Name: Akhil Sreedhara
NJIT UCID: as3638
Email: as3638@njit.edu
Date: Nov 13, 2024
Professor: Dr. Yasser Abduallah

GITHUB REPOSITORY: https://github.com/svakhil00/DataMiningFinal
*I had an existing github account so I just used that

Introduction
My code uses 3 different machine learning models to classify different flowers in the iris dataset. I first train Random Forest, Support Vector Machine, and LSTM on the dataset. I then make the models predict all the classes in the test set. Using the predictions and actual labels, I calculate various metrics(full list can be found below) and print the results. I repeat this 10 times to perform 10-fold cross-validation and print the overall performances of the models.

Installation: Please check the requirements file to see instructions on how to setup the project.

Algorithms Used:

1. Random Forest
2. Support Vector Machine
3. LSTM

Dataset: https://archive.ics.uci.edu/dataset/53/iris
I used the Iris dataset to categorize flowers. There are normally 3 classes of flowers, but I trimmed the data so that there are only 2. In my code, instead of loading the dataset from a csv, i pull the data through an api call. I still include the dataset in my project.

Metrics Calculated:

1. **True Positive** -
   model correctly predicts positive class
2. **True Negative** -
   model correctly predicts negative class
3. **False Positive** -
   model incorrectly predicts positive class
4. **False Negative** -
   model incorrectly predicts negative class
5. **True Positive Rate** -
   proportion of true positives that were correctly identified
6. **True Negative Rate** -
   proportion of negatives predicted negative
7. **False Positive Rate** -
   proportion of negatives predicted as positive
8. **False Negative Rate** -
   proportion of positives predicted as negative
9. **Recall** -
   same as tpr
10. **Precision** -
    quality of the positive prediction
11. **F1-Score** -
    harmonic recall of precision and recall
12. **Accuracy** -
    proportion of correctly predicted cases(positive and negative)
13. **Error Rate** -
    proportion of incorrectly predicted cases
14. **Specificity** -
    same as tnr
15. **Negative Predictive Value** -
    proportion of negative predictions that were correct
16. **False Discovery Rate** -
    proportion of positive predictions that were incorrect

17. **Balanced Accuracy** -
    more balanced when dataset is inbalanced
18. **True Skill Statistics** -
    measures how well the model predicts compared to randomly guessing
19. **Heidke Skill Score** -
    model's skill compared to random chance
20. **Brier Score** -
    (mean squared error) calculates the average error
21. **Brier Skill Score** -
    measure of how well a model performed compared to the baseline -1: worse 0: same 1: better

Code Walkthrough:

All required depenencies used by program. Installation instructions can be found in requirements file.

sklearn - used for 3 models, k-fold, getting tp, fp, fn, tp

numpy - math

pandas - data manipulation

matplotlib - graph

tensorflow - lstm

```
# imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler  # MinMaxScaler
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Input
```

Function that takes in the labels for the test set along with the predicted labels from each model. Calculates and returns all metrics explained above.

```
# Calculates all Metrics
def evaluatePerformance(yTest, yPredictions, yProbabilities=None):
    # Confusion Matrix
    tn, fp, fn, tp = confusion_matrix(yTest, yPredictions).ravel()
    # Positive
    p = tp + fn
    # Negative
    n = tn + fp
    # True Positive Rate
    tpr = tp / p
    # True Negative rate
    tnr = tn / n
    # False Positive Rate
    fpr = fp / n
    # False Negative Rate
    fnr = fn / p
    # Recall
    r = tp / p
    # Precision
    precision = tp / (tp + fp)
    # F1 Measure
    f1 = 2 * (precision * r) / (precision + r)
    # Accuracy
    acc = (tp + tn) / (p + n)
    # Error Rate
    e = (fp + fn) / (p + n)
    # Specificity
    spc = tn / (fp + tn)
    # Negative Predictive Value
    npv = tn / (tn + fn)
    # False Discovery Rate
    fdr = fp / (fp + tp)
    # Balanced Accuracy
```

```
    bacc = .5 * (tp / (tp + fn) + tn / (tn + fp))
    # True Skill Statistics
    tss = tp / (tp + fn) - fp / (fp + tn)
    # Heidke Skill Score
    hss = 2 * (tp * tn - fp * fn) / ((tp + fn) * (fn + tn) + (tp + fp) * (fp + tn))
    # Brier Score
    bs = -123456789
    if yProbabilities is not None:
        bs = np.mean((yTest - yProbabilities) ** 2)
    else:
        bs = np.mean((yTest - yPredictions) ** 2)
    # Brier Skill Score (The formula in the notes forgot to do "1 -" in the beginning)
    bss = 1 - (bs / np.mean((yTest - np.mean(yTest)) ** 2))

    return {
        'True Positive': tp,
        'True Negative': tn,
        'False Positive': fp,
        'False Negative': fn,
        'True Positive Rate': tpr,
        'True Negative Rate': tnr,
        'False Positive Rate': fpr,
        'False Negative Rate': fnr,
        'Recall': r,
        'Precision': precision,
        'F1-Score': f1,
        'Accuracy': acc,
        'Error Rate': e,
        'Specificity': spc,
        'Negative Predictive Value': npv,
        'False Discovery Rate': fdr,
        'Balanced Accuracy': bacc,
        'True Skill Statistics': tss,
        'Heidke Skill Score': hss,
        'Brier Score': bs,
        'Brier Skill Score': bss
    }
```

Function takes the metrics of each model in performance and the model name as a string. It then prints the data in a tabular format making it easier to read.

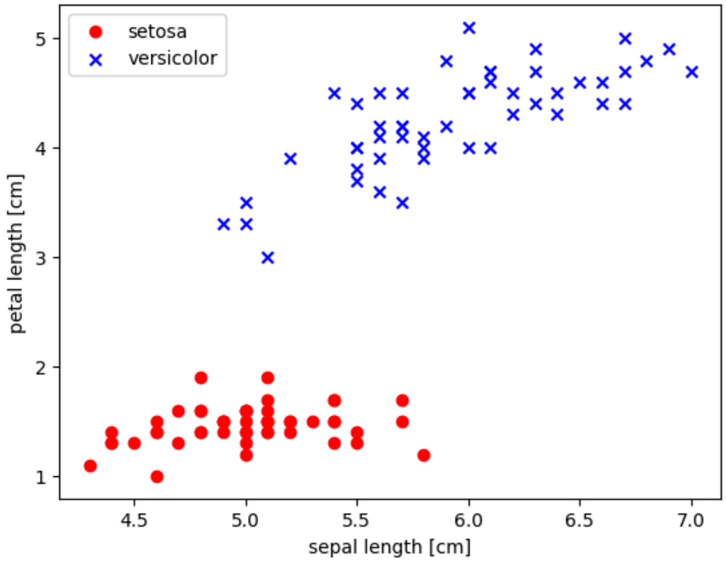
```
# Prints Metrics in Tabular Formant
def printMetrics(performance, modelName):
    df = pd.DataFrame(performance, index=[0])
    print(f'\n{modelName}')
    print(df.T.to_string(header=False))
```

Calls an api to get the dataset and plots the data using 2 of the features. I called an api instead of reading a csv because it was easier. The actual csv is also included in the project files.

```
df = pd.read_csv('sreedhara_akhil_iris.csv', header=None, encoding='utf-8')

y = df.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', 0, 1)
X = df.iloc[:100, [0, 1, 2, 3]].values

# Plot data
plt.scatter(X[:50, 0], X[:50, 2],
            color='red', marker='o', label='setosa')
plt.scatter(X[50:100, 0], X[50:100, 2],
            color='blue', marker='x', label='versicolor')
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```



Creating 10 folds of training and testing. Initializes arrays to store all metric information.

```
kf = KFold(n_splits=10, shuffle=True, random_state=42)

rfMetrics: list[dict] = []
svcMetrics: list[dict] = []
lstmMetrics: list[dict] = []
```

This loop iterates through the 10 folds. On each iteration it trains each of the models using the training set, predicts the classes for the test set, and calculates and prints the performance metrics for each model.

*** For a clearer view of the output please check outputs.txt

```
# K-Fold 10 times
for i, (train_index, test_index) in enumerate(kf.split(X), start=1):
    # Splitting the data
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Normalize the data
    scaler = MinMaxScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Reshape data for LSTM (samples, timesteps, features)
    T = 1  # Set timesteps to 1 for basic LSTM
    X_train_lstm = X_train_scaled.reshape((X_train_scaled.shape[0], T, X_train_scaled.shape[1]))
    X_test_lstm = X_test_scaled.reshape((X_test_scaled.shape[0], T, X_test_scaled.shape[1]))

    # Build the LSTM model
    lstm = Sequential()
    lstm.add(Input(shape=(T, X_train_scaled.shape[1])))
    lstm.add(LSTM(units=50, return_sequences=False))
    lstm.add(Dense(units=1, activation='sigmoid'))

    # Compile the model
    lstm.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    # Train the model
    lstm.fit(X_train_lstm, y_train, epochs=50, batch_size=8, validation_split=0.1, verbose=0)

    # Predict with LSTM
    lstmProbabilities = lstm.predict(X_test_lstm)
    lstmPredictions = (lstmProbabilities > 0.5).astype(int)

    # Random Forest Classifier
    rf = RandomForestClassifier()
    rf.fit(X_train, y_train)
    rfPredictions = rf.predict(X_test)

```

```
    # Support Vector Machine Classifier
    svc = LinearSVC(max_iter=10000)
    svc.fit(X_train, y_train)
    svcPredictions = svc.predict(X_test)

    # Calculate Performances
    performance = evaluatePerformance(y_test, rfPredictions)
    printMetrics(performance, 'Random Forest Metrics:')
    rfMetrics.append(performance)

    performance = evaluatePerformance(y_test, svcPredictions)
    printMetrics(performance, 'Support Vector Machine Metrics:')
    svcMetrics.append(performance)

    performance = evaluatePerformance(y_test, lstmPredictions, lstmProbabilities)
    printMetrics(performance, 'Long Short Term Memory Metrics:')
    lstmMetrics.append(performance)

```

```
Precision                  1.0
F1-Score                   1.0
Accuracy                   1.0
Error Rate                 0.0
Specificity                1.0
Negative Predictive Value  1.0
False Discovery Rate       0.0
Balanced Accuracy          1.0
True Skill Statistics      1.0
Heidke Skill Score         1.0
Brier Score                0.0
Brier Skill Score          1.0

Support Vector Machine Metrics:
True Positive              4.0
True Negative              6.0
False Positive             0.0
False Negative             0.0
True Positive Rate         1.0
True Negative Rate         1.0
False Positive Rate        0.0
False Negative Rate        0.0
Recall                     1.0
Precision                  1.0
F1-Score                   1.0
Accuracy                   1.0
Error Rate                 0.0
Specificity                1.0
Negative Predictive Value  1.0
False Discovery Rate       0.0
Balanced Accuracy          1.0
True Skill Statistics      1.0
Heidke Skill Score         1.0
Brier Score                0.0
Brier Skill Score          1.0

Long Short Term Memory Metrics:
True Positive              4.000000
True Negative              6.000000
False Positive             0.000000
False Negative             0.000000
True Positive Rate         1.000000
True Negative Rate         1.000000
False Positive Rate        0.000000
False Negative Rate        0.000000
Recall                     1.000000
Precision                  1.000000
F1-Score                   1.000000
Accuracy                   1.000000
Error Rate                 0.000000
Specificity                1.000000
Negative Predictive Value  1.000000
False Discovery Rate       0.000000
Balanced Accuracy          1.000000
True Skill Statistics      1.000000
Heidke Skill Score         1.000000
Brier Score                0.467046
Brier Skill Score         -0.946025
```

Once all 10 of the folds have been trained and tested, we find the overall performance of the models.

```
# Overall Performances
printMetrics(pd.DataFrame(rfMetrics).mean().to_dict(), 'Overall Random Forest Metrics:')
printMetrics(pd.DataFrame(svcMetrics).mean().to_dict(), 'Overall Support Vector Machine Metrics:')
printMetrics(pd.DataFrame(lstmMetrics).mean().to_dict(), 'Overall Long Short Term Memory Metrics:')
```

Analysis:

Overall, all 3 models performed perfectly on the test sets. They were able to correctly predict the class for every single piece of data in the dataset. I believe this to be because my dataset was pretty small easy to classify. Even though they performed perfectly, the lstm had a lower Brier Score and Brier Skill Score than the other two models. This is because lstm calculates probabilities instead of just a hard label. Using the probability we can tell how confident the model is in a prediction. Since the other 2 models are predicting with 100% confidence always, they give off the illusion of a greater brier score. In larger data sets, they would have a few wrong predictions and their true brier scores would show. The brier skill score for lstm came out to be negative which indicates that it performs worse than the baseline. I am curious to see what would happen on larger and more complicated datasets.

*** For a clearer view of the output please check outputs.txt