
      ↪7)
     # Y = 1 (or True): Lodgepole Pine (original value 2)

     # Thus, we use (cov.target > 1.5 and cov.target < 2.5) to convert the target␣
      ↪variable to binary
```

```
# This line of code will assign:
#       Y[i] = True  (which is equivalent to 1) if cov.target[k] > 1.5 and cov.
↪target[k] < 2.5 (Lodgpole Pine)
#       Y[i] = False (which is equivalent to 0)  if cov.target[k] <= 1.5 and cov.
↪target[k] >= 2.5 (The rest)

Y = np.logical_and(cov.target > 1.5,cov.target < 2.5).reshape(-1,1).astype(np.
↪float)

# The shape of Y is (581012, 1), which means there are 581012 data points, each␣
↪data point has 1 target value.

Y[Y==0] = -1
```

1.2 Logistic Regression

```
[ ]: train_set_accuracy = []
     test_set_accuracy = []
     train_set_roc = []
     test_set_roc = []
     train_set_f1score = []
     test_set_f1score = []
```

```
[ ]: for trial in range(5):

         # Divide the data points into training set and test set.
         X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size =␣
     ↪5000, random_state = 42, shuffle = True, stratify = Y)







         #Standardizing the data
         scaler = StandardScaler().fit(X_train)
         X_train_sc = scaler.transform(X_train)
         X_test_sc = scaler.transform(X_test)

         C_list = [100000000, 10000000, 1000000, 100000, 10000, 1000, 100, 10, 1, 0.
     ↪1, 0.01, 0.001, 0.0001, 0.00001]
         scoring = {'ROC': make_scorer(roc_auc_score), 'Accuracy':␣
     ↪make_scorer(accuracy_score), 'F-1': make_scorer(f1_score) }
```

```
    #Finding the hyperparameter settings that give best ACCURACY on the␣
↪training set.
    acc_gs = GridSearchCV(LogisticRegression(penalty = 'l2', solver = 'lbfgs',␣
↪max_iter = 10000).fit(X_train_sc, y_train.ravel()),
                  param_grid= {'C': C_list},
                  scoring = scoring, refit='Accuracy', return_train_score=True,␣
↪cv = 5)

    acc_gs.fit(X_train_sc, y_train.ravel())

    opt1_model = acc_gs.best_estimator_


    #Training the Model using the the best parameter settings for accuracy
    model1 = opt1_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the accuracy of this model on the training set
    train_set_accuracy.append(accuracy_score(y_train.ravel(), model1.
↪predict(X_train_sc)))


    #Finding and storing the accuracy of this model on the test set
    test_set_accuracy.append(accuracy_score(y_test.ravel(), model1.
↪predict(X_test_sc)))





    #Finding the hyperparameter settings that give best ROC on the training set.
    roc_gs = GridSearchCV(LogisticRegression(penalty = 'l2', solver = 'lbfgs',␣
↪max_iter = 10000).fit(X_train_sc, y_train.ravel()),
                  param_grid= {'C': C_list},
                  scoring=scoring, refit='ROC', return_train_score=True, cv = 5)

    roc_gs.fit(X_train_sc, y_train.ravel())

    opt2_model = roc_gs.best_estimator_


    #Training the Model using the the best parameter settings for ROC
    model2 = opt2_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the ROC of this model on the training set
    train_set_roc.append(roc_auc_score(y_train.ravel(), model2.
↪predict(X_train_sc)))


    #Finding and storing the ROC of this model on the test set
    test_set_roc.append(roc_auc_score(y_test.ravel(), model2.
↪predict(X_test_sc)))







    #Finding the hyperparameter settings that give best F-1 score on the␣
↪training set.
    f1_gs = GridSearchCV(LogisticRegression(penalty = 'l2', solver = 'lbfgs',␣
↪max_iter = 10000).fit(X_train_sc, y_train.ravel()),
                  param_grid= {'C': C_list},
                  scoring=scoring, refit='F-1', return_train_score=True, cv = 5)

    f1_gs.fit(X_train_sc, y_train.ravel())

    opt3_model = f1_gs.best_estimator_


    #Training the Model using the the best parameter settings for f1-score
    model3 = opt3_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the F-1 score of this model on the training set
    train_set_f1score.append(f1_score(y_train.ravel(), model3.
↪predict(X_train_sc)))


    #Finding and storing the F-1 of this model on the test set
    test_set_f1score.append(f1_score(y_test.ravel(), model3.predict(X_test_sc)))
```

[ ]:
```
print(train_set_accuracy)
print(test_set_accuracy)
print(train_set_roc)
print(test_set_roc)
print(train_set_f1score)
print(test_set_f1score)
```

```
[ ]: #Calculating average performance metrics over 5 trials for Logistic Regression␣
     ↪x Covtype Dataset
     avg_test_set_accuracy = 0.0
     avg_test_set_roc = 0.0
     avg_test_set_f1score = 0.0

     #Calculating average of accuracy
     for a in test_set_accuracy:
         avg_test_set_accuracy = avg_test_set_accuracy + a

     avg_test_set_accuracy = avg_test_set_accuracy / 5


     #Calculating average of ROC
     for r in test_set_roc:
         avg_test_set_roc = avg_test_set_roc + r

     avg_test_set_roc = avg_test_set_roc / 5


     #Calculating average of F1-Score
     for r in test_set_f1score:
         avg_test_set_f1score = avg_test_set_f1score + r

     avg_test_set_f1score = avg_test_set_f1score / 5

     print(avg_test_set_accuracy)
     print(avg_test_set_roc)
     print(avg_test_set_f1score)
```

1.3 Random Forests

```
[ ]: train_set_accuracy = []
     test_set_accuracy = []
     train_set_roc = []
     test_set_roc = []
     train_set_f1score = []
     test_set_f1score = []
```

```
[ ]: for trial in range(5):

         # Divide the data points into training set and test set.
         X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size =␣
     ↪5000, random_state = 42, shuffle = True, stratify = Y)
```

```
    #Standardizing the data
    scaler = StandardScaler().fit(X_train)
    X_train_sc = scaler.transform(X_train)
    X_test_sc = scaler.transform(X_test)

    max_features_list = [1, 2, 4, 6, 8, 12, 16, 20]
    scoring = {'ROC': make_scorer(roc_auc_score), 'Accuracy':␣
↪make_scorer(accuracy_score), 'F-1': make_scorer(f1_score) }





    #Finding the hyperparameter settings that give best ACCURACY on the␣
↪training set.
    acc_gs = GridSearchCV(RandomForestClassifier(n_estimators = 1024),
                    param_grid= {'max_features': max_features_list},
                    scoring = scoring, refit='Accuracy',␣
↪return_train_score=True, cv = 5)

    acc_gs.fit(X_train_sc, y_train.ravel())

    opt1_model = acc_gs.best_estimator_

    #Training the Model using the the best parameter settings for accuracy
    model1 = opt1_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the accuracy of this model on the training set
    train_set_accuracy.append(accuracy_score(y_train.ravel(), model1.
↪predict(X_train_sc)))


    #Finding and storing the accuracy of this model on the test set
    test_set_accuracy.append(accuracy_score(y_test.ravel(), model1.
↪predict(X_test_sc)))





    #Finding the hyperparameter settings that give best ROC on the training set.
    roc_gs = GridSearchCV(RandomForestClassifier(n_estimators = 1024),
                    param_grid= {'max_features': max_features_list},
                    scoring = scoring, refit='ROC', return_train_score=True, cv␣
↪= 5)

    roc_gs.fit(X_train_sc, y_train.ravel())

    opt2_model = roc_gs.best_estimator_

    #Training the Model using the the best parameter settings for ROC
    model2 = opt2_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the ROC of this model on the training set
    train_set_roc.append(roc_auc_score(y_train.ravel(), model2.
↪predict(X_train_sc)))


    #Finding and storing the ROC of this model on the test set
    test_set_roc.append(roc_auc_score(y_test.ravel(), model2.
↪predict(X_test_sc)))






    #Finding the hyperparameter settings that give best F-1 score on the␣
↪training set.
    f1_gs = GridSearchCV(RandomForestClassifier(n_estimators = 1024),
                    param_grid= {'max_features': max_features_list},
                    scoring = scoring, refit='F-1', return_train_score=True, cv␣
↪= 5)

    f1_gs.fit(X_train_sc, y_train.ravel())

    opt3_model = f1_gs.best_estimator_

    #Training the Model using the the best parameter settings for f1-score
    model3 = opt3_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the F-1 score of this model on the training set
    train_set_f1score.append(f1_score(y_train.ravel(), model3.
↪predict(X_train_sc)))
```

```
    #Finding and storing the F-1 of this model on the test set
    test_set_f1score.append(f1_score(y_test.ravel(), model3.predict(X_test_sc)))
```

[ ]:
```
print(train_set_accuracy)
print(test_set_accuracy)
print(train_set_roc)
print(test_set_roc)
print(train_set_f1score)
print(test_set_f1score)
```

[ ]:
```
#Calculating average performance metrics over 5 trials for RFs x Covtype Dataset
avg_test_set_accuracy = 0.0
avg_test_set_roc = 0.0
avg_test_set_f1score = 0.0

#Calculating average of accuracy
for a in test_set_accuracy:
    avg_test_set_accuracy = avg_test_set_accuracy + a

avg_test_set_accuracy = avg_test_set_accuracy / 5


#Calculating average of ROC
for r in test_set_roc:
    avg_test_set_roc = avg_test_set_roc + r

avg_test_set_roc = avg_test_set_roc / 5


#Calculating average of F1-Score
for r in test_set_f1score:
    avg_test_set_f1score = avg_test_set_f1score + r

avg_test_set_f1score = avg_test_set_f1score / 5

print(avg_test_set_accuracy)
print(avg_test_set_roc)
print(avg_test_set_f1score)
```

1.4 KNN

[ ]:
```
train_set_accuracy = []
test_set_accuracy = []
train_set_roc = []
test_set_roc = []
train_set_f1score = []
test_set_f1score = []
```

```
[ ]: for trial in range(5):
         # Divide the data points into training set and test set.
         X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size =␣
      ↪5000, random_state = 42, shuffle = True, stratify = Y)









         #Standardizing the data
         scaler = StandardScaler().fit(X_train)
         X_train_sc = scaler.transform(X_train)
         X_test_sc = scaler.transform(X_test)

         weights_list = ['uniform', 'distance']
         scoring = {'ROC': make_scorer(roc_auc_score), 'Accuracy':␣
      ↪make_scorer(accuracy_score), 'F-1': make_scorer(f1_score) }
         k_range = range(1, 500, 20)
         k_list = []
         for k in k_range:
             k_list.append(k)




         #Finding the hyperparameter settings that give best ACCURACY on the␣
      ↪training set.
         acc_gs = GridSearchCV(KNeighborsClassifier(),
                         param_grid= {'n_neighbors': k_list,'weights': weights_list},
                         scoring = scoring, refit='Accuracy',␣
      ↪return_train_score=True, cv = 5)

         acc_gs.fit(X_train_sc, y_train.ravel())

         opt1_model = acc_gs.best_estimator_


         #Training the Model using the the best parameter settings for accuracy
         model1 = opt1_model.fit(X_train_sc, y_train.ravel())


         #Finding and storing the accuracy of this model on the training set
         train_set_accuracy.append(accuracy_score(y_train.ravel(), model1.
      ↪predict(X_train_sc)))


         #Finding and storing the accuracy of this model on the test set
```

```
    test_set_accuracy.append(accuracy_score(y_test.ravel(), model1.
↪predict(X_test_sc)))






    #Finding the hyperparameter settings that give best ROC on the training set.
    roc_gs = GridSearchCV(KNeighborsClassifier(),
                          param_grid= {'n_neighbors': k_list,'weights': weights_list},
                          scoring = scoring, refit='ROC', return_train_score=True, cv␣
↪= 5)

    roc_gs.fit(X_train_sc, y_train.ravel())

    opt2_model = roc_gs.best_estimator_


    #Training the Model using the the best parameter settings for ROC
    model2 = opt2_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the ROC of this model on the training set
    train_set_roc.append(roc_auc_score(y_train.ravel(), model2.
↪predict(X_train_sc)))


    #Finding and storing the ROC of this model on the test set
    test_set_roc.append(roc_auc_score(y_test.ravel(), model2.
↪predict(X_test_sc)))






    #Finding the hyperparameter settings that give best F-1 score on the␣
↪training set.
    f1_gs = GridSearchCV(KNeighborsClassifier(),
                          param_grid= {'n_neighbors': k_list,'weights': weights_list},
                          scoring = scoring, refit='F-1', return_train_score=True, cv␣
↪= 5)

    f1_gs.fit(X_train_sc, y_train.ravel())

    opt3_model = f1_gs.best_estimator_
    #Training the Model using the the best parameter settings for f1-score
    model3 = opt3_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the F-1 score of this model on the training set
    train_set_f1score.append(f1_score(y_train.ravel(), model3.
↪predict(X_train_sc)))


    #Finding and storing the F-1 of this model on the test set
    test_set_f1score.append(f1_score(y_test.ravel(), model3.predict(X_test_sc)))
```

```
[ ]: print(train_set_accuracy)
     print(test_set_accuracy)
     print(train_set_roc)
     print(test_set_roc)
     print(train_set_f1score)
     print(test_set_f1score)
```

```
[ ]: #Calculating average performance metrics over 5 trials for KNN x Covtype Dataset
     avg_test_set_accuracy = 0.0
     avg_test_set_roc = 0.0
     avg_test_set_f1score = 0.0

     #Calculating average of accuracy
     for a in test_set_accuracy:
         avg_test_set_accuracy = avg_test_set_accuracy + a

     avg_test_set_accuracy = avg_test_set_accuracy / 5


     #Calculating average of ROC
     for r in test_set_roc:
         avg_test_set_roc = avg_test_set_roc + r

     avg_test_set_roc = avg_test_set_roc / 5


     #Calculating average of F1-Score
     for r in test_set_f1score:
         avg_test_set_f1score = avg_test_set_f1score + r

     avg_test_set_f1score = avg_test_set_f1score / 5

     print(avg_test_set_accuracy)
     print(avg_test_set_roc)
```

```python
print(avg_test_set_f1score)
```

COGS 118A - Final Project Code (LETTER.p1 DATASET)

March 18, 2021

1 Algorithm testing for LETTER.p1 dataset

```
[ ]: import numpy as np
     import matplotlib.pyplot as plt
     from sklearn import datasets
     from sklearn.linear_model import LogisticRegression
     from sklearn.model_selection import GridSearchCV
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import accuracy_score, roc_auc_score, make_scorer, f1_score
     from sklearn.preprocessing import StandardScaler
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.neighbors import KNeighborsClassifier
```

```
[ ]: # Letter Recognition dataset.
     letterp1 = datasets.fetch_openml(name = 'Letter')   #Load Letter.p1 dataset.
     X = letterp1.data                          # The shape of X is (20000, 16) which means
                                                # there are 20000 data points, each data␣
      ↪point
                                                # has 16 features.

     # Here, we convert the target variable to binary by assigning the letter "O" to␣
      ↪1 and the rest of the letters to 0
     # Y = 0 (or False): letter != "O"
     # Y = 1 (or True): letter = "O"

     # Thus, we use (adult.target == 'O')  to convert the target variable to binary
     # This line of code will assign:
     #      Y[i] = True  (which is equivalent to 1) if cov.target[k] == 'O'
     #      Y[i] = False (which is equivalent to 0)  if cov.target[k] != 'O'

     Y = (letterp1.target == 'O').reshape(-1,1).astype(np.float)


     # The shape of Y is (20000, 1), which means there are 48842 data points, each␣
      ↪data point has 1 target value.
```

```
Y[Y==0] = -1
```

1.1 Logistic Regression

```
[ ]: train_set_accuracy = []
     test_set_accuracy = []
     train_set_roc = []
     test_set_roc = []
     train_set_f1score = []
     test_set_f1score = []
```

```
[ ]: for trial in range(5):

         # Divide the data points into training set and test set.
         X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size =␣
      ↪5000, random_state = 42, shuffle = True, stratify = Y)









         #Standardizing the data
         scaler = StandardScaler().fit(X_train)
         X_train_sc = scaler.transform(X_train)
         X_test_sc = scaler.transform(X_test)

         C_list = [100000000, 10000000, 1000000, 100000, 10000, 1000, 100, 10, 1, 0.
      ↪1, 0.01, 0.001, 0.0001, 0.00001]
         scoring = {'ROC': make_scorer(roc_auc_score), 'Accuracy':␣
      ↪make_scorer(accuracy_score), 'F-1': make_scorer(f1_score) }









         #Finding the hyperparameter settings that give best ACCURACY on the␣
      ↪training set.
         acc_gs = GridSearchCV(LogisticRegression(penalty = 'l2', solver = 'lbfgs',␣
      ↪max_iter = 10000).fit(X_train_sc, y_train.ravel()),
                           param_grid= {'C': C_list},
                           scoring = scoring, refit='Accuracy', return_train_score=True,␣
      ↪cv = 5)

         acc_gs.fit(X_train_sc, y_train.ravel())
```

```
    opt1_model = acc_gs.best_estimator_


    #Training the Model using the the best parameter settings for accuracy
    model1 = opt1_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the accuracy of this model on the training set
    train_set_accuracy.append(accuracy_score(y_train.ravel(), model1.
↪predict(X_train_sc)))


    #Finding and storing the accuracy of this model on the test set
    test_set_accuracy.append(accuracy_score(y_test.ravel(), model1.
↪predict(X_test_sc)))






    #Finding the hyperparameter settings that give best ROC on the training set.
    roc_gs = GridSearchCV(LogisticRegression(penalty = 'l2', solver = 'lbfgs',␣
↪max_iter = 10000).fit(X_train_sc, y_train.ravel()),
                  param_grid= {'C': C_list},
                  scoring=scoring, refit='ROC', return_train_score=True, cv = 5)

    roc_gs.fit(X_train_sc, y_train.ravel())

    opt2_model = roc_gs.best_estimator_


    #Training the Model using the the best parameter settings for ROC
    model2 = opt2_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the ROC of this model on the training set
    train_set_roc.append(roc_auc_score(y_train.ravel(), model2.
↪predict(X_train_sc)))


    #Finding and storing the ROC of this model on the test set
    test_set_roc.append(roc_auc_score(y_test.ravel(), model2.
↪predict(X_test_sc)))
```

```
    #Finding the hyperparameter settings that give best F-1 score on the␣
↪training set.
    f1_gs = GridSearchCV(LogisticRegression(penalty = 'l2', solver = 'lbfgs',␣
↪max_iter = 10000).fit(X_train_sc, y_train.ravel()),
                  param_grid= {'C': C_list},
                  scoring=scoring, refit='F-1', return_train_score=True, cv = 5)

    f1_gs.fit(X_train_sc, y_train.ravel())

    opt3_model = f1_gs.best_estimator_


    #Training the Model using the the best parameter settings for f1-score
    model3 = opt3_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the F-1 score of this model on the training set
    train_set_f1score.append(f1_score(y_train.ravel(), model3.
↪predict(X_train_sc)))


    #Finding and storing the F-1 of this model on the test set
    test_set_f1score.append(f1_score(y_test.ravel(), model3.predict(X_test_sc)))
```

```
[ ]: print(train_set_accuracy)
     print(test_set_accuracy)
     print(train_set_roc)
     print(test_set_roc)
     print(train_set_f1score)
     print(test_set_f1score)
```

```
[ ]: #Calculating average performance metrics over 5 trials for Logistic Regression␣
     ↪x Letter.p1 Dataset
     avg_test_set_accuracy = 0.0
     avg_test_set_roc = 0.0
     avg_test_set_f1score = 0.0

     #Calculating average of accuracy
     for a in test_set_accuracy:
         avg_test_set_accuracy = avg_test_set_accuracy + a

     avg_test_set_accuracy = avg_test_set_accuracy / 5
```

```
#Calculating average of ROC
for r in test_set_roc:
    avg_test_set_roc = avg_test_set_roc + r

avg_test_set_roc = avg_test_set_roc / 5


#Calculating average of F1-Score
for r in test_set_f1score:
    avg_test_set_f1score = avg_test_set_f1score + r

avg_test_set_f1score = avg_test_set_f1score / 5

print(avg_test_set_accuracy)
print(avg_test_set_roc)
print(avg_test_set_f1score)
```

1.2 Random Forests

[ ]:
```
train_set_accuracy = []
test_set_accuracy = []
train_set_roc = []
test_set_roc = []
train_set_f1score = []
test_set_f1score = []
```

[ ]:
```
for trial in range(5):

    # Divide the data points into training set and test set.
    X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size =␣
 ↪5000, random_state = 42, shuffle = True, stratify = Y)






    #Standardizing the data
    scaler = StandardScaler().fit(X_train)
    X_train_sc = scaler.transform(X_train)
    X_test_sc = scaler.transform(X_test)

    max_features_list = [1, 2, 4, 6, 8, 12, 16, 20]
    scoring = {'ROC': make_scorer(roc_auc_score), 'Accuracy':␣
 ↪make_scorer(accuracy_score), 'F-1': make_scorer(f1_score) }
```

```
    #Finding the hyperparameter settings that give best ACCURACY on the␣
↪training set.
    acc_gs = GridSearchCV(RandomForestClassifier(n_estimators = 1024),
                    param_grid= {'max_features': max_features_list},
                    scoring = scoring, refit='Accuracy',␣
↪return_train_score=True, cv = 5)

    acc_gs.fit(X_train_sc, y_train.ravel())

    opt1_model = acc_gs.best_estimator_

    #Training the Model using the the best parameter settings for accuracy
    model1 = opt1_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the accuracy of this model on the training set
    train_set_accuracy.append(accuracy_score(y_train.ravel(), model1.
↪predict(X_train_sc)))


    #Finding and storing the accuracy of this model on the test set
    test_set_accuracy.append(accuracy_score(y_test.ravel(), model1.
↪predict(X_test_sc)))





    #Finding the hyperparameter settings that give best ROC on the training set.
    roc_gs = GridSearchCV(RandomForestClassifier(n_estimators = 1024),
                    param_grid= {'max_features': max_features_list},
                    scoring = scoring, refit='ROC', return_train_score=True, cv␣
↪= 5)

    roc_gs.fit(X_train_sc, y_train.ravel())

    opt2_model = roc_gs.best_estimator_

    #Training the Model using the the best parameter settings for ROC
    model2 = opt2_model.fit(X_train_sc, y_train.ravel())
    #Finding and storing the ROC of this model on the training set
    train_set_roc.append(roc_auc_score(y_train.ravel(), model2.
↪predict(X_train_sc)))


    #Finding and storing the ROC of this model on the test set
    test_set_roc.append(roc_auc_score(y_test.ravel(), model2.
↪predict(X_test_sc)))






    #Finding the hyperparameter settings that give best F-1 score on the␣
↪training set.
    f1_gs = GridSearchCV(RandomForestClassifier(n_estimators = 1024),
                    param_grid= {'max_features': max_features_list},
                    scoring = scoring, refit='F-1', return_train_score=True, cv␣
↪= 5)

    f1_gs.fit(X_train_sc, y_train.ravel())

    opt3_model = f1_gs.best_estimator_

    #Training the Model using the the best parameter settings for f1-score
    model3 = opt3_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the F-1 score of this model on the training set
    train_set_f1score.append(f1_score(y_train.ravel(), model3.
↪predict(X_train_sc)))


    #Finding and storing the F-1 of this model on the test set
    test_set_f1score.append(f1_score(y_test.ravel(), model3.predict(X_test_sc)))
```

[ ]:
```
print(train_set_accuracy)
print(test_set_accuracy)
print(train_set_roc)
print(test_set_roc)
print(train_set_f1score)
print(test_set_f1score)
```

[ ]:
```
#Calculating average performance metrics over 5 trials for RFs x Letter.p1␣
↪Dataset
```

```
avg_test_set_accuracy = 0.0
avg_test_set_roc = 0.0
avg_test_set_f1score = 0.0

#Calculating average of accuracy
for a in test_set_accuracy:
    avg_test_set_accuracy = avg_test_set_accuracy + a

avg_test_set_accuracy = avg_test_set_accuracy / 5


#Calculating average of ROC
for r in test_set_roc:
    avg_test_set_roc = avg_test_set_roc + r

avg_test_set_roc = avg_test_set_roc / 5


#Calculating average of F1-Score
for r in test_set_f1score:
    avg_test_set_f1score = avg_test_set_f1score + r

avg_test_set_f1score = avg_test_set_f1score / 5

print(avg_test_set_accuracy)
print(avg_test_set_roc)
print(avg_test_set_f1score)
```

1.3 KNN

```
[ ]: train_set_accuracy = []
     test_set_accuracy = []
     train_set_roc = []
     test_set_roc = []
     train_set_f1score = []
     test_set_f1score = []
```

```
[ ]: for trial in range(5):
         # Divide the data points into training set and test set.
         X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size =␣
      ↪5000, random_state = 42, shuffle = True, stratify = Y)
```

```
    #Standardizing the data
    scaler = StandardScaler().fit(X_train)
    X_train_sc = scaler.transform(X_train)
    X_test_sc = scaler.transform(X_test)

    weights_list = ['uniform', 'distance']
    scoring = {'ROC': make_scorer(roc_auc_score), 'Accuracy':␣
↪make_scorer(accuracy_score), 'F-1': make_scorer(f1_score) }
    k_range = range(1, 500, 20)
    k_list = []
    for k in k_range:
        k_list.append(k)



    #Finding the hyperparameter settings that give best ACCURACY on the␣
↪training set.
    acc_gs = GridSearchCV(KNeighborsClassifier(),
                    param_grid= {'n_neighbors': k_list,'weights': weights_list},
                    scoring = scoring, refit='Accuracy',␣
↪return_train_score=True, cv = 5)

    acc_gs.fit(X_train_sc, y_train.ravel())

    opt1_model = acc_gs.best_estimator_


    #Training the Model using the the best parameter settings for accuracy
    model1 = opt1_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the accuracy of this model on the training set
    train_set_accuracy.append(accuracy_score(y_train.ravel(), model1.
↪predict(X_train_sc)))


    #Finding and storing the accuracy of this model on the test set
    test_set_accuracy.append(accuracy_score(y_test.ravel(), model1.
↪predict(X_test_sc)))





    #Finding the hyperparameter settings that give best ROC on the training set.
    roc_gs = GridSearchCV(KNeighborsClassifier(),
                    param_grid= {'n_neighbors': k_list,'weights': weights_list},
                     scoring = scoring, refit='ROC', return_train_score=True, cv␣
↪= 5)

    roc_gs.fit(X_train_sc, y_train.ravel())

    opt2_model = roc_gs.best_estimator_


    #Training the Model using the the best parameter settings for ROC
    model2 = opt2_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the ROC of this model on the training set
    train_set_roc.append(roc_auc_score(y_train.ravel(), model2.
↪predict(X_train_sc)))


    #Finding and storing the ROC of this model on the test set
    test_set_roc.append(roc_auc_score(y_test.ravel(), model2.
↪predict(X_test_sc)))






    #Finding the hyperparameter settings that give best F-1 score on the␣
↪training set.
    f1_gs = GridSearchCV(KNeighborsClassifier(),
                     param_grid= {'n_neighbors': k_list,'weights': weights_list},
                     scoring = scoring, refit='F-1', return_train_score=True, cv␣
↪= 5)

    f1_gs.fit(X_train_sc, y_train.ravel())

    opt3_model = f1_gs.best_estimator_


    #Training the Model using the the best parameter settings for f1-score
    model3 = opt3_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the F-1 score of this model on the training set
    train_set_f1score.append(f1_score(y_train.ravel(), model3.
↪predict(X_train_sc)))
```

```
    #Finding and storing the F-1 of this model on the test set
    test_set_f1score.append(f1_score(y_test.ravel(), model3.predict(X_test_sc)))
```

[ ]:
```
print(train_set_accuracy)
print(test_set_accuracy)
print(train_set_roc)
print(test_set_roc)
print(train_set_f1score)
print(test_set_f1score)
```

[ ]:
```
#Calculating average performance metrics over 5 trials for KNN x Letter.p1␣
 ↪Dataset
avg_test_set_accuracy = 0.0
avg_test_set_roc = 0.0
avg_test_set_f1score = 0.0

#Calculating average of accuracy
for a in test_set_accuracy:
    avg_test_set_accuracy = avg_test_set_accuracy + a

avg_test_set_accuracy = avg_test_set_accuracy / 5


#Calculating average of ROC
for r in test_set_roc:
    avg_test_set_roc = avg_test_set_roc + r

avg_test_set_roc = avg_test_set_roc / 5


#Calculating average of F1-Score
for r in test_set_f1score:
    avg_test_set_f1score = avg_test_set_f1score + r

avg_test_set_f1score = avg_test_set_f1score / 5

print(avg_test_set_accuracy)
print(avg_test_set_roc)
print(avg_test_set_f1score)
```

COGS 118A - Final Project Code (Letter.p2 DATASET)

March 18, 2021

1 Algorithm Testing for Letter.p2 dataset

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_auc_score, make_scorer, f1_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
```

[ ]:
```
# Letter Recognition dataset.
letterp2 = datasets.fetch_openml(name = 'Letter')   #Load Letter.p2 dataset.
X = letterp2.data                                   # The shape of X is (20000, 16) which means
                                                    # there are 20000 data points, each data␣
 ↪point
                                                    # has 16 features.

# Here, we convert the target variable to binary by assigning the letters "A-M"␣
 ↪to 1 and the rest of the letters to 0
# Y = 0 (or False): letter != "A-M"
# Y = 1 (or True): letter = "A-M"

# Thus, we use (adult.target >= A and adult.target <=M)  to convert the target␣
 ↪variable to binary
# This line of code will assign:
#       Y[i] = True  (which is equivalent to 1) if cov.target[k] >= A and cov.
 ↪target[k] <=M
#       Y[i] = False (which is equivalent to 0)  if cov.target[k] < A and cov.
 ↪target[k] > M

Y = np.logical_and(letterp2.target >= 'A', letterp2.target <= 'M').
 ↪reshape(-1,1).astype(np.float)
```

```
# The shape of Y is (20000, 1), which means there are 20000 data points, each␣
↪data point has 1 target value.

Y[Y==0] = -1
```

1.1 Logistic Regression

[ ]:
```
train_set_accuracy = []
test_set_accuracy = []
train_set_roc = []
test_set_roc = []
train_set_f1score = []
test_set_f1score = []
```

[ ]:
```
for trial in range(5):

    # Divide the data points into training set and test set.
    X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size =␣
↪5000, random_state = 42, shuffle = True, stratify = Y)








    #Standardizing the data
    scaler = StandardScaler().fit(X_train)
    X_train_sc = scaler.transform(X_train)
    X_test_sc = scaler.transform(X_test)

    C_list = [100000000, 10000000, 1000000, 100000, 10000, 1000, 100, 10, 1, 0.
↪1, 0.01, 0.001, 0.0001, 0.00001]
    scoring = {'ROC': make_scorer(roc_auc_score), 'Accuracy':␣
↪make_scorer(accuracy_score), 'F-1': make_scorer(f1_score) }








    #Finding the hyperparameter settings that give best ACCURACY on the␣
↪training set.
    acc_gs = GridSearchCV(LogisticRegression(penalty = 'l2', solver = 'lbfgs',␣
↪max_iter = 10000).fit(X_train_sc, y_train.ravel()),
                  param_grid= {'C': C_list},
                  scoring = scoring, refit='Accuracy', return_train_score=True,␣
↪cv = 5)

    acc_gs.fit(X_train_sc, y_train.ravel())

    opt1_model = acc_gs.best_estimator_


    #Training the Model using the the best parameter settings for accuracy
    model1 = opt1_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the accuracy of this model on the training set
    train_set_accuracy.append(accuracy_score(y_train.ravel(), model1.
↪predict(X_train_sc)))


    #Finding and storing the accuracy of this model on the test set
    test_set_accuracy.append(accuracy_score(y_test.ravel(), model1.
↪predict(X_test_sc)))






    #Finding the hyperparameter settings that give best ROC on the training set.
    roc_gs = GridSearchCV(LogisticRegression(penalty = 'l2', solver = 'lbfgs',␣
↪max_iter = 10000).fit(X_train_sc, y_train.ravel()),
                  param_grid= {'C': C_list},
                  scoring=scoring, refit='ROC', return_train_score=True, cv = 5)

    roc_gs.fit(X_train_sc, y_train.ravel())

    opt2_model = roc_gs.best_estimator_


    #Training the Model using the the best parameter settings for ROC
    model2 = opt2_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the ROC of this model on the training set
    train_set_roc.append(roc_auc_score(y_train.ravel(), model2.
↪predict(X_train_sc)))


    #Finding and storing the ROC of this model on the test set
    test_set_roc.append(roc_auc_score(y_test.ravel(), model2.
↪predict(X_test_sc)))








    #Finding the hyperparameter settings that give best F-1 score on the␣
↪training set.
    f1_gs = GridSearchCV(LogisticRegression(penalty = 'l2', solver = 'lbfgs',␣
↪max_iter = 10000).fit(X_train_sc, y_train.ravel()),
                    param_grid= {'C': C_list},
                    scoring=scoring, refit='F-1', return_train_score=True, cv = 5)

    f1_gs.fit(X_train_sc, y_train.ravel())

    opt3_model = f1_gs.best_estimator_


    #Training the Model using the the best parameter settings for f1-score
    model3 = opt3_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the F-1 score of this model on the training set
    train_set_f1score.append(f1_score(y_train.ravel(), model3.
↪predict(X_train_sc)))


    #Finding and storing the F-1 of this model on the test set
    test_set_f1score.append(f1_score(y_test.ravel(), model3.predict(X_test_sc)))
```

[ ]:
```
print(train_set_accuracy)
print(test_set_accuracy)
print(train_set_roc)
print(test_set_roc)
print(train_set_f1score)
print(test_set_f1score)
```

[ ]:
```
#Calculating average performance metrics over 5 trials for Logistic Regression␣
↪x Letter.p2 Dataset
avg_test_set_accuracy = 0.0
avg_test_set_roc = 0.0
avg_test_set_f1score = 0.0

#Calculating average of accuracy
```

```
for a in test_set_accuracy:
    avg_test_set_accuracy = avg_test_set_accuracy + a

avg_test_set_accuracy = avg_test_set_accuracy / 5


#Calculating average of ROC
for r in test_set_roc:
    avg_test_set_roc = avg_test_set_roc + r

avg_test_set_roc = avg_test_set_roc / 5


#Calculating average of F1-Score
for r in test_set_f1score:
    avg_test_set_f1score = avg_test_set_f1score + r

avg_test_set_f1score = avg_test_set_f1score / 5

print(avg_test_set_accuracy)
print(avg_test_set_roc)
print(avg_test_set_f1score)
```

1.2 Random Forests

```
[ ]: train_set_accuracy = []
     test_set_accuracy = []
     train_set_roc = []
     test_set_roc = []
     train_set_f1score = []
     test_set_f1score = []
```

```
[ ]: for trial in range(5):

         # Divide the data points into training set and test set.
         X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size =␣
       ↪5000, random_state = 42, shuffle = True, stratify = Y)








         #Standardizing the data
         scaler = StandardScaler().fit(X_train)
         X_train_sc = scaler.transform(X_train)
         X_test_sc = scaler.transform(X_test)
```

```
   max_features_list = [1, 2, 4, 6, 8, 12, 16, 20]
   scoring = {'ROC': make_scorer(roc_auc_score), 'Accuracy':␣
↪make_scorer(accuracy_score), 'F-1': make_scorer(f1_score) }






   #Finding the hyperparameter settings that give best ACCURACY on the␣
↪training set.
   acc_gs = GridSearchCV(RandomForestClassifier(n_estimators = 1024),
                   param_grid= {'max_features': max_features_list},
                   scoring = scoring, refit='Accuracy',␣
↪return_train_score=True, cv = 5)

   acc_gs.fit(X_train_sc, y_train.ravel())

   opt1_model = acc_gs.best_estimator_

   #Training the Model using the the best parameter settings for accuracy
   model1 = opt1_model.fit(X_train_sc, y_train.ravel())


   #Finding and storing the accuracy of this model on the training set
   train_set_accuracy.append(accuracy_score(y_train.ravel(), model1.
↪predict(X_train_sc)))


   #Finding and storing the accuracy of this model on the test set
   test_set_accuracy.append(accuracy_score(y_test.ravel(), model1.
↪predict(X_test_sc)))






   #Finding the hyperparameter settings that give best ROC on the training set.
   roc_gs = GridSearchCV(RandomForestClassifier(n_estimators = 1024),
                   param_grid= {'max_features': max_features_list},
                   scoring = scoring, refit='ROC', return_train_score=True, cv␣
↪= 5)

   roc_gs.fit(X_train_sc, y_train.ravel())
```

```
    opt2_model = roc_gs.best_estimator_

    #Training the Model using the the best parameter settings for ROC
    model2 = opt2_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the ROC of this model on the training set
    train_set_roc.append(roc_auc_score(y_train.ravel(), model2.
↪predict(X_train_sc)))


    #Finding and storing the ROC of this model on the test set
    test_set_roc.append(roc_auc_score(y_test.ravel(), model2.
↪predict(X_test_sc)))







    #Finding the hyperparameter settings that give best F-1 score on the␣
↪training set.
    f1_gs = GridSearchCV(RandomForestClassifier(n_estimators = 1024),
                    param_grid= {'max_features': max_features_list},
                    scoring = scoring, refit='F-1', return_train_score=True, cv␣
↪= 5)

    f1_gs.fit(X_train_sc, y_train.ravel())

    opt3_model = f1_gs.best_estimator_

    #Training the Model using the the best parameter settings for f1-score
    model3 = opt3_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the F-1 score of this model on the training set
    train_set_f1score.append(f1_score(y_train.ravel(), model3.
↪predict(X_train_sc)))


    #Finding and storing the F-1 of this model on the test set
    test_set_f1score.append(f1_score(y_test.ravel(), model3.predict(X_test_sc)))
```

[ ]:
```
print(train_set_accuracy)
print(test_set_accuracy)
print(train_set_roc)
print(test_set_roc)
print(train_set_f1score)
print(test_set_f1score)
```

[ ]:
```
#Calculating average performance metrics over 5 trials for RFs x Letter.p2␣
↪Dataset
avg_test_set_accuracy = 0.0
avg_test_set_roc = 0.0
avg_test_set_f1score = 0.0

#Calculating average of accuracy
for a in test_set_accuracy:
    avg_test_set_accuracy = avg_test_set_accuracy + a

avg_test_set_accuracy = avg_test_set_accuracy / 5


#Calculating average of ROC
for r in test_set_roc:
    avg_test_set_roc = avg_test_set_roc + r

avg_test_set_roc = avg_test_set_roc / 5


#Calculating average of F1-Score
for r in test_set_f1score:
    avg_test_set_f1score = avg_test_set_f1score + r

avg_test_set_f1score = avg_test_set_f1score / 5

print(avg_test_set_accuracy)
print(avg_test_set_roc)
print(avg_test_set_f1score)
```

1.3 KNN

[ ]:
```
train_set_accuracy = []
test_set_accuracy = []
train_set_roc = []
test_set_roc = []
train_set_f1score = []
test_set_f1score = []
```

[ ]:
```
for trial in range(5):
    # Divide the data points into training set and test set.
    X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size =␣
↪5000, random_state = 42, shuffle = True, stratify = Y)
```

```
#Standardizing the data
scaler = StandardScaler().fit(X_train)
X_train_sc = scaler.transform(X_train)
X_test_sc = scaler.transform(X_test)

weights_list = ['uniform', 'distance']
scoring = {'ROC': make_scorer(roc_auc_score), 'Accuracy':␣
↪make_scorer(accuracy_score), 'F-1': make_scorer(f1_score) }
k_range = range(1, 500, 20)
k_list = []
for k in k_range:
    k_list.append(k)



#Finding the hyperparameter settings that give best ACCURACY on the␣
↪training set.
acc_gs = GridSearchCV(KNeighborsClassifier(),
                param_grid= {'n_neighbors': k_list,'weights': weights_list},
                scoring = scoring, refit='Accuracy',␣
↪return_train_score=True, cv = 5)

acc_gs.fit(X_train_sc, y_train.ravel())

opt1_model = acc_gs.best_estimator_

#Training the Model using the the best parameter settings for accuracy
model1 = opt1_model.fit(X_train_sc, y_train.ravel())


#Finding and storing the accuracy of this model on the training set
train_set_accuracy.append(accuracy_score(y_train.ravel(), model1.
↪predict(X_train_sc)))


#Finding and storing the accuracy of this model on the test set
test_set_accuracy.append(accuracy_score(y_test.ravel(), model1.
↪predict(X_test_sc)))
```

```
    #Finding the hyperparameter settings that give best ROC on the training set.
    roc_gs = GridSearchCV(KNeighborsClassifier(),
                    param_grid= {'n_neighbors': k_list,'weights': weights_list},
                    scoring = scoring, refit='ROC', return_train_score=True, cv␣
↪= 5)

    roc_gs.fit(X_train_sc, y_train.ravel())

    opt2_model = roc_gs.best_estimator_

    #Training the Model using the the best parameter settings for ROC
    model2 = opt2_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the ROC of this model on the training set
    train_set_roc.append(roc_auc_score(y_train.ravel(), model2.
↪predict(X_train_sc)))


    #Finding and storing the ROC of this model on the test set
    test_set_roc.append(roc_auc_score(y_test.ravel(), model2.
↪predict(X_test_sc)))





    #Finding the hyperparameter settings that give best F-1 score on the␣
↪training set.
    f1_gs = GridSearchCV(KNeighborsClassifier(),
                    param_grid= {'n_neighbors': k_list,'weights': weights_list},
                    scoring = scoring, refit='F-1', return_train_score=True, cv␣
↪= 5)

    f1_gs.fit(X_train_sc, y_train.ravel())

    opt3_model = f1_gs.best_estimator_

    #Training the Model using the the best parameter settings for f1-score
    model3 = opt3_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the F-1 score of this model on the training set
    train_set_f1score.append(f1_score(y_train.ravel(), model3.
↪predict(X_train_sc)))
```

```
    #Finding and storing the F-1 of this model on the test set
    test_set_f1score.append(f1_score(y_test.ravel(), model3.predict(X_test_sc)))
```

```
[ ]: print(train_set_accuracy)
     print(test_set_accuracy)
     print(train_set_roc)
     print(test_set_roc)
     print(train_set_f1score)
     print(test_set_f1score)
```

```
[ ]: #Calculating average performance metrics over 5 trials for KNN x Letter.p2␣
      ↪Dataset
     avg_test_set_accuracy = 0.0
     avg_test_set_roc = 0.0
     avg_test_set_f1score = 0.0

     #Calculating average of accuracy
     for a in test_set_accuracy:
         avg_test_set_accuracy = avg_test_set_accuracy + a

     avg_test_set_accuracy = avg_test_set_accuracy / 5


     #Calculating average of ROC
     for r in test_set_roc:
         avg_test_set_roc = avg_test_set_roc + r

     avg_test_set_roc = avg_test_set_roc / 5


     #Calculating average of F1-Score
     for r in test_set_f1score:
         avg_test_set_f1score = avg_test_set_f1score + r

     avg_test_set_f1score = avg_test_set_f1score / 5

     print(avg_test_set_accuracy)
     print(avg_test_set_roc)
     print(avg_test_set_f1score)
```

COGS 118A - Final Project Code (Connect4 DATASET)

March 18, 2021

1 Algorithm testing for Connect-4 dataset

```
[ ]: import numpy as np
     import matplotlib.pyplot as plt
     from sklearn import datasets
     from sklearn.linear_model import LogisticRegression
     from sklearn.model_selection import GridSearchCV
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import accuracy_score, roc_auc_score, make_scorer, f1_score
     from sklearn.preprocessing import StandardScaler
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.neighbors import KNeighborsClassifier
```

```
[ ]: # Connect-4 dataset.
     connect = datasets.fetch_openml(name = 'Connect-4', version = 2)    #Load␣
      ↪Connect-4 dataset.
     X = connect.data                     # The shape of X is (67557, 42) which means
                                          # there are 67557 data points, each data␣
      ↪point
                                          # has 42 features.

     # Here for convenience, we divide the 3 kinds of outcomes into 2 groups:
     # Y = 0 (or False): loss (original value '1') / draw (original value '0')
     # Y = 1 (or True): win (original value '2')

     # Thus, we use (connect.target > 1.5) to divide the target into 2 groups.
     # This line of code will assign:
     #       Y[i] = True  (which is equivalent to 1) if connect.target[k] > '1.5'␣
      ↪(Win)
     #       Y[i] = False (which is equivalent to 0)  if connect.target[k] <= '1.5'␣
      ↪(Loss / Draw)

     Y = (connect.target > '1.5').reshape(-1,1).astype(np.float)

     # The shape of Y is (67557, 1), which means there are 48842 data points, each␣
      ↪data point has 1 target value.
```

```
Y[Y==0] = -1
```

1.1 Logistic Regression

```
[ ]: train_set_accuracy = []
     test_set_accuracy = []
     train_set_roc = []
     test_set_roc = []
     train_set_f1score = []
     test_set_f1score = []
```

```
[ ]: for trial in range(5):

         # Divide the data points into training set and test set.
         X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size =␣
      ↪5000, random_state = 42, shuffle = True, stratify = Y)







         #Standardizing the data
         scaler = StandardScaler().fit(X_train)
         X_train_sc = scaler.transform(X_train)
         X_test_sc = scaler.transform(X_test)

         C_list = [100000000, 10000000, 1000000, 100000, 10000, 1000, 100, 10, 1, 0.
      ↪1, 0.01, 0.001, 0.0001, 0.00001]
         scoring = {'ROC': make_scorer(roc_auc_score), 'Accuracy':␣
      ↪make_scorer(accuracy_score), 'F-1': make_scorer(f1_score) }







         #Finding the hyperparameter settings that give best ACCURACY on the␣
      ↪training set.
         acc_gs = GridSearchCV(LogisticRegression(penalty = 'l2', solver = 'lbfgs',␣
      ↪max_iter = 10000).fit(X_train_sc, y_train.ravel()),
                               param_grid= {'C': C_list},
                               scoring = scoring, refit='Accuracy', return_train_score=True,␣
      ↪cv = 5)
```

```
    acc_gs.fit(X_train_sc, y_train.ravel())

    opt1_model = acc_gs.best_estimator_


    #Training the Model using the the best parameter settings for accuracy
    model1 = opt1_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the accuracy of this model on the training set
    train_set_accuracy.append(accuracy_score(y_train.ravel(), model1.
↪predict(X_train_sc)))


    #Finding and storing the accuracy of this model on the test set
    test_set_accuracy.append(accuracy_score(y_test.ravel(), model1.
↪predict(X_test_sc)))






    #Finding the hyperparameter settings that give best ROC on the training set.
    roc_gs = GridSearchCV(LogisticRegression(penalty = 'l2', solver = 'lbfgs',␣
↪max_iter = 10000).fit(X_train_sc, y_train.ravel()),
                          param_grid= {'C': C_list},
                          scoring=scoring, refit='ROC', return_train_score=True, cv = 5)

    roc_gs.fit(X_train_sc, y_train.ravel())

    opt2_model = roc_gs.best_estimator_


    #Training the Model using the the best parameter settings for ROC
    model2 = opt2_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the ROC of this model on the training set
    train_set_roc.append(roc_auc_score(y_train.ravel(), model2.
↪predict(X_train_sc)))


    #Finding and storing the ROC of this model on the test set
    test_set_roc.append(roc_auc_score(y_test.ravel(), model2.
↪predict(X_test_sc)))
```

```
    #Finding the hyperparameter settings that give best F-1 score on the␣
↪training set.
    f1_gs = GridSearchCV(LogisticRegression(penalty = 'l2', solver = 'lbfgs',␣
↪max_iter = 10000).fit(X_train_sc, y_train.ravel()),
                    param_grid= {'C': C_list},
                    scoring=scoring, refit='F-1', return_train_score=True, cv = 5)

    f1_gs.fit(X_train_sc, y_train.ravel())

    opt3_model = f1_gs.best_estimator_


    #Training the Model using the the best parameter settings for f1-score
    model3 = opt3_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the F-1 score of this model on the training set
    train_set_f1score.append(f1_score(y_train.ravel(), model3.
↪predict(X_train_sc)))


    #Finding and storing the F-1 of this model on the test set
    test_set_f1score.append(f1_score(y_test.ravel(), model3.predict(X_test_sc)))
```

[ ]:
```
print(train_set_accuracy)
print(test_set_accuracy)
print(train_set_roc)
print(test_set_roc)
print(train_set_f1score)
print(test_set_f1score)
```

[ ]:
```
#Calculating average performance metrics over 5 trials for Logistic Regression␣
↪x Connect4 Dataset
avg_test_set_accuracy = 0.0
avg_test_set_roc = 0.0
avg_test_set_f1score = 0.0

#Calculating average of accuracy
for a in test_set_accuracy:
    avg_test_set_accuracy = avg_test_set_accuracy + a

avg_test_set_accuracy = avg_test_set_accuracy / 5
```

```
#Calculating average of ROC
for r in test_set_roc:
    avg_test_set_roc = avg_test_set_roc + r

avg_test_set_roc = avg_test_set_roc / 5


#Calculating average of F1-Score
for r in test_set_f1score:
    avg_test_set_f1score = avg_test_set_f1score + r

avg_test_set_f1score = avg_test_set_f1score / 5

print(avg_test_set_accuracy)
print(avg_test_set_roc)
print(avg_test_set_f1score)
```

1.2 Random Forests

```
[ ]: train_set_accuracy = []
     test_set_accuracy = []
     train_set_roc = []
     test_set_roc = []
     train_set_f1score = []
     test_set_f1score = []
```

```
[ ]: for trial in range(5):

         # Divide the data points into training set and test set.
         X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size =␣
      ↪5000, random_state = 42, shuffle = True, stratify = Y)






         #Standardizing the data
         scaler = StandardScaler().fit(X_train)
         X_train_sc = scaler.transform(X_train)
         X_test_sc = scaler.transform(X_test)

         max_features_list = [1, 2, 4, 6, 8, 12, 16, 20]
         scoring = {'ROC': make_scorer(roc_auc_score), 'Accuracy':␣
      ↪make_scorer(accuracy_score), 'F-1': make_scorer(f1_score) }
```

```
    #Finding the hyperparameter settings that give best ACCURACY on the␣
↪training set.
    acc_gs = GridSearchCV(RandomForestClassifier(n_estimators = 1024),
                    param_grid= {'max_features': max_features_list},
                    scoring = scoring, refit='Accuracy',␣
↪return_train_score=True, cv = 5)

    acc_gs.fit(X_train_sc, y_train.ravel())

    opt1_model = acc_gs.best_estimator_

    #Training the Model using the the best parameter settings for accuracy
    model1 = opt1_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the accuracy of this model on the training set
    train_set_accuracy.append(accuracy_score(y_train.ravel(), model1.
↪predict(X_train_sc)))


    #Finding and storing the accuracy of this model on the test set
    test_set_accuracy.append(accuracy_score(y_test.ravel(), model1.
↪predict(X_test_sc)))





    #Finding the hyperparameter settings that give best ROC on the training set.
    roc_gs = GridSearchCV(RandomForestClassifier(n_estimators = 1024),
                    param_grid= {'max_features': max_features_list},
                    scoring = scoring, refit='ROC', return_train_score=True, cv␣
↪= 5)

    roc_gs.fit(X_train_sc, y_train.ravel())

    opt2_model = roc_gs.best_estimator_

    #Training the Model using the the best parameter settings for ROC
    model2 = opt2_model.fit(X_train_sc, y_train.ravel())
    #Finding and storing the ROC of this model on the training set
    train_set_roc.append(roc_auc_score(y_train.ravel(), model2.
↪predict(X_train_sc)))


    #Finding and storing the ROC of this model on the test set
    test_set_roc.append(roc_auc_score(y_test.ravel(), model2.
↪predict(X_test_sc)))






    #Finding the hyperparameter settings that give best F-1 score on the␣
↪training set.
    f1_gs = GridSearchCV(RandomForestClassifier(n_estimators = 1024),
                    param_grid= {'max_features': max_features_list},
                    scoring = scoring, refit='F-1', return_train_score=True, cv␣
↪= 5)

    f1_gs.fit(X_train_sc, y_train.ravel())

    opt3_model = f1_gs.best_estimator_

    #Training the Model using the the best parameter settings for f1-score
    model3 = opt3_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the F-1 score of this model on the training set
    train_set_f1score.append(f1_score(y_train.ravel(), model3.
↪predict(X_train_sc)))


    #Finding and storing the F-1 of this model on the test set
    test_set_f1score.append(f1_score(y_test.ravel(), model3.predict(X_test_sc)))
```

[ ]:
```
print(train_set_accuracy)
print(test_set_accuracy)
print(train_set_roc)
print(test_set_roc)
print(train_set_f1score)
print(test_set_f1score)
```

```
[ ]: #Calculating average performance metrics over 5 trials for RFs x Connect4␣
     ↪Dataset
     avg_test_set_accuracy = 0.0
     avg_test_set_roc = 0.0
     avg_test_set_f1score = 0.0

     #Calculating average of accuracy
     for a in test_set_accuracy:
         avg_test_set_accuracy = avg_test_set_accuracy + a

     avg_test_set_accuracy = avg_test_set_accuracy / 5


     #Calculating average of ROC
     for r in test_set_roc:
         avg_test_set_roc = avg_test_set_roc + r

     avg_test_set_roc = avg_test_set_roc / 5


     #Calculating average of F1-Score
     for r in test_set_f1score:
         avg_test_set_f1score = avg_test_set_f1score + r

     avg_test_set_f1score = avg_test_set_f1score / 5

     print(avg_test_set_accuracy)
     print(avg_test_set_roc)
     print(avg_test_set_f1score)
```

1.3 KNN

```
[ ]: train_set_accuracy = []
     test_set_accuracy = []
     train_set_roc = []
     test_set_roc = []
     train_set_f1score = []
     test_set_f1score = []
```

```
[ ]: for trial in range(5):
         # Divide the data points into training set and test set.
         X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size =␣
     ↪5000, random_state = 42, shuffle = True, stratify = Y)
```

```
    #Standardizing the data
    scaler = StandardScaler().fit(X_train)
    X_train_sc = scaler.transform(X_train)
    X_test_sc = scaler.transform(X_test)

    weights_list = ['uniform', 'distance']
    scoring = {'ROC': make_scorer(roc_auc_score), 'Accuracy':␣
↪make_scorer(accuracy_score), 'F-1': make_scorer(f1_score) }
    k_range = range(1, 500, 20)
    k_list = []
    for k in k_range:
        k_list.append(k)



    #Finding the hyperparameter settings that give best ACCURACY on the␣
↪training set.
    acc_gs = GridSearchCV(KNeighborsClassifier(),
                    param_grid= {'n_neighbors': k_list,'weights': weights_list},
                    scoring = scoring, refit='Accuracy',␣
↪return_train_score=True, cv = 5)

    acc_gs.fit(X_train_sc, y_train.ravel())

    opt1_model = acc_gs.best_estimator_


    #Training the Model using the the best parameter settings for accuracy
    model1 = opt1_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the accuracy of this model on the training set
    train_set_accuracy.append(accuracy_score(y_train.ravel(), model1.
↪predict(X_train_sc)))


    #Finding and storing the accuracy of this model on the test set
    test_set_accuracy.append(accuracy_score(y_test.ravel(), model1.
↪predict(X_test_sc)))





    #Finding the hyperparameter settings that give best ROC on the training set.
    roc_gs = GridSearchCV(KNeighborsClassifier(),
                          param_grid= {'n_neighbors': k_list,'weights': weights_list},
                          scoring = scoring, refit='ROC', return_train_score=True, cv␣
↪= 5)

    roc_gs.fit(X_train_sc, y_train.ravel())

    opt2_model = roc_gs.best_estimator_


    #Training the Model using the the best parameter settings for ROC
    model2 = opt2_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the ROC of this model on the training set
    train_set_roc.append(roc_auc_score(y_train.ravel(), model2.
↪predict(X_train_sc)))


    #Finding and storing the ROC of this model on the test set
    test_set_roc.append(roc_auc_score(y_test.ravel(), model2.
↪predict(X_test_sc)))





    #Finding the hyperparameter settings that give best F-1 score on the␣
↪training set.
    f1_gs = GridSearchCV(KNeighborsClassifier(),
                          param_grid= {'n_neighbors': k_list,'weights': weights_list},
                          scoring = scoring, refit='F-1', return_train_score=True, cv␣
↪= 5)

    f1_gs.fit(X_train_sc, y_train.ravel())

    opt3_model = f1_gs.best_estimator_


    #Training the Model using the the best parameter settings for f1-score
    model3 = opt3_model.fit(X_train_sc, y_train.ravel())


    #Finding and storing the F-1 score of this model on the training set
    train_set_f1score.append(f1_score(y_train.ravel(), model3.
↪predict(X_train_sc)))
```

```
    #Finding and storing the F-1 of this model on the test set
    test_set_f1score.append(f1_score(y_test.ravel(), model3.predict(X_test_sc)))
```

```
[ ]: print(train_set_accuracy)
     print(test_set_accuracy)
     print(train_set_roc)
     print(test_set_roc)
     print(train_set_f1score)
     print(test_set_f1score)
```

```
[ ]: #Calculating average performance metrics over 5 trials for KNN x Connect-4␣
      ↪Dataset
     avg_test_set_accuracy = 0.0
     avg_test_set_roc = 0.0
     avg_test_set_f1score = 0.0

     #Calculating average of accuracy
     for a in test_set_accuracy:
         avg_test_set_accuracy = avg_test_set_accuracy + a

     avg_test_set_accuracy = avg_test_set_accuracy / 5


     #Calculating average of ROC
     for r in test_set_roc:
         avg_test_set_roc = avg_test_set_roc + r

     avg_test_set_roc = avg_test_set_roc / 5


     #Calculating average of F1-Score
     for r in test_set_f1score:
         avg_test_set_f1score = avg_test_set_f1score + r

     avg_test_set_f1score = avg_test_set_f1score / 5

     print(avg_test_set_accuracy)
     print(avg_test_set_roc)
     print(avg_test_set_f1score)
```