Email Classifier - CS470 Final Project

Aiden Seay | May 3, 2024

Project Link

GitHub: https://github.com/aidengseay/CS470FinalProject

Main Program File Details

Below is the Jupyter Notebook that does the following:

- Retrieve and split data into training and testing (split training into training and validation)
- K-Nearest Neighbors (KNN)
- Naive Bayes
- Logistic Regression

Details of each algorithm are found in the Utilities folder on GitHub.

Email Classifier - CS470 Final Project

Aiden Seay Spring 2024

Set Up Program

Import Necessary Libraries

In [1]:

```
# IMPORT FUNCTION UTILITIES
from Utilities.KNN import KNNClass
from Utilities.LogisticRegression import LogisticRegressionClass
from Utilities.NaiveBayes import NaiveBayesClass
import Utilities.SplitDataset as GetData
import Utilities.Analysis as Analysis
import warnings

# CONSTANTS
TARGET = 1
FEATURE = 0
```

Load Email Spam Data

You can find the data set here. Split the dataset into training, test and evaluation sub categories.

In [2]:

```
# read dataset into pandas df
df = GetData.read_dataset("./Data/spambase.csv")

# split the dataset (refer to SplitDataset for fold data structure)
(train_data, validation_data, X_test, y_test, X_train,
                                            y_train) = GetData.split_dataset(df)

# append all results from each algorithm here
results = []
```

Test Algorithms

All algorithm implementation can be found in the Utilities folder.

Naive Bayes

In [3]:

```
# initialize analysis data lists
v_acc_list = []
v_fp_list = []
v_tp_list = []
v_auc_list = []

t_acc_list = []
t_fp_list = []
t_tp_list = []
t_auc_list = []

# run naive bayes algorithm across 5 folds
for fold in range(len(train_data)):

    # initialize the class
    naive_bayes = NaiveBayesClass(train_data[fold], validation_data[fold],
                                                   X_test, y_test)

    # train the model
    (spam_prop, non_spam_prop, spam_word_freq,
                                 non_spam_word_freq) = naive_bayes.learn()

    # evaluate the model with validation set
    v_calc_result, v_true_result = naive_bayes.evaluate(spam_prop,
                             non_spam_prop, spam_word_freq, non_spam_word_freq,
                                   naive_bayes.validation_fold[FEATURE],
                                   naive_bayes.validation_fold[TARGET])

    # evaluate model with test set
    t_calc_result, t_true_result = naive_bayes.evaluate(spam_prop,
                             non_spam_prop, spam_word_freq, non_spam_word_freq,
                                  naive_bayes.X_test, naive_bayes.y_test)


    # analyze the results (validation)
    acc, fp, tp, auc = Analysis.analyze_results(v_calc_result, v_true_result)
    v_acc_list.append(acc)
    v_fp_list.append(fp)
    v_tp_list.append(tp)
    v_auc_list.append(tp)

    # analyze the results (test)
    acc, fp, tp, auc = Analysis.analyze_results(t_calc_result, t_true_result)
    t_acc_list.append(acc)
    t_fp_list.append(fp)
    t_tp_list.append(tp)
    t_auc_list.append(tp)

# get the average for the final results (test and validation)
v_acc_avg, v_fp_avg, v_tp_avg, v_auc_avg = Analysis.average_stats(v_acc_list,
                                        v_fp_list, v_tp_list, v_auc_list)

t_acc_avg, t_fp_avg, t_tp_avg, t_auc_avg = Analysis.average_stats(t_acc_list,
                                        t_fp_list, t_tp_list, t_auc_list)

# append test and validation results
results.append((("Naive Bayes Algorithm - Validation AVG", v_acc_avg, v_fp_avg,
                                                    v_tp_avg, v_auc_avg),
                (("Naive Bayes Algorithm - Test AVG", t_acc_avg, t_fp_avg,
                                                  t_tp_avg, t_auc_avg))))
```

The cell above executes the Naive Bayes algorithm. The algorithm is trained with the training data and is tested with a validation and test dataset. Naive Bayes works by calculating the frequency of words appearing for each email.

K-Nearest Neighbors (KNN)

In [4]:

```
# initialize analysis data lists
v_acc_list = []
v_fp_list = []
v_tp_list = []
v_auc_list = []

t_acc_list = []
t_fp_list = []
t_tp_list = []
t_auc_list = []

# run knn algorithm across 5 folds
for fold in range(len(train_data)):

    # initialize knn class
    knn = KNNClass(train_data[fold], validation_data[fold], X_test, y_test)

    # no training

    # evaluate model with validation set
    v_calc_result, v_true_result = knn.evaluate(knn.validation_fold[FEATURE],
                                                knn.validation_fold[TARGET])

    # evaluate mode with test set
    t_calc_result, t_true_result = knn.evaluate(knn.X_test, knn.y_test)

    # analyze validation results
    acc, fp, tp, auc = Analysis.analyze_results(v_calc_result, v_true_result)
    v_acc_list.append(acc)
    v_fp_list.append(fp)
    v_tp_list.append(tp)
    v_auc_list.append(tp)

    # analyze test results
    acc, fp, tp, auc = Analysis.analyze_results(t_calc_result, t_true_result)
    t_acc_list.append(acc)
    t_fp_list.append(fp)
    t_tp_list.append(tp)
    t_auc_list.append(tp)

# get the average for the final results (test and validation)
v_acc_avg, v_fp_avg, v_tp_avg, v_auc_avg = Analysis.average_stats(v_acc_list,
                                               v_fp_list, v_tp_list, v_auc_list)

t_acc_avg, t_fp_avg, t_tp_avg, t_auc_avg = Analysis.average_stats(t_acc_list,
                                               t_fp_list, t_tp_list, t_auc_list)

# append test and validation results
results.append((("KNN Algorithm - Validation AVG", v_acc_avg, v_fp_avg,
                                                         v_tp_avg, v_auc_avg),
                ("KNN Algorithm - Test AVG", t_acc_avg, t_fp_avg, t_tp_avg,
                                                                 t_auc_avg)))
```

The cell above executes the K-Nearest Neighbor (KNN) algorithm. There is no training for this algorithm. It starts by calculating the closest neighbors and determines the classification based on the k closest neighbors.

Logistic Regression

```
In [5]:
# suppress overflow warnings
warnings.filterwarnings('ignore')

# initialize analysis data lists
v_acc_list = []
v_fp_list = []
v_tp_list = []
v_auc_list = []

t_acc_list = []
t_fp_list = []
t_tp_list = []
t_auc_list = []

# run logistic regression algorithm across 5 folds
for fold in range(len(train_data)):
    log_reg = LogisticRegressionClass(train_data[fold], validation_data[fold],
                                                                X_test, y_test)

    # train the logistic regression model
    log_reg.learn()

    # evaluate model with validation set
    v_calc_result, v_true_result = log_reg.evaluate(
              log_reg.validation_fold[FEATURE], log_reg.validation_fold[TARGET])

    # evaluate mode with test set
    t_calc_result, t_true_result = log_reg.evaluate(log_reg.X_test,
                                                                log_reg.y_test)

    # analyze validation results
    acc, fp, tp, auc = Analysis.analyze_results(v_calc_result, v_true_result)
    v_acc_list.append(acc)
    v_fp_list.append(fp)
    v_tp_list.append(tp)
    v_auc_list.append(tp)

    # analyze test results
    acc, fp, tp, auc = Analysis.analyze_results(t_calc_result, t_true_result)
    t_acc_list.append(acc)
    t_fp_list.append(fp)
    t_tp_list.append(tp)
    t_auc_list.append(tp)

# get the average for the final results (test and validation)
v_acc_avg, v_fp_avg, v_tp_avg, v_auc_avg = Analysis.average_stats(v_acc_list,
                                               v_fp_list, v_tp_list, v_auc_list)

t_acc_avg, t_fp_avg, t_tp_avg, t_auc_avg = Analysis.average_stats(t_acc_list,
                                               t_fp_list, t_tp_list, t_auc_list)

# append test and validation results
results.append((("LR Algorithm - Validation AVG", v_acc_avg, v_fp_avg,
                                                           v_tp_avg, v_auc_avg),
                (("LR Algorithm - Test AVG", t_acc_avg, t_fp_avg, t_tp_avg,
                                                                  t_auc_avg))))
```

The cell above executes the logistic regression algorithm. First, it has to find the best curve to fit the training points. This is done in the learning function. After the curve is fitted everything above 0.5 is classified as spam and everything less than 0.5 is classified as non-spam.

Analyze Performance

Measure performance by:

- Accuracy (ACC)
- False Positive (FP)
- True Positive (TP)
- Area Under ROC Curve (AUC)

In [6]:

```
print("Algorithm Performance Analysis")
print("==============================\n")

for algorithm in results:

    for test_set in algorithm:

        print(test_set[0] + "\n" + "-" * len(test_set[0]))
        print(f"Accuracy:        {test_set[1]}")
        print(f"False Positive:  {test_set[2]}")
        print(f"True Positive:   {test_set[3]}")
        print(f"Area Under Curve: {test_set[4]}\n")
```

```
Algorithm Performance Analysis
==============================

Naive Bayes Algorithm - Validation AVG
--------------------------------------
Accuracy:         0.8657608695652174
False Positive:   0.23978150803391038
True Positive:    0.9577895163845683
Area Under Curve: 0.9577895163845683

Naive Bayes Algorithm - Test AVG
--------------------------------
Accuracy:         0.8603691639522258
False Positive:   0.25697255073954317
True Positive:    0.9563342318059299
Area Under Curve: 0.9563342318059299

KNN Algorithm - Validation AVG
------------------------------
Accuracy:         0.8078804347826087
False Positive:   0.17111735141914483
True Positive:    0.7364606402442577
Area Under Curve: 0.7364606402442577

KNN Algorithm - Test AVG
------------------------
Accuracy:         0.7854505971769815
False Positive:   0.21686910114259667
True Positive:    0.7315363881401618
Area Under Curve: 0.7315363881401618

LR Algorithm - Validation AVG
-----------------------------
Accuracy:         0.7603260869565217
False Positive:   0.28830592341197464
True Positive:    0.7288369167899997
Area Under Curve: 0.7288369167899997

LR Algorithm - Test AVG
-----------------------
Accuracy:         0.7300760043431053
False Positive:   0.34071699263089966
True Positive:    0.7029649595687332
Area Under Curve: 0.7029649595687332
```

Each algorithm is tested on the same training data, validation data, and training data. The algorithms are tested 5 times on 5 different training and validation data sets. Above is the average of results from each validation fold and testing set. The testing set is the same for each fold. Additionally, the accuracy is above 70% for each validation and test on each algorithm.