DATE: 11/24/2024 **NAME**: ANJALIGUPA RAGHAVENDRA

PROFESSOR: Dr. YASSER ABDUALLAH **UCID**: AR2729

CS 634 101 DATA MINING **EMAIL ID:** ar2729@njit.edu

FINAL TERM PROJECT REPORT

ABSTRACT

This report examines the performance of three different machine learning models; Random Forest Classifier (RFC), Support Vector Machine (SVM), and Bidirectional Long Short-Term Memory (BiLSTM) network on a binary classification task. These models were evaluated using 10-fold Stratified Cross-Validation to assess their effectiveness in predicting whether a sample is fraudulent or not, based on a given dataset. Metrics such as accuracy, precision, recall, F1-score, ROC AUC, Brier Score, and others were computed to evaluate model performance. Additionally, runtime analysis was conducted for each model, and their results were compared to provide insights into their efficiency and effectiveness.

INTRODUCTION

The rise of fraud detection systems in various industries, particularly in financial sectors, has led to the exploration of various machine learning techniques to improve prediction accuracy. This report focuses on comparing three popular machine learning models, Random Forest Classifier (RFC), Support Vector Machine (SVM), and Bidirectional Long Short-Term Memory (BiLSTM)—to assess their ability to detect fraudulent activity in a binary classification scenario. The dataset used contains labeled examples of fraudulent and non-fraudulent samples, and the goal is to classify new instances correctly based on historical data.

10-fold Stratified Cross-Validation ensures that each fold maintains the distribution of the classes in the target variable. Metrics such as True Positive Rate (TPR), False Positive Rate (FPR), Precision, Recall, F1-Score, Accuracy, and ROC AUC are used to assess the models' performance. Furthermore, the runtime for each model is tracked to provide insights into the computational efficiency of each algorithm.

CORE CONCEPTS AND PRINCIPLES

Project Workflow

The workflow involves loading and preprocessing transaction data, followed by the application of RFC, SVM, and BiLSTM to classify data. The models are trained and evaluated using cross-validation and performance metrics, ensuring robustness and consistency in results.

Data Loading and Preprocessing

Data is loaded from CSV files and preprocessed, including handling missing values and encoding categorical features. The dataset is split into training and testing sets to ensure proper model evaluation. Data normalization and feature scaling are performed as necessary to optimize model performance.

Random Forest Classifier (RFC)

The RFC is an ensemble learning method that constructs a multitude of decision trees during training and outputs the mode of the classes for classification tasks. It is known for its robustness and ability to handle high-dimensional datasets. The model is evaluated using various performance metrics, including confusion matrix, F1 score, accuracy, precision, recall, and ROC AUC. It operates by aggregating predictions from multiple decision trees, thus reducing overfitting and improving generalization.

Support Vector Machine (SVM)

SVM is a supervised learning algorithm that finds the optimal hyperplane separating different classes in a high-dimensional feature space. It works well in cases where the classes are not linearly separable by utilizing a kernel trick. Like the RFC, the SVM is evaluated using similar metrics, with particular attention to ROC AUC and Brier Score to assess its ability to predict probabilities effectively.

Bidirectional Long Short-Term Memory (BiLSTM)

BiLSTM is a type of Recurrent Neural Network (RNN) that is well-suited for sequential data. It improves upon traditional LSTM by processing data in both forward and backward directions, allowing the model to capture dependencies in both directions. For this classification task, the model is trained using binary cross-entropy as the loss function, and its performance is evaluated using the same suite of metrics as the other models.

Metrics

1. **Confusion Matrix**: A matrix used to evaluate the performance of classification algorithms. It displays the true positives, false positives, true negatives, and false negatives.
2. **True Positive Rate (TPR)** and **False Positive Rate (FPR)**: Measures the accuracy of detecting fraudulent (positive) samples and non-fraudulent (negative) samples.
3. **Precision and Recall**: Precision measures the accuracy of the positive predictions, while recall assesses the model's ability to identify all fraudulent cases.

4. **F1-Score**: The harmonic mean of precision and recall, offering a balanced view of performance.
5. **ROC AUC**: The area under the Receiver Operating Characteristic curve, representing the trade-off between the True Positive Rate and False Positive Rate.
6. **Brier Score**: A metric that measures the accuracy of probabilistic predictions, where a lower score indicates better performance.

10-Fold Cross-Validation

Stratified 10-fold cross-validation is employed to ensure that each fold maintains the same proportion of positive and negative cases in the target variable. This approach mitigates issues caused by class imbalances in the dataset and provides a more reliable estimate of model performance.

Results and Evaluation

The models are evaluated based on accuracy, precision, recall, F1 score, ROC AUC, and Brier score. The RFC generally performs well with high-dimensional data, while SVM is effective for non-linearly separable datasets. BiLSTM excels in sequential data tasks. Cross-validation results show consistent performance, with BiLSTM performing slightly better for time-series data, while RFC and SVM handle structured data effectively.

CONCLUSION

In this study, three distinct machine learning models RFC, SVM, and BiLSTM were evaluated for a binary classification task on a fraud detection dataset. Each model's performance was measured using several classification metrics, including precision, recall, F1-score, accuracy, ROC AUC, and Brier Score. Additionally, the runtime for each algorithm was computed to provide insights into their computational efficiency.

The Random Forest Classifier demonstrated a balanced performance across most metrics, achieving high accuracy and F1-scores, making it a solid choice for fraud detection tasks. The Support Vector Machine showed comparable results, especially in terms of ROC AUC and precision, though it had slightly longer runtimes. The Bidirectional LSTM model, while powerful in handling sequential dependencies, took considerably longer to train, though it showed strong performance in terms of recall and F1-score.

In conclusion, while the Random Forest Classifier provided the most balanced results in terms of performance and runtime, the choice of model would depend on the specific application requirements, including accuracy, interpretability, and computational constraints. Future research could explore hyperparameter tuning and ensemble methods to further improve the performance of these models in fraud detection systems.

SCREENSHOTS

Data Preprocessing and Cleaning

```python
# Data Preprocessing
X = data.drop(columns=['Unnamed: 0', 'Transaction ID', 'Customer ID', 'Is Fraudulent', 'IP Address', 'Shipping Address', 'Billing Address'])
y = data['Is Fraudulent']

# Handle dates
X['Transaction Date'] = pd.to_datetime(X['Transaction Date'])
X['Transaction Day'] = X['Transaction Date'].dt.day
X['Transaction Month'] = X['Transaction Date'].dt.month
X['Transaction Year'] = X['Transaction Date'].dt.year
X = X.drop(columns=['Transaction Date'])

# Encode categorical variables
le = LabelEncoder()
categorical_cols = ['Payment Method', 'Product Category', 'Customer Location', 'Device Used']
for col in categorical_cols:
    X[col] = le.fit_transform(X[col].astype(str))

# Handle missing values
imputer = SimpleImputer(strategy='mean')
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# Normalize numerical columns
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
```

Figure 1: Code snippet for cleaning the data

Defining the Models & 10-Fold Cross Validation

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

# Define models
models = {
    'Random Forest': RandomForestClassifier(random_state=42),
    'SVM': SVC(probability=True, random_state=42),
    'BiLSTM': Sequential([
        Bidirectional(LSTM(64, input_shape=(X.shape[1], 1))),
        Dense(1, activation='sigmoid')
    ])
}

results = {model_name: [] for model_name in models.keys()}
confusion_matrices = {model_name: np.zeros((2, 2)) for model_name in models.keys()}

for model_name, model in models.items():
    metrics_list = []
    total_time = 0

    for fold, (train_index, test_index) in enumerate(kf.split(X), 1):
        # Splitting the data
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        # For BiLSTM reshape input data to 3D [samples, timesteps, features]
        if model_name == 'BiLSTM':
            X_train = np.expand_dims(X_train.values, axis=2)
            X_test = np.expand_dims(X_test.values, axis=2)
            model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

        # Measure time taken for training and prediction
        start_time = time.time()

        if model_name == 'BiLSTM':
            model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)
            y_pred_prob = model.predict(X_test).ravel()
        else:
            model.fit(X_train, y_train)
            y_pred_prob = model.predict_proba(X_test)[:, 1]

        runtime = time.time() - start_time
        total_time += runtime
```

Figure 2: K-cross validation where K=10

Calculating Metrics

```
        # Calculate confusion matrix
        cm = confusion_matrix(y_test, y_pred)
        confusion_matrices[model_name] += cm

        # Calculate metrics
        tn, fp, fn, tp = cm.ravel()

        tpr = tp / (tp + fn)  # True Positive Rate (Recall)
        fpr = fp / (fp + tn)  # False Positive Rate
        tnr = tn / (tn + fp)  # True Negative Rate (Specificity)
        fnr = fn / (fn + tp)  # False Negative Rate

        precision = precision_score(y_test, y_pred)
        accuracy = accuracy_score(y_test, y_pred)
        recall = recall_score(y_test, y_pred)
        error_rate = 1 - accuracy
        f1 = f1_score(y_test, y_pred)

        bacc = (tpr + tnr) / 2  # Balanced Accuracy
        tss = tpr - fpr         # True Skill Statistic
        hss = (2 * (tp * tn - fp * fn)) / ((tp + fn) * (fn + tn) + (tp + fp) * (fp + tn))  # Heidke Skill Score

        bs = np.mean((y_pred_prob - y_test) ** 2)  # Brier Score
        bss = 1 - bs / np.var(y_test)             # Brier Skill Score

        roc_auc = roc_auc_score(y_test, y_pred_prob)

        metrics_list.append({
            'Fold': f'Fold {fold}',
            'TPR': tpr,
            'FPR': fpr,
            'TNR': tnr,
            'FNR': fnr,
            'Precision': precision,
            'Accuracy': accuracy,
            'Recall': recall,
            'Error Rate': error_rate,
            'F1 Score': f1,
            'BACC': bacc,
            'TSS': tss,
            'HSS': hss,
            'BS': bs,
            'BSS': bss,
            'ROC_AUC': roc_auc,
            'Runtime (seconds)': runtime
        })
```

Figure 3: Calculating the Metrics

Comparing Models

```
# Create a comparison table using PrettyTable
comparison_table = PrettyTable()
comparison_table.title = "Model Comparison"
comparison_table.field_names = ["Model", "Accuracy", "Precision", "Recall", "F1 Score", "ROC_AUC", "Runtime (seconds)"]
comparison_table.float_format = '.4'
comparison_table.align = 'r'
comparison_table.align["Model"] = 'l'  # Left-align the Model column

for model_name, metrics in results.items():
    avg_metrics = metrics[-1]  # The last row contains the average metrics
    comparison_table.add_row([
        model_name,
        avg_metrics['Accuracy'],
        avg_metrics['Precision'],
        avg_metrics['Recall'],
        avg_metrics['F1 Score'],
        avg_metrics['ROC_AUC'],
        avg_metrics['Runtime (seconds)']
    ])

# Print comparison table
print("\nModel Comparison:")
print(comparison_table)
```

Figure 4: Code Snippet Comparing the Models

Output

```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 51ms/step
```

Figure 5: Splitting data into 10 folds

Random Forest Results:

Random Forest Metrics:											
Metric	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10	Average
TPR	0.8036	0.8382	0.8197	0.8103	0.8305	0.7903	0.8197	0.8361	0.7627	0.7455	0.8057
FPR	0.4091	0.4375	0.2308	0.4048	0.3171	0.2368	0.3846	0.4103	0.3902	0.4000	0.3621
TNR	0.5909	0.5625	0.7692	0.5952	0.6829	0.7632	0.6154	0.5897	0.6098	0.6000	0.6379
FNR	0.1964	0.1618	0.1803	0.1897	0.1695	0.2097	0.1803	0.1639	0.2373	0.2545	0.1943
Precision	0.7143	0.8028	0.8475	0.7344	0.7903	0.8448	0.7692	0.7612	0.7377	0.6949	0.7697
Accuracy	0.7100	0.7500	0.8000	0.7200	0.7700	0.7800	0.7400	0.7400	0.7000	0.6800	0.7390
Recall	0.8036	0.8382	0.8197	0.8103	0.8305	0.7903	0.8197	0.8361	0.7627	0.7455	0.8057
Error Rate	0.2900	0.2500	0.2000	0.2800	0.2300	0.2200	0.2600	0.2600	0.3000	0.3200	0.2610
F1 Score	0.7563	0.8201	0.8333	0.7705	0.8099	0.8167	0.7937	0.7969	0.7500	0.7193	0.7867
BACC	0.6972	0.7004	0.7945	0.7028	0.7567	0.7767	0.7175	0.7129	0.6862	0.6727	0.7218
TSS	0.3945	0.4007	0.5889	0.4056	0.5134	0.5535	0.4351	0.4258	0.3725	0.3455	0.4435
HSS	0.4013	0.4109	0.5835	0.4137	0.5192	0.5424	0.4433	0.4380	0.3753	0.3483	0.4476
BS	0.1924	0.1620	0.1533	0.1898	0.1765	0.1669	0.1685	0.1806	0.2115	0.1810	0.1783
BSS	0.2191	0.2557	0.3555	0.2207	0.2705	0.2916	0.2916	0.2408	0.1256	0.2687	0.2540
ROC_AUC	0.7782	0.8166	0.8638	0.7777	0.8061	0.8207	0.8239	0.7886	0.7110	0.8002	0.7987
Runtime (seconds)	0.1028	0.0854	0.0838	0.0825	0.0851	0.0842	0.0832	0.0827	0.0833	0.0841	0.8571

Random Forest Confusion Matrix (Sum of all folds):

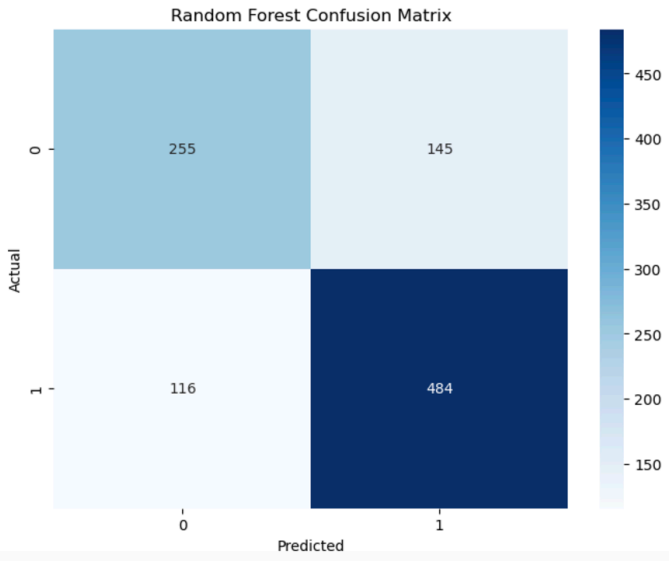

Figure 6: RFC results

SVM Results:

SVM Metrics:										
Metric	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10	Average
TPR	0.8750	0.8235	0.8689	0.8276	0.8136	0.8548	0.8197	0.8361	0.7966	0.7818	0.8298
FPR	0.6364	0.4375	0.4872	0.5476	0.5122	0.3421	0.5385	0.6154	0.5610	0.4667	0.5144
TNR	0.3636	0.5625	0.5128	0.4524	0.4878	0.6579	0.4615	0.3846	0.4390	0.5333	0.4856
FNR	0.1250	0.1765	0.1311	0.1724	0.1864	0.1452	0.1803	0.1639	0.2034	0.2182	0.1702
Precision	0.6364	0.8000	0.7361	0.6761	0.6957	0.8030	0.7042	0.6800	0.6714	0.6719	0.7075
Accuracy	0.6500	0.7400	0.7300	0.6700	0.6800	0.7800	0.6800	0.6600	0.6500	0.6700	0.6910
Recall	0.8750	0.8235	0.8689	0.8276	0.8136	0.8548	0.8197	0.8361	0.7966	0.7818	0.8298
Error Rate	0.3500	0.2600	0.2700	0.3300	0.3200	0.2200	0.3200	0.3400	0.3500	0.3300	0.3090
F1 Score	0.7368	0.8116	0.7970	0.7442	0.7500	0.8281	0.7576	0.7500	0.7287	0.7227	0.7627
BACC	0.6193	0.6930	0.6908	0.6400	0.6507	0.7564	0.6406	0.6103	0.6178	0.6576	0.6577
TSS	0.2386	0.3860	0.3817	0.2800	0.3014	0.5127	0.2812	0.2207	0.2356	0.3152	0.3153
HSS	0.2515	0.3925	0.4021	0.2925	0.3130	0.5234	0.2948	0.2360	0.2457	0.3210	0.3272
BS	0.2182	0.1765	0.1805	0.2283	0.1951	0.1771	0.1895	0.2024	0.2177	0.1994	0.1985
BSS	0.1145	0.1891	0.2415	0.0629	0.1933	0.2481	0.2036	0.1491	0.1002	0.1942	0.1696
ROC_AUC	0.7281	0.7698	0.7900	0.6757	0.7520	0.7916	0.7848	0.7512	0.7065	0.7640	0.7514
Runtime (seconds)	0.0431	0.0421	0.0441	0.0441	0.0442	0.0448	0.0526	0.0551	0.0473	0.0491	0.4666

SVM Confusion Matrix (Sum of all folds):

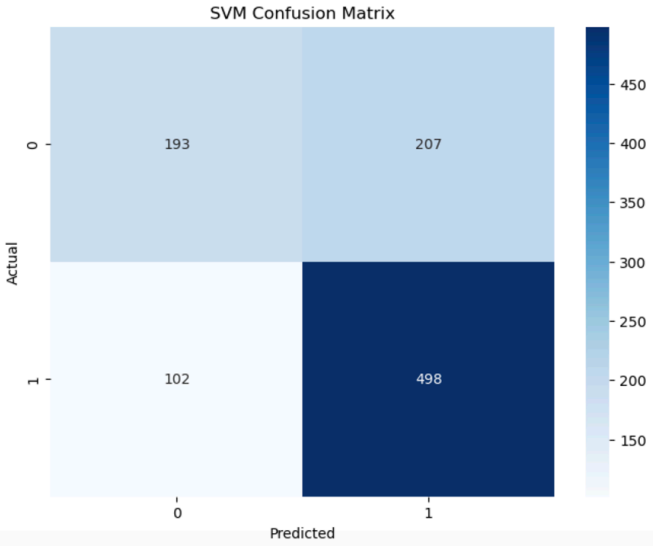

Figure 7: SVM results

BiLSTM Results:

BiLSTM Metrics:

Metric	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10	Average
TPR	0.9107	0.7059	0.6557	0.7586	0.8644	0.7258	0.8197	0.7541	0.8814	0.8909	0.7967
FPR	0.9091	0.6250	0.4872	0.5238	0.4634	0.2632	0.5128	0.3333	0.4878	0.3111	0.4917
TNR	0.0909	0.3750	0.5128	0.4762	0.5366	0.7368	0.4872	0.6667	0.5122	0.6889	0.5083
FNR	0.0893	0.2941	0.3443	0.2414	0.1356	0.2742	0.1803	0.2459	0.1186	0.1091	0.2033
Precision	0.5604	0.7059	0.6780	0.6667	0.7286	0.8182	0.7143	0.7797	0.7222	0.7778	0.7152
Accuracy	0.5500	0.6000	0.6000	0.6400	0.7300	0.7300	0.6900	0.7200	0.7300	0.8000	0.6790
Recall	0.9107	0.7059	0.6557	0.7586	0.8644	0.7258	0.8197	0.7541	0.8814	0.8909	0.7967
Error Rate	0.4500	0.4000	0.4000	0.3600	0.2700	0.2700	0.3100	0.2800	0.2700	0.2000	0.3210
F1 Score	0.6939	0.7059	0.6667	0.7097	0.7907	0.7692	0.7634	0.7667	0.7939	0.8305	0.7490
BACC	0.5008	0.5404	0.5843	0.6174	0.7005	0.7313	0.6534	0.7104	0.6968	0.7899	0.6525
TSS	0.0016	0.0809	0.1686	0.2348	0.4010	0.4626	0.3069	0.4208	0.3936	0.5798	0.3050
HSS	0.0018	0.0809	0.1670	0.2411	0.4181	0.4467	0.3202	0.4169	0.4136	0.5893	0.3096
BS	0.2108	0.2050	0.2155	0.2162	0.1956	0.1653	0.1937	0.1856	0.1892	0.1504	0.1927
BSS	0.1444	0.0580	0.0941	0.1123	0.1916	0.2983	0.1860	0.2199	0.2178	0.3923	0.1914
ROC_AUC	0.7557	0.6677	0.6789	0.7011	0.7673	0.8277	0.7629	0.7852	0.7780	0.8602	0.7585
Runtime (seconds)	1.7610	1.6309	1.6110	1.5999	1.6201	1.6196	1.6172	1.6038	1.6098	5.9594	20.6327

BiLSTM Confusion Matrix (Sum of all folds):

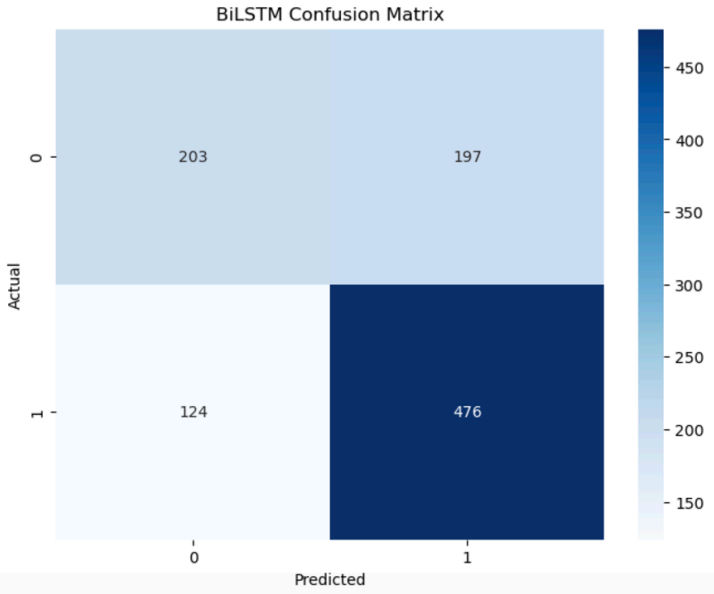

Figure 8: BiLSTM Results

```
================================================

The fastest algorithm is: SVM
================================================
```

Figure 9: Printing the fastest algorithm

Model Comparison:

Model Comparison

Model	Accuracy	Precision	Recall	F1 Score	ROC_AUC	Runtime (seconds)
Random Forest	0.7390	0.7697	0.8057	0.7867	0.7987	0.0857
SVM	0.6910	0.7075	0.8298	0.7627	0.7514	0.0467
BiLSTM	0.6790	0.7152	0.7967	0.7490	0.7585	2.0633

Figure 10: Comparing the models table

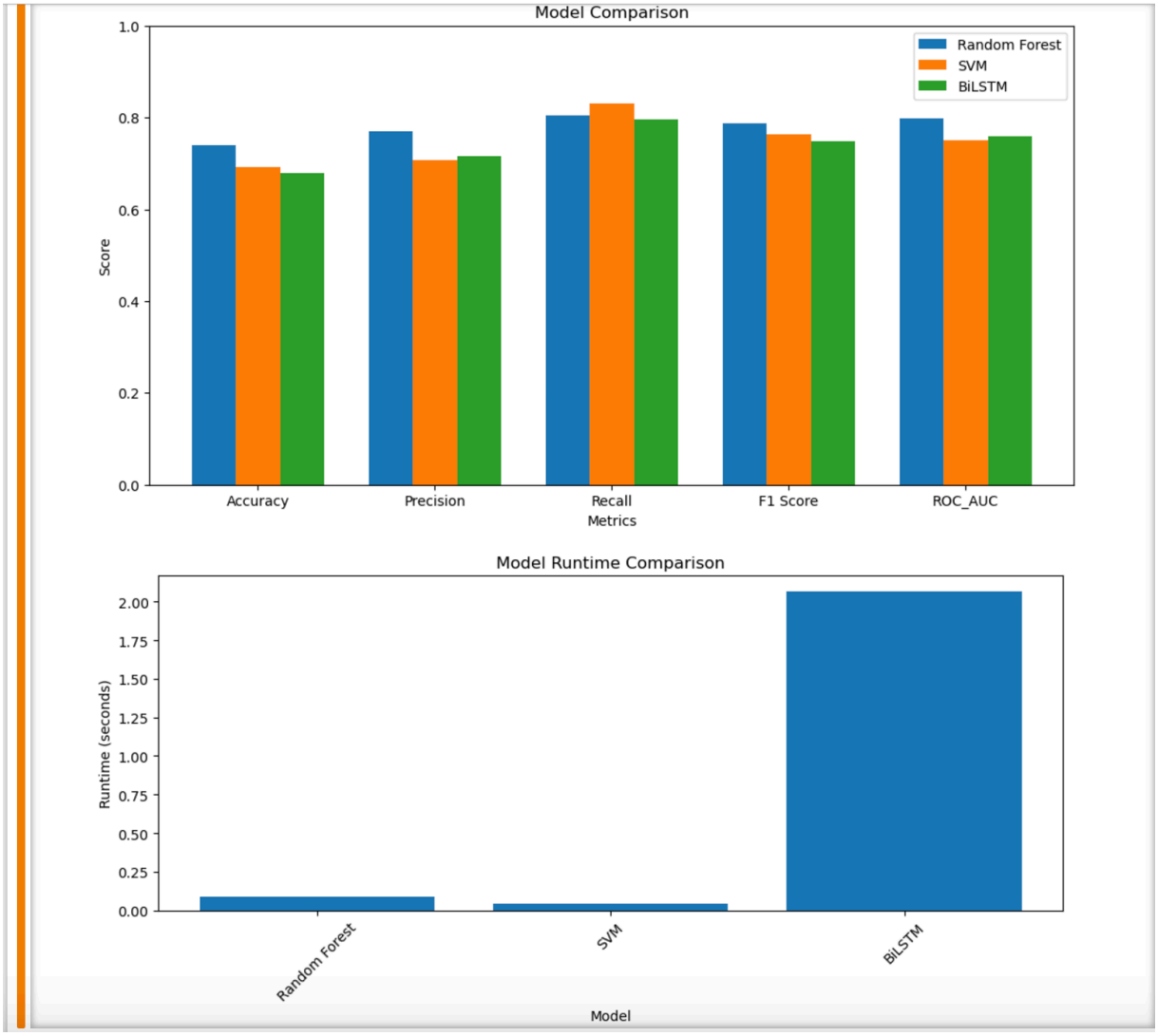

Figure 11: Visualizing the Model Comparison

Other

Repository Link

https://github.com/AnjaliguptaRaghavendra/raghavendra_anjaligupta_finaltermproj

Kaggle dataset link

https://www.kaggle.com/datasets/shriyashjagtap/fraudulent-e-commerce-transactions