Determining Trends in Diabetes Related Health Markers Using Machine Language

Joel Thomas

Week 7 – Rough Draft Template

DSC 580

Data Preparation:

This dataset was taken publically from Kaggle and was already in a healthy state for other data scientists to utilize data science prediction techniques and it was already split into a 80/20 split for testing and training amongst Machine learning Models. Below in my work, we do some minor fixes to this dataset that include the following:

- Dropping and targeting the Diabetes_012 column for ML Practices.
- Splitting the dataset into a 80/20 set.

Outside of this, the EDA process is complete and is ready to create models to predict Diabetes from related health markers using Logistic Regression and Random Forest ML Models.

Below is a brief look at how the dataset looks like:

In [ ]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('Diabetes.csv')
data.head()
```

Out[ ]:

	Diabetes_012	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttac
0	0.0	1.0	1.0	1.0	40.0	1.0	0.0	0
1	0.0	0.0	0.0	0.0	25.0	1.0	0.0	0
2	0.0	1.0	1.0	1.0	28.0	0.0	0.0	0
3	0.0	1.0	0.0	1.0	27.0	0.0	0.0	0
4	0.0	1.0	1.0	1.0	24.0	0.0	0.0	0

5 rows × 22 columns

Now lets take this data, split it and test/training the Logistic Regression and Random Forest Models and provide charts and graphs to show the strength of our predictions.

Split Train and Test Data 20/80

In [ ]:
```
# Preprocess the data
# Remove any rows with missing values
data.dropna(inplace=True)

# Define features and target variable
X = data.drop(columns=['Diabetes_012'])
y = data['Diabetes_012']
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

Train Logistic Regression Classifier

In [ ]:
```
# Import necessary libraries
from sklearn.preprocessing import StandardScaler

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train Logistic Regression Classifier with scaled data
lr_classifier = LogisticRegression(random_state=42, max_iter=1000)
lr_classifier.fit(X_train_scaled, y_train)

# Predict using Logistic Regression Classifier
lr_pred = lr_classifier.predict(X_test_scaled)
```

Train Random Forest Classifier

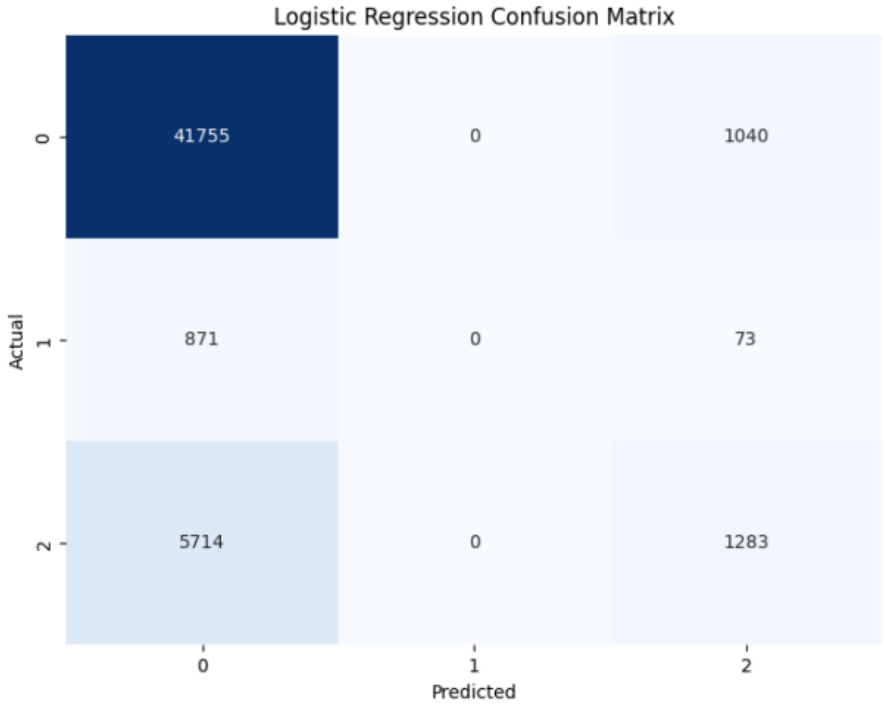
[ ]:
```
# Import necessary libraries
from sklearn.preprocessing import StandardScaler

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train_scaled, y_train)

# Predict using Random Forest Classifier
rf_pred = rf_classifier.predict(X_test_scaled)

# Train Logistic Regression Classifier
lr_classifier = LogisticRegression(random_state=42, max_iter=1000)
lr_classifier.fit(X_train_scaled, y_train)

# Predict using Logistic Regression Classifier
lr_pred = lr_classifier.predict(X_test_scaled)
```

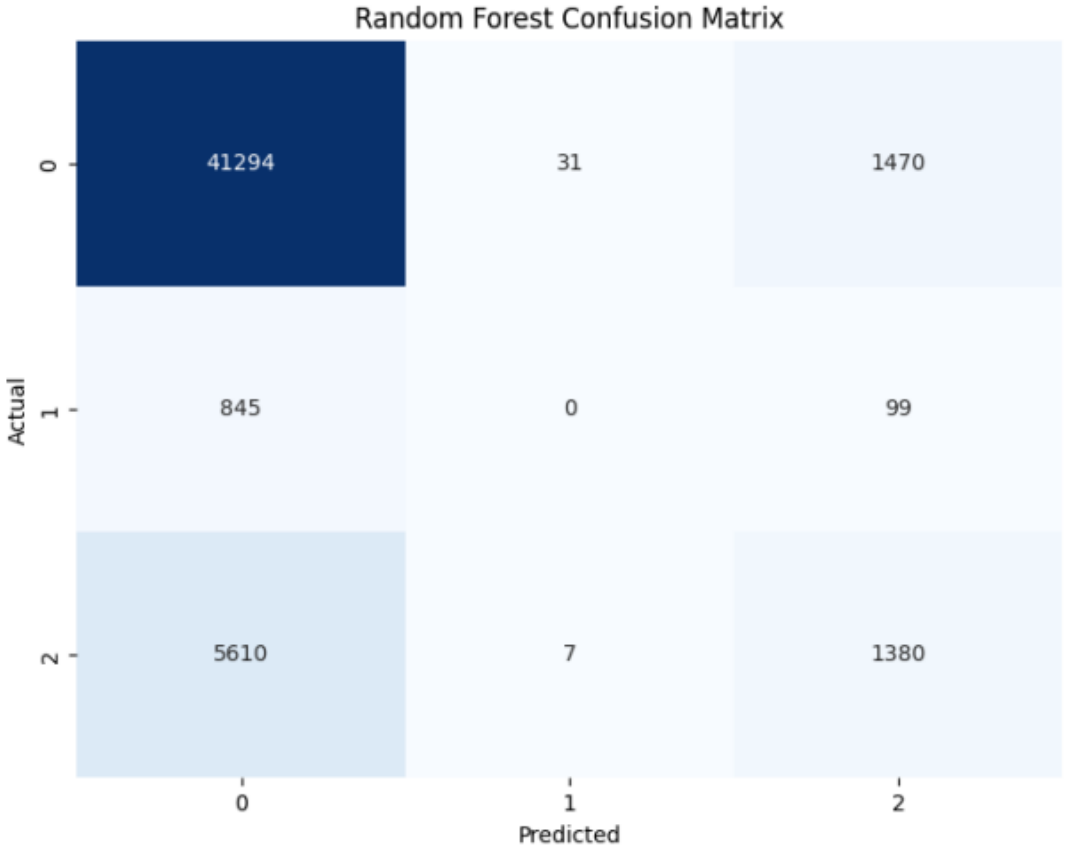
Now that our models are tested, Lets show Confusion matrix' to evaluate model effectiveness.

Plot Confusion Matrix to show results of models trained to predict Diabetes from other Metrics.

In [ ]:
```
import seaborn as sns


# Calculate evaluation metrics
rf_accuracy = accuracy_score(y_test, rf_pred)
```

```
rf_precision = precision_score(y_test, rf_pred, average='macro')
rf_recall = recall_score(y_test, rf_pred, average='macro')
rf_f1 = f1_score(y_test, rf_pred, average='macro')

lr_accuracy = accuracy_score(y_test, lr_pred)
lr_precision = precision_score(y_test, lr_pred, average='macro')
lr_recall = recall_score(y_test, lr_pred, average='macro')
lr_f1 = f1_score(y_test, lr_pred, average='macro')

# Confusion Matrix for Random Forest
rf_conf_matrix = confusion_matrix(y_test, rf_pred)

# Confusion Matrix for Logistic Regression
lr_conf_matrix = confusion_matrix(y_test, lr_pred)

# Plot Confusion Matrix for Random Forest
plt.figure(figsize=(8, 6))
plt.title('Random Forest Confusion Matrix')
sns.heatmap(rf_conf_matrix, annot=True, fmt='g', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Plot Confusion Matrix for Logistic Regression
plt.figure(figsize=(8, 6))
plt.title('Logistic Regression Confusion Matrix')
sns.heatmap(lr_conf_matrix, annot=True, fmt='g', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```


Logistic Regression Confusion Matrix
41755
0
1040
871
0
73
5714
0
1283
Actual
0
1
2
0
1
2
Predicted


Random Forest Confusion Matrix
41294
31
1470
845
0
99
5610
7
1380
Actual
0
1
2
0
1
2
Predicted

From the Matrix's above, Both models are densily populating the (0,0) square of the plot. This pattern often represents True Negatives - Which suggests that the model represents negative cases.

[ ]:
```
# Output statistical parameters
print("Random Forest Classifier Metrics:")
print("Accuracy:", rf_accuracy)
print("Precision:", rf_precision)
print("Recall:", rf_recall)
print("F1 Score:", rf_f1)
print("\n")
print("Logistic Regression Classifier Metrics:")
print("Accuracy:", lr_accuracy)
print("Precision:", lr_precision)
print("Recall:", lr_recall)
print("F1 Score:", lr_f1)
```

```
Random Forest Classifier Metrics:
Accuracy: 0.8410990223904131
Precision: 0.4442563872895852
Recall: 0.38738439741802005
F1 Score: 0.3965432034760695


Logistic Regression Classifier Metrics:
Accuracy: 0.8482734153263954
Precision: 0.4664177343335753
Recall: 0.38635413151906395
F1 Score: 0.396505023212427
```

Data Interpretation and Analysis –

Accuracy: Both classifiers achieve relatively high accuracy scores, with the Logistic Regression slightly outperforming the Random Forest classifier. This indicates that both models are able to correctly classify a significant portion of the instances in the dataset.

Precision: Precision measures the proportion of true positive predictions among all positive predictions made by the model. Both classifiers have low precision scores, indicating that they have a relatively high rate of false positive predictions. In the context of identifying diabetes, this means that a considerable number of individuals predicted to have diabetes by the models may not actually have it. Recall:

Recall, also known as sensitivity, measures the proportion of true positive predictions among all actual positive instances in the dataset. Both classifiers have moderate recall scores, suggesting that they are able to capture a substantial portion of the actual positive cases. However, given the information about the confusion matrix being heavily biased towards the true negative (0,0) square, it's likely that the recall for the positive class (indicating diabetes) is lower than desired.

F1 Score: The F1 score is the harmonic mean of precision and recall and provides a balanced assessment of a classifier's performance. Both classifiers have relatively low F1 scores, indicating that they struggle to achieve both high precision and high recall simultaneously. This suggests that there's room for improvement in the classifiers' performance in identifying diabetes.

Conclusion –

In summary, while both classifiers demonstrate decent performance in terms of accuracy, there is room for improvement in terms of precision, recall, and overall F1 score, especially in correctly

identifying diabetic cases. Further analysis and possibly model refinement are warranted to enhance the classifiers' performance in identifying diabetes accurately. Additionally, addressing the imbalance in the confusion matrix should be a priority to ensure balanced and reliable predictions for both diabetic and non-diabetic cases.