Diabetes Prediction: A Comparative Analysis of Classification Models

Srikar Goud Kalal (sk3772)
Course: CS 634 (101) Data Mining
Date: 11-20-2024

Submitted to: Professor: Yasser Abduallah

Contents

1 Introduction

This report presents the implementation of three different classification algorithms on the Diabetes Prediction dataset from the Kaggle. The selected algorithms include K-Nearest Neighbors (KNN), Random Forest, and Long Short-Term Memory (LSTM). Each algorithm is evaluated based on its classification performance using metrics such as True Positive Rate, True Negative Rate, False Positive Rate, and others. The evaluation is conducted using 10-fold cross-validation.

2 Dataset

The dataset used for this project is the Diabetes Prediction dataset, available at kaggle. The Diabetes prediction dataset is a collection of medical and demographic data from patients, along with their diabetes status (positive or negative). The data includes features such as age, gender, body mass index (BMI), hypertension, heart disease, smoking history, HbA1c level, and blood glucose level. This dataset can be used to build machine learning models to predict diabetes in patients based on their medical history and demographic information. This can be useful for healthcare professionals in identifying patients who may be at risk of developing diabetes and in developing personalized treatment plans. Additionally, the dataset can be used by researchers to explore the relationships between various medical and demographic factors and the likelihood of developing diabetes.

- **Age**: Integer feature representing the individual's age.
- **BMI**: Continuous feature for Body Mass Index, indicating body fat based on height and weight.
- **Glucose**: Numeric feature for glucose concentration levels in blood (e.g., mg/dL).

- **BloodPressure**: Continuous feature for diastolic blood pressure (e.g., mm Hg).
- **SkinThickness**: Numeric feature representing triceps skinfold thickness (e.g., mm).
- **Insulin**: Continuous feature measuring serum insulin levels (e.g., μU/mL).
- **DiabetesPedigreeFunction**: Continuous feature calculating a score for diabetes likelihood based on family history.
- **Pregnancies**: Integer feature showing the number of times the individual has been pregnant.
- **Outcome**: Binary target variable indicating diabetes diagnosis (0: No, 1: Yes).

3 Algorithms

- **K-Nearest Neighbors (KNN)**: This algorithm is straightforward and works by comparing each new instance to its nearest neighbors in the training data.
- **Random Forest**: This method creates several decision trees and combines their predictions to determine the final result.
- **Long Short-Term Memory (LSTM)**: A special type of recurrent neural network (RNN) designed to remember information over long periods.

4 Methodology

The methodology includes the following steps:

4.1 Data Preprocessing

1. Handling Missing Values:
    - Filled in missing values in ”workclass,” ”occupation,” and ”native-country” columns using the most common value in each column.
2. Encoding Target Variable:
    - Converted the target variable ”income” to binary format, where incomes over 50K are marked as 1, and 50K or less are marked as 0.
3. Encoding Categorical Features:
    - Transformed categorical features into numerical values with label encoding, and also used one-hot encoding to create dummy variables for each unique category.
4. Feature Scaling:
    - Standardized continuous features like age, fnlwgt, education-num, capital-gain, capital-loss, and hours-per-week to ensure they are on a similar scale, which helps in improving the model's performance.
5. Train-Test Split:
    - Divided the dataset into 70% for training and 30% for testing the model.

This preprocessed dataset is now ready for implementing and training classification models.

5 Model Training

5.1 Hyperparameter Tuning

- To train the three algorithms—Random Forest, K-Nearest Neighbors (KNN), and LSTM—I began by tuning their hyperparameters.
- For Random Forest, I used RandomizedSearchCV to explore various parameter configurations, such as the number of trees (*n_estimators*), *min_samples_leaf*, and the criterion.
- This method allowed me to sample from a predefined range of values efficiently without the exhaustive process of grid search, striking a balance between model complexity and accuracy.
- For the K-Nearest Neighbors (KNN) model, I used GridSearchCV to systematically test all possible parameter combinations, mainly focusing on the number of neighbors (*n_neighbors*) and the distance metric for computing similarity.
- This approach helped me identify the optimal k value that minimized error and improved prediction accuracy.
- After these tuning processes, I had the best-performing hyperparameters for both Random Forest and KNN, ready for cross-validation.

5.2 10-Fold Cross-Validation

- With the best hyperparameters identified, I evaluated each model using 10-fold cross-validation on the training data.
- The data was split into 10 equal subsets.
- For each fold, I trained the model on 9 subsets and validated it on the remaining one, repeating this process across all 10 folds.

- This ensured that each data point was used for both training and validation, providing a comprehensive assessment of each model's performance.
- The 10-fold cross-validation helped assess the stability and reliability of the models and offered insights into their generalization capability by reducing the risk of overfitting.
- Through this structured training and validation workflow, I was able to refine the models effectively before making final predictions on unseen data.

6 Performance Metrics Calculation

The 'calculate_metrics' function evaluates the model by computing the following:

1. **True Positives (TP)**: Correctly predicted positive cases.
2. **True Negatives (TN)**: Correctly predicted negative cases.
3. **False Positives (FP)**: Negative cases incorrectly predicted as positive.
4. **False Negatives (FN)**: Positive cases incorrectly predicted as negative.
5. **Precision**: Measures how many of the predicted positive cases are actually positive.
6. **Recall (True Positive Rate)**: Measures how well the model identifies actual positive cases.
7. **True Negative Rate (Specificity)**: Measures how well the model identifies actual negative cases.
8. **False Positive Rate (FPR)**: Measures the proportion of actual negatives incorrectly predicted as positive.

9. **False Negative Rate (FNR)**: Measures the proportion of actual positives incorrectly predicted as negative.
10. **Balanced Accuracy (BACC)**: Provides a balanced view of accuracy across classes, especially useful for imbalanced datasets.
11. **True Skill Statistic (TSS)**: Assesses the model's ability to distinguish between classes.
12. **Heidke Skill Score (HSS)**: Compares model performance to random chance.
13. **F1 Score**: Balances Precision and Recall, offering a combined view of the two metrics.
14. **Accuracy**: Overall correctness of the model's predictions across both positive and negative cases.
15. **Error Rate**: Proportion of incorrect predictions made by the model.
16. **Sklearn Accuracy**: Standard accuracy measure calculated using sklearn's function.
17. **AUC (Area Under the Curve)**: Reflects the model's ability to distinguish between classes, with higher values indicating better performance.
18. **Brier Score**: Evaluates the accuracy of probability predictions, with lower values indicating more accurate probability estimates.

7 Code

Diabetes prediction: A Comparative Analysis of Classification Models

- K Srikar Goud
- NJIT UCID: Sk3772

Importing Libararies

[8]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[10]:
```
from keras.models import Sequential
from sklearn.model_selection import KFold
from sklearn.metrics import auc, roc_curve
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from keras.layers import LSTM, Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
```

Figure 1:

Load and Preprocess Data

[19]:
```
# Load the dataset
dataset = pd.read_csv('diabetes_prediction_dataset.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, -1].values
```

[21]:
```
# Encode categorical features
le = LabelEncoder()
X[:, 0] = le.fit_transform(X[:, 0])

# One-hot encoding for a categorical column
ct = ColumnTransformer(transformers=[('Encoding', OneHotEncoder(), [4])], remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

[23]:
```
# Split the dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=0)
```

[25]:
```
# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Reshape data for LSTM model
X_train_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0], X_train_scaled.shape[1], 1)
X_test_reshaped = X_test_scaled.reshape(X_test_scaled.shape[0], X_test_scaled.shape[1], 1)
```

Figure 2:

Define LSTM Model

[46]:
```
"""def create_lstm_model():
    lstm_model = Sequential()
    lstm_model.add(LSTM(units=50, activation='relu', input_shape=(X_train_reshaped.shape[1], 1)))
    lstm_model.add(Dense(units=1, activation='sigmoid'))
    lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return lstm_model"""
from keras.layers import Input

# Initialize LSTM model
def create_lstm_model():
    lstm_model = Sequential()
    lstm_model.add(Input(shape=(X_train_reshaped.shape[1], 1)))
    lstm_model.add(LSTM(units=50, activation='relu'))
    lstm_model.add(Dense(units=1, activation='sigmoid'))
    lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return lstm_model
```

Figure 3:

Define Classifiers and K-Fold Cross-Validation

[49]:
```
# Define classifiers
classifiers = {
    'Random Forest': RandomForestClassifier(random_state=0),
    'K-Nearest Neighbors': KNeighborsClassifier(n_neighbors=5),
    'LSTM': create_lstm_model()
}

# Initialize KFold with 5 folds (you can adjust as needed)
kf = KFold(n_splits=5, shuffle=True, random_state=0)

# Initialize a list to store fold-wise metrics
fold_metrics = []
plt.figure()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
```

[49]:
```
Text(0.5, 1.0, 'ROC Curve')
```

Figure 4:

Perform Cross-Validation and Calculate Metrics

[52]:
```python
# Perform KFold cross-validation for each classifier
for clf_name, clf in classifiers.items():
    print(f"Evaluating {clf_name}...")
    accuracy_values = []
    precision_values = []
    tp_values = []
    tn_values = []
    fp_values = []
    fn_values = []
    tpr_values = []
    tnr_values = []
    npv_values = []
    fpr_values = []
    fdr_values = []
    fnr_values = []
    f1_values = []
    bacc_values = []
    tss_values = []
    hss_values = []
    bs_values = []
    bss_values = []

    for fold_no, (train_index, test_index) in enumerate(kf.split(X_train_scaled), 1):
        X_train_fold, X_val_fold = X_train_scaled[train_index], X_train_scaled[test_index]
        Y_train_fold, Y_val_fold = Y_train[train_index], Y_train[test_index]

        if clf_name != 'LSTM':
            # Fit classifier to training fold
            clf.fit(X_train_fold, Y_train_fold)

            # Predict labels on validation fold
            Y_pred_fold = clf.predict(X_val_fold)
        else:
            # Create a new instance of LSTM model for each fold
            lstm_model = create_lstm_model()

            # Fit LSTM model to training fold
            lstm_model.fit(X_train_reshaped[train_index], Y_train_fold, epochs=1, batch_size=32, verbose=0)

            # Predict labels on validation fold
            Y_pred_fold = (lstm_model.predict(X_train_reshaped[test_index]) > 0.5).astype("int32").flatten()
```

Figure 5:

```python
        # Calculate performance metrics
        tp = np.sum((Y_pred_fold == 1) & (Y_val_fold == 1))
        tn = np.sum((Y_pred_fold == 0) & (Y_val_fold == 0))
        fp = np.sum((Y_pred_fold == 1) & (Y_val_fold == 0))
        fn = np.sum((Y_pred_fold == 0) & (Y_val_fold == 1))
        acc = (tp + tn) / (tp + fp + tn + fn)
        ppv = tp / (tp + fp)
        tpr = tp / (tp + fn)
        tnr = tn / (tn + fp)
        npv = tn / (tn + fn)
        fpr = fp / (fp + tn)
        fdr = fp / (fp + tp)
        fnr = fn / (fn + tp)
        f1 = 2 * tp / (2 * tp + fp + fn)
        bacc = (tpr + tnr) / 2
        tss = tp / (tp + fn)
        hss = 2 * (tp * tn - fp * fn) / ((tp + fn) * (fn + tn) + (tp + fp) * (fp + tn))
        bs = np.mean((Y_pred_fold - Y_val_fold) ** 2)
        reference_forecast = np.mean(Y_val_fold)
        bss = 1 - (bs / reference_forecast)

        # Append metrics for current fold
        fold_metrics.append({
            'Classifier': clf_name,
            'Fold': fold_no,
            'True Positives': tp,
            'True Negatives': tn,
            'False Positives': fp,
            'False Negatives': fn,
            'Accuracy': round(acc * 100, 2),
            'Precision': round(ppv * 100, 2),
            'Sensitivity': round(tpr * 100, 2),
            'Specificity': round(tnr * 100, 2),
            'Negative Predictive Value': round(npv * 100, 2),
            'False Positive Rate': round(fpr * 100, 2),
            'False Discovery Rate': round(fdr * 100, 2),
            'False Negative Rate': round(fnr * 100, 2),
            'F1 Score': round(f1 * 100, 2),
            'Balanced Accuracy': round(bacc * 100, 2),
            'True Skill Statistics': round(tss * 100, 2),
            'Heidke Skill Score': round(hss * 100, 2),
            'Brier Score': bs,
            'Brier Skill Score': bss
        })

    # Plot ROC curve
    fpr2, tpr2, thresholds = roc_curve(Y_val_fold, Y_pred_fold)
    plt.plot(fpr2, tpr2, label=f'{clf_name}')
    plt.legend(loc="lower right")
```

Figure 6:

8 How to run the code

Use Python 3.8 or later (recommended version: Python 3.8/3.9).

8.1 Install the required packages using pip.

```
pip install numpy
pip install pandas
pip install matplotlib
pip install scikit-learn
pip install keras
```

Figure 7:

9 Results

output of the code :

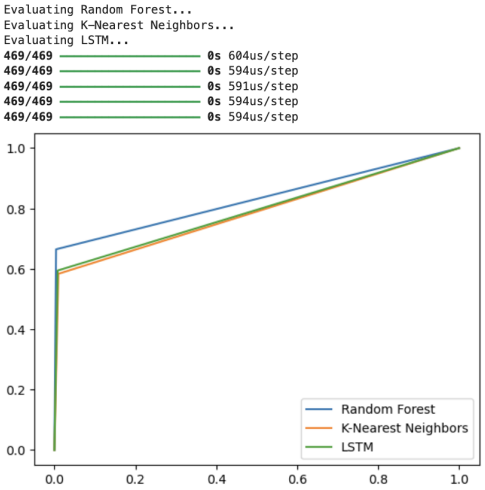

Figure 8:

Display Metrics and Save to CSV

[55]:
```
plt.show()

# Save metrics to CSV
metrics_df = pd.DataFrame(fold_metrics)
print(metrics_df)
csv_file = 'output.csv'
if os.path.exists(csv_file):
    os.remove(csv_file)
metrics_df.to_csv(csv_file, index=False)
```

Figure 9:

Delimiter: ,

	Classifier	Fold	True Positives	True Negatives	False Positives	False Negatives	Accuracy
1	Random Forest	1	867	13723	48	362	97.27
2	Random Forest	2	885	13625	43	447	96.73
3	Random Forest	3	862	13692	43	403	97.03
4	Random Forest	4	918	13670	41	371	97.25
5	Random Forest	5	851	13663	57	429	96.76
6	K-Nearest Neighbors	1	771	13654	117	458	96.17
7	K-Nearest Neighbors	2	774	13570	98	558	95.63
8	K-Nearest Neighbors	3	764	13637	98	501	96.01
9	K-Nearest Neighbors	4	821	13599	112	468	96.13
10	K-Nearest Neighbors	5	747	13587	133	533	95.56
11	LSTM	1	768	13680	91	461	96.32
12	LSTM	2	793	13597	71	539	95.93
13	LSTM	3	678	13708	27	587	95.91
14	LSTM	4	734	13653	58	555	95.91
15	LSTM	5	761	13609	111	519	95.8

Figure 10:

Delimiter: ,

	Precision	Sensitivity	Specificity	Negative Predictive Value	False Positive Rate	False Discovery Rate	False Negative Rate	F1 Score
1	94.75	70.55	99.65	97.43	0.35	5.25	29.45	80.88
2	95.37	66.44	99.69	96.82	0.31	4.63	33.56	78.32
3	95.25	68.14	99.69	97.14	0.31	4.75	31.86	79.45
4	95.72	71.22	99.7	97.36	0.3	4.28	28.78	81.67
5	93.72	66.48	99.58	96.96	0.42	6.28	33.52	77.79
6	86.82	62.73	99.15	96.75	0.85	13.18	37.27	72.84
7	88.76	58.11	99.28	96.05	0.72	11.24	41.89	70.24
8	88.63	60.4	99.29	96.46	0.71	11.37	39.6	71.84
9	88.0	63.69	99.18	96.67	0.82	12.0	36.31	73.9
10	84.89	58.36	99.03	96.23	0.97	15.11	41.64	69.17
11	89.41	62.49	99.34	96.74	0.66	10.59	37.51	73.56
12	91.78	59.53	99.48	96.19	0.52	8.22	40.47	72.22
13	96.17	53.6	99.8	95.89	0.2	3.83	46.4	68.83
14	92.68	56.94	99.58	96.09	0.42	7.32	43.06	70.54
15	87.27	59.45	99.19	96.33	0.81	12.73	40.55	70.72

Figure 11:

Delimiter: ,

	False Discovery Rate	False Negative Rate	F1 Score	Balanced Accuracy	True Skill Statistics	Heidke Skill Score	Brier Score	Brier Skill Score
1	5.25	29.45	80.88	85.1	70.55	79.44	0.027333333333333334	0.6663954434499593
2	4.63	33.56	78.32	83.06	66.44	76.61	0.03266666666666666	0.6321321321321323
3	4.75	31.86	79.45	83.91	68.14	77.89	0.029733333333333334	0.6474308300395257
4	4.28	28.78	81.67	85.46	71.22	80.22	0.027466666666666667	0.6803723816912335
5	6.28	33.52	77.79	83.03	66.48	76.09	0.0324	0.6203125
6	13.18	37.27	72.84	80.94	62.73	70.83	0.03833333333333333	0.532139951179821
7	11.24	41.89	70.24	78.7	58.11	67.99	0.04373333333333333	0.5075075075075075
8	11.37	39.6	71.84	79.84	60.4	69.77	0.039933333333333335	0.5264822134387351
9	12.0	36.31	73.9	81.44	63.69	71.87	0.03866666666666667	0.5500387897595035
10	15.11	41.64	69.17	78.69	58.36	66.86	0.0444	0.47968749999999993
11	10.59	37.51	73.56	80.91	62.49	71.65	0.0368	0.5508543531326282
12	8.22	40.47	72.22	79.51	59.53	70.14	0.04066666666666666	0.5420420420420421
13	3.83	46.4	68.83	76.7	53.6	66.83	0.040933333333333335	0.5146245059288537
14	7.32	43.06	70.54	78.26	56.94	68.48	0.04086666666666667	0.5244375484871994
15	12.73	40.55	70.72	79.32	59.45	68.55	0.042	0.5078125

Figure 12:

10 Conclusion

In this project, we tried various classification algorithms on the Adult Income dataset. The results highlighted each model's pros and cons, giving us useful information for choosing the best method for similar tasks in the future. This analysis shows how important it is to evaluate models to get accurate predictions.

11 Github Repository

https://github.com/srikargoud2002/KNN-RANDOM-FOREST-LSTM-Implementation-.git