Seniz Ozdemir

UCID: so9

Email: so9@njit.edu

11/24/2024

Professor: Yasser Abdullah

CS 634-101 Data Mining

Final Project Report

Usage Notes

In order for this project to function as intended, the installation of the following libraries is required: pandas, numpy, scikit-learn, ucimlrepo, tensorflow. They may be installed with the following commands:

pip install pandas

pip install numpy

pip install scikit-learn

pip install tensorflow

pip install ucimlrepo

The version of Python used for this project is version Python 3.12.6.

The data used in this project is imported from ucimlrepo when the program is run. However, a copy of the data is included, and was retrieved from the following webpage: https://archive.ics.uci.edu/dataset/94/spambase.

This project's Python code, a Jupyter Notebook file of the project, the data set used, and this report are also available at: https://github.com/atwoodmachine/ozdemir_seniz_finaltermproj

Implementation Details

This project implements three machine learning classification algorithms trained on the Spambase data set-Random Forest, K Nearest Neighbors, and LSTM-with the purpose of classifying emails as Spam or Not Spam based on the frequency of certain words and characters as well as the length of capitalizations, as enumerated by the Spambase data set. None of the algorithms are purposefully optimized, but are implemented in their minimal functioning form. K-fold cross validation was performed on every algorithm for k = 10, with confusion scores calculated as well as AUC and Brier Loss scores calculated using Python libraries.

The calculation of confusion metrics is performed in the code below, with the calc_performance function taking in input of a confusion matrix (calculated using the

confusion_matrix function from the scikit-learn library later in the program) and calculating the following using the known formulas.

```
#Performance Calculations
def calc_performance(confusion_matrix):
    TP = confusion_matrix[0][0]
    FN = confusion_matrix[0][1]
    FP = confusion_matrix[1][0]
    TN = confusion_matrix[1][1]

    P = TP + FN
    N = TN + FP

    TPR = TP/P
    TNR = TN/N
    FPR = FP/N
    FNR = FN/P

    Precision = TP/(TP + FP)
    F1_measure = (2*TP)/(2*TP + FP + FN)
    Accuracy = (TP + TN)/(P + N)
    Error_rate = (FP + FN)/(P + N)

    BACC = (TPR + TNR)/2
    TSS = (TP/(TP+FN)) - (FP/(FP + TN))
    HSS = (2 * (TP * TN - FP * FN))/((TP + FN) * (FN + TN) + (TP + FP) * (FP + TN))

    return [TP, FN, FP, TN, TPR, TNR, FPR, FNR, Precision, F1_measure, Accuracy, Error_rate, BACC, TSS, HSS]
```

The number of True Positive, False Negative, True Positive, and False Positive classifications returned from the model are extracted from the confusion matrix calculated when each model is trained and tested. The number of positive and negative classifications total are also calculated for readability. These values are then used to calculate the True Positive Rate (sensitivity), True Negative Rate (specificity), False Positive Rate, and False Negative Rate. Additionally, the Precision, F1 Measure, Accuracy, Error Rate, Balanced Accuracy, True Skill Statistics, and Heidke Skill Score are calculated using the above values. All calculated performance metrics are returned by the function in a list.

Next is the implementation for the Random Forest Classifier.

```
#Random Forest
def rf_classifier(features_train, features_test, targets_train, targets_test):
    rf = RandomForestClassifier()
    rf.fit(features_train, targets_train)

    rf_prediction = rf.predict(features_test)

    conf_matrix = metrics.confusion_matrix(targets_test, rf_prediction)
    performance = calc_performance(conf_matrix)

    #calculate additional metrics
    brier = metrics.brier_score_loss(targets_test, rf.predict_proba(features_test)[:, 1])
    roc_auc = metrics.roc_auc_score(targets_test, rf.predict_proba(features_test)[:, 1])
    performance.append(brier)
    performance.append(roc_auc)
    return performance
```

The rf_classifier function takes a set of training data features and its respective set of training targets for the model to be trained on, as well as a set of testing features and their respective targets. The model is initialized as a Random Forest Classifier using the scikit-learn library. The model is trained on the training set and features, and then a set of predictions is calculated based on testing data. The confusion matrix is claculated using the predictions the Random Forest model calculated based on the testing data and the target testing data. Performance is calculated as previously described in the calc_performance function. Additionally, the scikit-learn library is used to calculate the Brier Score Loss as well as the ROC/AUC score. All performance metrics are returned by the function in a list.

The function for K Nearest Neighbors functions near identically, with a few adjustments for the model.

```
#KNN
def knn_classifier(features_train, features_test, targets_train, targets_test):

    knn = KNeighborsClassifier(n_neighbors=3)
    knn.fit(features_train, targets_train)
    knn_prediction = knn.predict(features_test)

    conf_matrix = metrics.confusion_matrix(targets_test, knn_prediction)
    performance = calc_performance(conf_matrix)

    #calculate additional metrics
    brier = metrics.brier_score_loss(targets_test, knn.predict_proba(features_test)[:, 1])
    roc_auc = metrics.roc_auc_score(targets_test, knn.predict_proba(features_test)[:, 1])
    performance.append(brier)
    performance.append(roc_auc)
    return performance
```

The KNN model is initialized with the KNeighborsClassifier from the scikit-learn library, with an arbitrary value of n_neighbors=3. Then performance metrics are calculated as before.

The LSTM model is slightly more complex than the prior two models. The nature of the model and the implementation requires some data reshaping in order for the model to accept the training input. However, the general process of fitting the model to the data, getting predictions, and calculating performance metrics based on the predicted classes and the test data remains the same as the previous two classification algorithms.

```
#LSTM
def lstm_classifier(features_train, features_test, targets_train, targets_test):
    Xtrain, Xtest, ytrain, ytest = map(np.array, [features_train, features_test, targets_train, targets_test])
    shape = Xtrain.shape

    Xtrain_reshaped = Xtrain.reshape(len(Xtrain), shape[1], 1)
    Xtest_reshaped = Xtest.reshape(len(Xtest), shape[1], 1)

    lstm = Sequential()
    lstm.add(LSTM(10, activation='relu', input_shape=(57, 1), return_sequences=False))
    lstm.add(Dense(1, activation="sigmoid"))
    lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

    lstm.fit(Xtrain_reshaped, ytrain, validation_data=(Xtest_reshaped, ytest), epochs=10, batch_size=64, verbose=0)

    pred_prob = lstm.predict(Xtest_reshaped)
    y_pred = (pred_prob >= 0.5).astype(int)

    y_pred = y_pred.reshape(-1)
    ytest = ytest.reshape(-1)

    conf_matrix = metrics.confusion_matrix(ytest, y_pred)
    performance = calc_performance(conf_matrix)
    #calculate additional metrics
    brier = metrics.brier_score_loss(ytest, pred_prob)
    roc_auc = metrics.roc_auc_score(ytest, pred_prob)
    performance.append(brier)
    performance.append(roc_auc)

    return performance
```

The following code simply fetches the Spambase data set and splits the features from the classifications.

```
# fetch dataset
spambase = fetch_ucirepo(id=94) #94 for spam

# data (as pandas dataframes)
features = spambase.data.features #X
targets = spambase.data.targets #y
targets = np.ravel(targets)

metric_names = ['TP', 'FN', 'FP', 'TN', 'TPR', 'TNR', 'FPR',
                'FNR', 'Precision', 'F1_measure', 'Accuracy', 'Error_rate', 'BACC',
                'TSS', 'HSS', 'Brier Score', 'AUC']

kf = KFold(n_splits=10, shuffle=True, random_state=1)

all_rf_metrics = []
all_knn_metrics = []
all_lstm_metrics = []
```

Additionally, lists for all algorithms metrics are initialized, as well as the KFold validation for 10 folds.

The K-fold cross validation follows. Each classification algorithm is called for each iteration, with all three algorithms performance metrics being put into a Pandas Dataframe and then printed at the end of each iteration for comparison purposes.

```
for i, (train_index, test_index) in enumerate(kf.split(features), start=1):
    #Split training and test data sets
    X_train, X_test, y_train, y_test = train_test_split(features,
    targets, test_size=0.1, stratify=targets)

    #Normalize data
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    #Train models
    #Random Forest
    rf_performance = rf_classifier(X_train, X_test, y_train, y_test)
    #KNN
    knn_performance = knn_classifier(X_train, X_test, y_train, y_test)
    #LSTM
    lstm_perfomance = lstm_classifier(X_train, X_test, y_train, y_test)

    performance_metrics = pd.DataFrame([rf_performance, knn_performance, lstm_perfomance], columns=metric_names, index=['RF',
    'KNN', 'LSTM'])
    print("\n***Iteration {} Performance Metrics***" .format(i))
    print(performance_metrics)

    all_rf_metrics.append(rf_performance)
    all_knn_metrics.append(knn_performance)
    all_lstm_metrics.append(lstm_perfomance)
```

When all iterations are complete, the average performance of each algorithm is calculated and displayed as follows.

```
print("\n***Performance Summary for Individual Algorithms***\n")
metric_iter_names = ['iter1', 'iter2', 'iter3', 'iter4', 'iter5', 'iter6', 'iter7', 'iter8', 'iter9', 'iter10']

all_rf_metrics_df = pd.DataFrame(all_rf_metrics, columns=metric_names, index=metric_iter_names)
print("\n***All Metrics for All Interations: Random Forest***")
print(all_rf_metrics_df)

all_knn_metrics_df = pd.DataFrame(all_knn_metrics, columns=metric_names, index=metric_iter_names)
print("\n***All Metrics for All Interations: K Nearest Neighbors***")
print(all_knn_metrics_df)

all_lstm_metrics_df = pd.DataFrame(all_lstm_metrics, columns=metric_names, index=metric_iter_names)
print("\n***All Metrics for All Interations: LSTM***")
print(all_lstm_metrics_df)

print("\n***Average Performance of All Algorithms***\n")
avg_rf = all_rf_metrics_df.mean()
avg_knn = all_knn_metrics_df.mean()
avg_lstm = all_lstm_metrics_df.mean()

avg_all = pd.DataFrame({'RF': avg_rf, 'KNN': avg_knn, 'LSTM': avg_lstm}, index=metric_names)
print(avg_all)
```

Output

```
***Iteration 1 Performance Metrics***
      TP  FN  FP   TN       TPR       TNR       FPR       FNR  Precision  F1_measure  Accuracy  Error_rate      BACC       TSS       HSS  Brier Score       AUC
RF   273   6  12  170  0.978495  0.934066  0.065934  0.021505   0.957895    0.968085  0.960954    0.039046  0.956280  0.912561  0.917820     0.033908  0.992841
KNN  263  16  22  160  0.942652  0.879121  0.120879  0.057348   0.922807    0.932624  0.917570    0.082430  0.910887  0.821773  0.826510     0.069414  0.947960
LSTM 244  35  51  131  0.874552  0.719780  0.280220  0.125448   0.827119    0.850174  0.813449    0.186551  0.797166  0.594332  0.603556     0.133399  0.892749
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step

***Iteration 2 Performance Metrics***
      TP  FN  FP   TN       TPR       TNR       FPR       FNR  Precision  F1_measure  Accuracy  Error_rate      BACC       TSS       HSS  Brier Score       AUC
RF   271   8  15  167  0.971326  0.917582  0.082418  0.028674   0.947552    0.959292  0.950108    0.049892  0.944454  0.888909  0.894892     0.040186  0.985506
KNN  252  27  28  154  0.903226  0.846154  0.153846  0.096774   0.900000    0.901610  0.880694    0.119306  0.874690  0.749380  0.750096     0.087973  0.936311
LSTM 258  21  44  138  0.924731  0.758242  0.241758  0.075269   0.854305    0.888124  0.859002    0.140998  0.841486  0.682973  0.698314     0.108119  0.913565
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step

***Iteration 3 Performance Metrics***
      TP  FN  FP   TN       TPR       TNR       FPR       FNR  Precision  F1_measure  Accuracy  Error_rate      BACC       TSS       HSS  Brier Score       AUC
RF   270   9  12  170  0.967742  0.934066  0.065934  0.032258   0.957447    0.962567  0.954447    0.045553  0.950904  0.901808  0.904399     0.035670  0.992201
KNN  261  18  23  159  0.935484  0.873626  0.126374  0.064516   0.919014    0.927176  0.911063    0.088937  0.904555  0.809110  0.812993     0.076886  0.939472
LSTM 263  16 134   48  0.942652  0.263736  0.736264  0.057348   0.662469    0.778107  0.674620    0.325380  0.603194  0.206389  0.232605     0.198512  0.821635
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step

***Iteration 4 Performance Metrics***
      TP  FN  FP   TN       TPR       TNR       FPR       FNR  Precision  F1_measure  Accuracy  Error_rate      BACC       TSS       HSS  Brier Score       AUC
RF   271   8  11  171  0.971326  0.939560  0.060440  0.028674   0.960993    0.966132  0.958785    0.041215  0.955443  0.910887  0.913504     0.034267  0.994555
KNN  267  12  24  158  0.956989  0.868132  0.131868  0.043011   0.917526    0.936842  0.921909    0.078091  0.912561  0.825121  0.834688     0.065076  0.949033
LSTM 254  25  43  139  0.910394  0.763736  0.236264  0.089606   0.855219    0.881944  0.852495    0.147505  0.837065  0.674131  0.685923     0.100185  0.926799
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step

***Iteration 5 Performance Metrics***
      TP  FN  FP   TN       TPR       TNR       FPR       FNR  Precision  F1_measure  Accuracy  Error_rate      BACC       TSS       HSS  Brier Score       AUC
RF   265  14   9  173  0.949821  0.950549  0.049451  0.050179   0.967153    0.958409  0.950108    0.049892  0.950185  0.900370  0.896091     0.036907  0.988578
KNN  257  22  16  166  0.921147  0.912088  0.087912  0.078853   0.941392    0.931159  0.917570    0.082430  0.916617  0.833235  0.828487     0.069173  0.948935
LSTM 257  22  29  153  0.921147  0.840659  0.159341  0.078853   0.898601    0.909735  0.889371    0.110629  0.880903  0.761806  0.766934     0.085328  0.947792
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step

***Iteration 6 Performance Metrics***
      TP  FN  FP   TN       TPR       TNR       FPR       FNR  Precision  F1_measure  Accuracy  Error_rate      BACC       TSS       HSS  Brier Score       AUC
RF   271   8  16  166  0.971326  0.912088  0.087912  0.028674   0.944251    0.957597  0.947939    0.052061  0.941707  0.883414  0.890216     0.042376  0.984678
KNN  265  14  18  164  0.949821  0.901099  0.098901  0.050179   0.936396    0.943060  0.930586    0.069414  0.925460  0.850920  0.854183     0.058809  0.963370
LSTM 261  18 133   49  0.935484  0.269231  0.730769  0.064516   0.662437    0.775632  0.672451    0.327549  0.602357  0.204715  0.229975     0.194028  0.809406
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step

***Iteration 7 Performance Metrics***
      TP  FN  FP   TN       TPR       TNR       FPR       FNR  Precision  F1_measure  Accuracy  Error_rate      BACC       TSS       HSS  Brier Score       AUC
RF   275   4   9  173  0.985663  0.950549  0.049451  0.014337   0.968310    0.976909  0.971800    0.028200  0.968106  0.936213  0.940705     0.026667  0.996691
KNN  262  17  16  166  0.939068  0.912088  0.087912  0.060932   0.942446    0.940754  0.928416    0.071584  0.925578  0.851156  0.850344     0.059773  0.960111
LSTM 250  29  34  148  0.896057  0.813187  0.186813  0.103943   0.880282    0.888099  0.863341    0.136659  0.854622  0.709244  0.712648     0.100583  0.934814
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step

***Iteration 8 Performance Metrics***
      TP  FN  FP   TN       TPR       TNR       FPR       FNR  Precision  F1_measure  Accuracy  Error_rate      BACC       TSS       HSS  Brier Score       AUC
RF   272   7  14  168  0.974910  0.923077  0.076923  0.025090   0.951049    0.962832  0.954447    0.045553  0.948994  0.897987  0.904032     0.036075  0.988469
KNN  254  25  18  164  0.910394  0.901099  0.098901  0.089606   0.933824    0.921960  0.906725    0.093275  0.905747  0.811493  0.806104     0.074958  0.943982
LSTM 279   0 175    7  1.000000  0.038462  0.961538  0.000000   0.614537    0.761255  0.620390    0.379610  0.519231  0.038462  0.046181     0.219157  0.730513
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step

***Iteration 9 Performance Metrics***
      TP  FN  FP   TN       TPR       TNR       FPR       FNR  Precision  F1_measure  Accuracy  Error_rate      BACC       TSS       HSS  Brier Score       AUC
RF   268  11  10  172  0.960573  0.945055  0.054945  0.039427   0.964029    0.962298  0.954447    0.045553  0.952814  0.905628  0.904764     0.034912  0.990882
KNN  262  17  23  159  0.939068  0.873626  0.126374  0.060932   0.919298    0.929078  0.913232    0.086768  0.906347  0.812694  0.817379     0.069655  0.946355
LSTM 252  27 104   78  0.903226  0.428571  0.571429  0.096774   0.707865    0.793701  0.715835    0.284165  0.665899  0.331797  0.358137     0.179015  0.827110
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step

***Iteration 10 Performance Metrics***
      TP  FN  FP   TN       TPR       TNR       FPR       FNR  Precision  F1_measure  Accuracy  Error_rate      BACC       TSS       HSS  Brier Score       AUC
RF   272   7  14  168  0.974910  0.923077  0.076923  0.025090   0.951049    0.962832  0.954447    0.045553  0.948994  0.897987  0.904032     0.039887  0.989907
KNN  266  13  27  155  0.953405  0.851648  0.148352  0.046595   0.907850    0.930070  0.913232    0.086768  0.902527  0.805053  0.815964     0.067727  0.954340
LSTM 265  14 109   73  0.949821  0.401099  0.598901  0.050179   0.708556    0.811639  0.733189    0.266811  0.675460  0.350920  0.385939     0.176624  0.857182
```

Performance Summary for Individual Algorithms

All Metrics for All Interations: Random Forest

	TP	FN	FP	TN	TPR	TNR	FPR	FNR	Precision	F1_measure	Accuracy	Error_rate	BACC	TSS	HSS	Brier Score	AUC
iter1	273	6	12	170	0.978495	0.934066	0.065934	0.021505	0.957895	0.968085	0.960954	0.039046	0.956280	0.912561	0.917820	0.033908	0.992841
iter2	271	8	15	167	0.971326	0.917582	0.082418	0.028674	0.947552	0.959292	0.950108	0.049892	0.944454	0.888909	0.894892	0.040186	0.985506
iter3	270	9	12	170	0.967742	0.934066	0.065934	0.032258	0.957447	0.962567	0.954447	0.045553	0.950904	0.901808	0.904399	0.035670	0.992201
iter4	271	8	11	171	0.971326	0.939560	0.060440	0.028674	0.960993	0.966132	0.958785	0.041215	0.955443	0.910887	0.913504	0.034267	0.994555
iter5	265	14	9	173	0.949821	0.950549	0.049451	0.050179	0.967153	0.958409	0.950108	0.049892	0.950185	0.900370	0.896091	0.036907	0.988578
iter6	271	8	16	166	0.971326	0.912088	0.087912	0.028674	0.944251	0.957597	0.947939	0.052061	0.941707	0.883414	0.890216	0.042376	0.984678
iter7	275	4	9	173	0.985663	0.950549	0.049451	0.014337	0.968310	0.976909	0.971800	0.028200	0.968106	0.936213	0.940705	0.026667	0.996691
iter8	272	7	14	168	0.974910	0.923077	0.076923	0.025090	0.951049	0.962832	0.954447	0.045553	0.948994	0.897987	0.904032	0.036075	0.988469
iter9	268	11	10	172	0.960573	0.945055	0.054945	0.039427	0.964029	0.962298	0.954447	0.045553	0.952814	0.905628	0.904764	0.034912	0.990882
iter10	272	7	14	168	0.974910	0.923077	0.076923	0.025090	0.951049	0.962832	0.954447	0.045553	0.948994	0.897987	0.904032	0.039887	0.989907

All Metrics for All Interations: K Nearest Neighbors

	TP	FN	FP	TN	TPR	TNR	FPR	FNR	Precision	F1_measure	Accuracy	Error_rate	BACC	TSS	HSS	Brier Score	AUC
iter1	263	16	22	160	0.942652	0.879121	0.120879	0.057348	0.922807	0.932624	0.917570	0.082430	0.910887	0.821773	0.826510	0.069414	0.947960
iter2	252	27	28	154	0.903226	0.846154	0.153846	0.096774	0.900000	0.901610	0.880694	0.119306	0.874690	0.749380	0.750096	0.087973	0.936311
iter3	261	18	23	159	0.935484	0.873626	0.126374	0.064516	0.919014	0.927176	0.911063	0.088937	0.904555	0.809110	0.812993	0.076886	0.939472
iter4	267	12	24	158	0.956989	0.868132	0.131868	0.043011	0.917526	0.936842	0.921909	0.078091	0.912561	0.825121	0.834688	0.065076	0.949033
iter5	257	22	16	166	0.921147	0.912088	0.087912	0.078853	0.941392	0.931159	0.917570	0.082430	0.916617	0.833235	0.828487	0.069173	0.948935
iter6	265	14	18	164	0.949821	0.901099	0.098901	0.050179	0.936396	0.943060	0.930586	0.069414	0.925460	0.850920	0.854183	0.058809	0.963370
iter7	262	17	16	166	0.939068	0.912088	0.087912	0.060932	0.942446	0.940754	0.928416	0.071584	0.925578	0.851156	0.850344	0.059773	0.960111
iter8	254	25	18	164	0.910394	0.901099	0.098901	0.089606	0.933824	0.921960	0.906725	0.093275	0.905747	0.811493	0.806104	0.074958	0.943982
iter9	262	17	23	159	0.939068	0.873626	0.126374	0.060932	0.919298	0.929078	0.913232	0.086768	0.906347	0.812694	0.817379	0.069655	0.946355
iter10	266	13	27	155	0.953405	0.851648	0.148352	0.046595	0.907850	0.930070	0.913232	0.086768	0.902527	0.805053	0.815964	0.067727	0.954340

All Metrics for All Interations: LSTM

	TP	FN	FP	TN	TPR	TNR	FPR	FNR	Precision	F1_measure	Accuracy	Error_rate	BACC	TSS	HSS	Brier Score	AUC
iter1	244	35	51	131	0.874552	0.719780	0.280220	0.125448	0.827119	0.850174	0.813449	0.186551	0.797166	0.594332	0.603556	0.133399	0.892749
iter2	258	21	44	138	0.924731	0.758242	0.241758	0.075269	0.854305	0.888124	0.859002	0.140998	0.841486	0.682973	0.698314	0.108119	0.913565
iter3	263	16	134	48	0.942652	0.263736	0.736264	0.057348	0.662469	0.778107	0.674620	0.325380	0.603194	0.206389	0.232605	0.198512	0.821635
iter4	254	25	43	139	0.910394	0.763736	0.236264	0.089606	0.855219	0.881944	0.852495	0.147505	0.837065	0.674131	0.685923	0.100185	0.926799
iter5	257	22	29	153	0.921147	0.840659	0.159341	0.078853	0.898601	0.909735	0.889371	0.110629	0.880903	0.761806	0.766934	0.085328	0.947792
iter6	261	18	133	49	0.935484	0.269231	0.730769	0.064516	0.662437	0.775632	0.672451	0.327549	0.602357	0.204715	0.229975	0.194028	0.809406
iter7	250	29	34	148	0.896057	0.813187	0.186813	0.103943	0.880282	0.888099	0.863341	0.136659	0.854622	0.709244	0.712648	0.100583	0.934814
iter8	279	0	175	7	1.000000	0.038462	0.961538	0.000000	0.614537	0.761255	0.620390	0.379610	0.519231	0.038462	0.046181	0.219157	0.730513
iter9	252	27	104	78	0.903226	0.428571	0.571429	0.096774	0.707865	0.793701	0.715835	0.284165	0.665899	0.331797	0.358137	0.179015	0.827110
iter10	265	14	109	73	0.949821	0.401099	0.598901	0.050179	0.708556	0.811639	0.733189	0.266811	0.675460	0.350920	0.385939	0.176624	0.857182

Average Performance of All Algorithms

	RF	KNN	LSTM
TP	270.800000	260.900000	258.300000
FN	8.200000	18.100000	20.700000
FP	12.200000	21.500000	85.600000
TN	169.800000	160.500000	96.400000
TPR	0.970609	0.935125	0.925806
TNR	0.932967	0.881868	0.529670
FPR	0.067033	0.118132	0.470330
FNR	0.029391	0.064875	0.074194
Precision	0.956973	0.924055	0.767139
F1_measure	0.963695	0.929433	0.833841
Accuracy	0.955748	0.914100	0.769414
Error_rate	0.044252	0.085900	0.230586
BACC	0.951788	0.908497	0.727738
TSS	0.903576	0.816994	0.455477
HSS	0.907046	0.819675	0.472021
Brier Score	0.036085	0.069945	0.149495
AUC	0.990431	0.948987	0.866157

PS C:\Users\ozdem\Desktop\CS 634\Final project>

Discussion

The purpose of this project was to compare the performance of each algorithm according to the calculated metrics. None of the algorithms were specifically optimized, and all were trained and tested on the same dataset. The Spambase data set contains numerical frequencies of the occurrences of certain words and sequences in a set of emails labelled as Spam and Not Spam.

The most helpful way to compare each algorithm's performance is to compare the average performance of each algorithm over each of the 10 iterations they were trained and tested for. In terms of raw numbers of true positive, false negative, false positive, and true negative outcomes, the Random Forest algorithm predicted the most true positives and false negatives and the least false negatives and false positives on average. The LSTM algorithm predicted the most false positives and least true negatives. To contextualize these numbers, the true positive rate, true negative rate, false positive rate, and false negative rate will provide a deeper understanding into the algorithms' comparative performance.

The Random Forest algorithm has the highest true positive rate on average at about 97%, compared to about 94% for KNN and 93% for LSTM. With this metric, each algorithm performs similarly. However, LSTM has a 53% true negative rate, which is much smaller than 93% for Random Forest and 88% for KNN. LSTM also has a 47% false positive rate, much higher than Random Forest at 7% and KNN at 12%. In these regards, LSTM is the weakest algorithm. All algorithms have a similar small false negative rate, with LSTM having the largest at about 7%.

Random Forest and KNN have high precision scores on average while LSTM scored lower, at about 77% compared to 96% and 92% respectively. Overall, these performance patterns hold, with Random Forest having the best accuracy, TSS, BACC, HSS and AUC, and the lowest error rate and Brier score. KNN ranks second in these metrics, and LSTM is last, which suggests LSTM is the weakest of the algorithms, Random Forest is the strongest, and KNN is similarly effective.

The performance of these algorithms can be explained by the choice of dataset. Random Forest classification is known to be incredibly strong when it comes to text classification, which resulted in its performance classifying emails as spam or not being the highest of the three algorithms. The Random Forest algorithm also avoids overfitting by using multiple decision trees. LSTM's poor performance may be explained by the lack of optimization performed in choosing its various layer parameters, so it may suffer from over or under-fitting. It is also trained for a limited number of epochs for the purposes of this project to maintain a low runtime, so training for a longer amount of time will likely increase the algorithm's effectiveness. There is also a possibility that imbalanced data may have had an impact on the LSTM model's efficacy along with the low number of epochs chosen. With some optimization, it is likely that all models would perform with similar effectiveness.