Predicting Stroke Susceptibility By Health Data Using Machine Learning

CS-C3240 Machine Learning D – Stage 2

I. Introduction

In the healthcare field, it is crucial to accurately assess a patient's risk for stroke, as early intervention can significantly improve outcomes and save lives. Stroke is ranked as the second leading cause of death worldwide with an annual mortality rate of about 5.5 million [1]. However, identifying individuals at risk is challenging due to the multifactorial nature of the disease, which is influenced by a combination of demographic and medical factors. This project's application domain is medical diagnosis, which aims to predict the likelihood of stroke in individuals based on their health data using machine learning techniques.

The paper is structured as follows: Section II discusses the problem formulation and the features of the dataset; Section III covers data preparation, model selection, and validation techniques; Section IV presents the results of the study; and Section V concludes the report with reflections on the limitations and potential improvements for future work.

II. Problem Formulation

In this project, we aim to predict whether or not a person is at risk of experiencing a stroke based on various health and demographic features listed below. We are utilizing the dataset "Stroke Prediction Dataset" from Fedesoriano and made available on Kaggle [2], containing individuals with features related to their health and lifestyle. The dataset includes the following feature variables:

• Gender, Hypertension, Heart Disease, Married	(Binary)
• Age, BMI (Body Mass Index), Average Glucose Level	(Continuous)
• Work Type, Residence Type, Smoking Status	(Categorical)

This is a binary classification problem where machine learning models will classify individuals as either at risk (1) or not at risk (0) of stroke. This project utilizes supervised learning techniques, where the label we are concerned with is the binary outcome of stroke occurrence.

III. Methods

1. Data Preprocessing

Overall, the data set has 5110 datapoints of 14 features, and there are 201 missing data in the feature BMI, which is critical for predicting strokes as BMI is a key health indicator. To handle these missing values, the mean of the BMI column is calculated and used to replace any null values.

Several categorical features in the dataset, such as gender, residence_type, smoking_status, and work_type, need to be converted into numerical values for use in machine learning algorithms. Gender is mapped to integers: 'Male' to 0, 'Female' to 1, and 'Other' to -1. Residence Type is mapped to 0 for 'Rural' and 1 for 'Urban'. Different Work Types were encoded as follows: 'Private' as 0, 'Self-employed' as 1, 'Govt_job' as 2, 'children' as 3, and 'Never_worked' as 4. And finally the Smoking status is mapped as 'never smoked' for 0, 'formerly smoked' for 1, 'smokes' for 2, 'Unknown' for 0.

In the process of feature selection, we initially eliminate features deemed irrelevant to the prediction of stroke occurrence, such as 'id', and 'married'. Subsequent analysis using boxplot and histogram visualizations reveals that the age feature is a significant determinant influencing the likelihood of stroke incidence, whereas bmi and glucose_level show minor correlations to the labels. However, we still keep those features to ensure the diversity of medical background.

The dataset is imbalanced, with far more negative cases (non-stroke patients) than positive cases (stroke patients). To address this, we applied SMOTE (Synthetic Minority Oversampling Technique) [3], a widely used method for handling class imbalances in binary classification problems.

Finally, scaling may be required for continuous features such as age, avg_glucose_level, and bmi to ensure all features contribute equally to the prediction model. In this case, scaling can be applied to these features using StandardScaler from the sklearn library [4].

2. Logistic Regression

In this project, Logistic Regression was chosen as one of the machine learning models to predict whether an individual is at risk of experiencing a stroke. Logistic Regression is a well-established and powerful algorithm for binary classification tasks, which is appropriate for this problem where the target is a binary outcome—either a person will have a stroke (1) or they will not (0). It is particularly suited for this task because it is simple, interpretable, and provides probabilistic outputs. These probability scores can be thresholded (e.g., at 0.5) to make a final binary classification.

3. Random Forest

For this project, Random Forest [5] was chosen as the second model after Logistic Regression to handle the binary classification task of predicting stroke susceptibility. In detail, it operates by constructing multiple decision trees during training and aggregating their results (either by majority voting for classification or averaging for regression) to improve predictive accuracy and control overfitting. The algorithm involves Decision Trees, Bootstrap Aggregation (Bagging), Random Feature Selection, Ensemble Prediction.

Therefore, Random Forest offers an advantage due to its ability to handle non-linear relationships between features and the target variable, making it highly suitable for our health dataset, which includes complex relationships between factors such as age, BMI, average glucose level, and smoking status.

4. Loss function – Logistic Loss

Given that our dataset comprises a large number of features with complex relationships to the target labels, we can use the Logistic Loss function to predict labels based on probability estimates. This function provides a probability score for each data point, enabling classification into one of two binary outcomes. Moreover, the logistic loss is chosen as it allows the use of available library for logistic regression.

For Random Forest classifiers, there isn't a specific "loss function" in the traditional sense, since Random Forest is an ensemble method that uses decision trees. Log-loss can still be used to evaluate its performance by measuring the accuracy of predicted probabilities. It penalizes incorrect classifications more heavily when the model is more confident but wrong.

5. Model validation

To validate the model, we first shuffle the dataset to avoid any ordering biases. This is accomplished using the pandas.DataFrame.sample function [6], which ensures a random sample of the data. After shuffling, we use the function train_test_split in sklearn [7] to divde the data into sets. 80% for training and validating, and 20% for testing. The decision is based on experience and the aim to achieve a balance between having enough data for both training, validating and testing while minimizing bias and variance.

Given that our dataset only contains 5110 data points, which is considered moderate-sized dataset, we can utilize k-fold cross-validation (k = 10) to allow every data point to be used for both training and validation. This leads to the partition in each fold is 72% for training, 8% for validating, and 20% for testing. This approach helps in providing a more robust estimate of model performance and better generalization by leveraging the full dataset for validation while reducing the risk of overfitting or underfitting.

IV. Results

The performance of both the Random Forest and Logistic Regression models was evaluated with a particular focus on **log-loss function** and metric **recall_score**, as correctly identifying individuals at risk of stroke (the positive class) is critical in this medical prediction context. Incorrectly classifying stroke cases as non-stroke (false negatives) could result in high penalties due to missed opportunities for early intervention, making recall a crucial metric in this study. The test set consists of 20% of the data, which is allocated above based on experience to ensure sufficient data for each set. First, we transformed the test set features using Standard Scaler as we had done above with the training set.

After testing with various feature matrices and hyperparameters, we decided that for logistic regression we would use the the hyperparameter C = 0.1, and penalty ‘l2’. Amongst the values for hyperparameters tested, C = 0,1 yields the best results. Similarly, after various testings, for random forest, we use max_features = 2 , n_estimators = 100, and max_depth = 5.

For training and validating process, the mean recall score across the folds was observed to be significantly high for the Random Forest model, with a mean value of **0.9072**, where as the **Logistic Regression** model achieved a mean recall score of **0.8294**. For testing process, the recall score was **0.6428** for Random Forest algorithm and **0.6190** for Logistic Regression. The log-loss error for Random Forest was **8.7816** and for Logistic Regression was **9.2048**.

Hence, selecting the final model for our problem is not difficult, as comparing the outputs of the two methods, Random Forest yields slightly better performance in training and validating set (0.9072>0.8294), testing set (0.6428>0.6190) and loss values (8.7816<9.2048). Thus, the Random Forest algorithm with specific hyperparameters is our final chosen model for this project.

V. Conclusion

This report demonstrates that we attained a test recall score of **0.6458** using Random Forest model, indicating that machine learning can effectively tackle the challenge of predicting stroke susceptibility based on health data. Random Forest was a better choice for Logistic Regression for this classification task probably due to its ability to capture non-linear relationship between features and target variables.

Although Random Forest showed promising results, there is potential for further enhancement. The dataset includes class imbalances and possible outliers that may hinder the model's generalization capabilities. Additionally, some features, such as average glucose level and BMI, might not be directly related to stroke risk, which could affect prediction accuracy.

To improve the predictive performance of our models, future investigations could incorporate a broader range of health metrics or biomarkers that reflect the physiological impacts of stroke risk. By refining our dataset and models, we aim to create a more accurate tool for identifying individuals at risk of stroke, thereby contributing to enhanced prevention strategies in healthcare.

References

[1] Wang, Y., Wang, Y., and Hu, Y. (2018). 'Understanding the risk factors for stroke: a systematic review', PMC6288566. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6288566/ (Accessed: 6 October 2024).

[2] Fedesoriano. (n.d.). Stroke Prediction Dataset. Available at: https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset (Accessed: 6 October 2024).

[3] Scikit-learn. (n.d.). 'SMOTE', Imbalanced-learn documentation. Available at: https://imbalanced-learn.org/stable/references/api.html#imblearn.over_sampling.SMOTE (Accessed: 6 October 2024).

[4] Scikit-learn. (n.d.). 'StandardScaler', Scikit-learn Documentation. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (Accessed: 6 October 2024).

[5] Scikit-learn. (n.d.). 'RandomForestClassifier', Scikit-learn Documentation. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (Accessed: 6 October 2024).

[6] Pandas. (n.d.). 'DataFrame.sample', Pandas Documentation. Available at: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sample.html (Accessed: 6 October 2024).

[7] Scikit-learn. (n.d.). 'train_test_split', Scikit-learn Documentation. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (Accessed: 6 October 2024).

main

October 6, 2024

[669]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from imblearn.over_sampling import SMOTE
from sklearn.model_selection import cross_val_score, KFold, train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import log_loss, recall_score, confusion_matrix,␣
 ↪precision_score, f1_score
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('healthcare-dataset-stroke-data.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
 1   gender             5110 non-null   object
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64
 4   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

[670]:
```
df.head()
```

[670]:
```
      id  gender   age  hypertension  heart_disease ever_married  \
0   9046    Male  67.0             0              1          Yes
1  51676  Female  61.0             0              0          Yes
2  31112    Male  80.0             0              1          Yes
3  60182  Female  49.0             0              0          Yes
4   1665  Female  79.0             1              0          Yes

       work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
0        Private          Urban             228.69  36.6  formerly smoked
1  Self-employed          Rural             202.21   NaN     never smoked
2        Private          Rural             105.92  32.5     never smoked
3        Private          Urban             171.23  34.4           smokes
4  Self-employed          Rural             174.12  24.0     never smoked

   stroke
0       1
1       1
2       1
3       1
4       1
```

1 Data Preprocessing

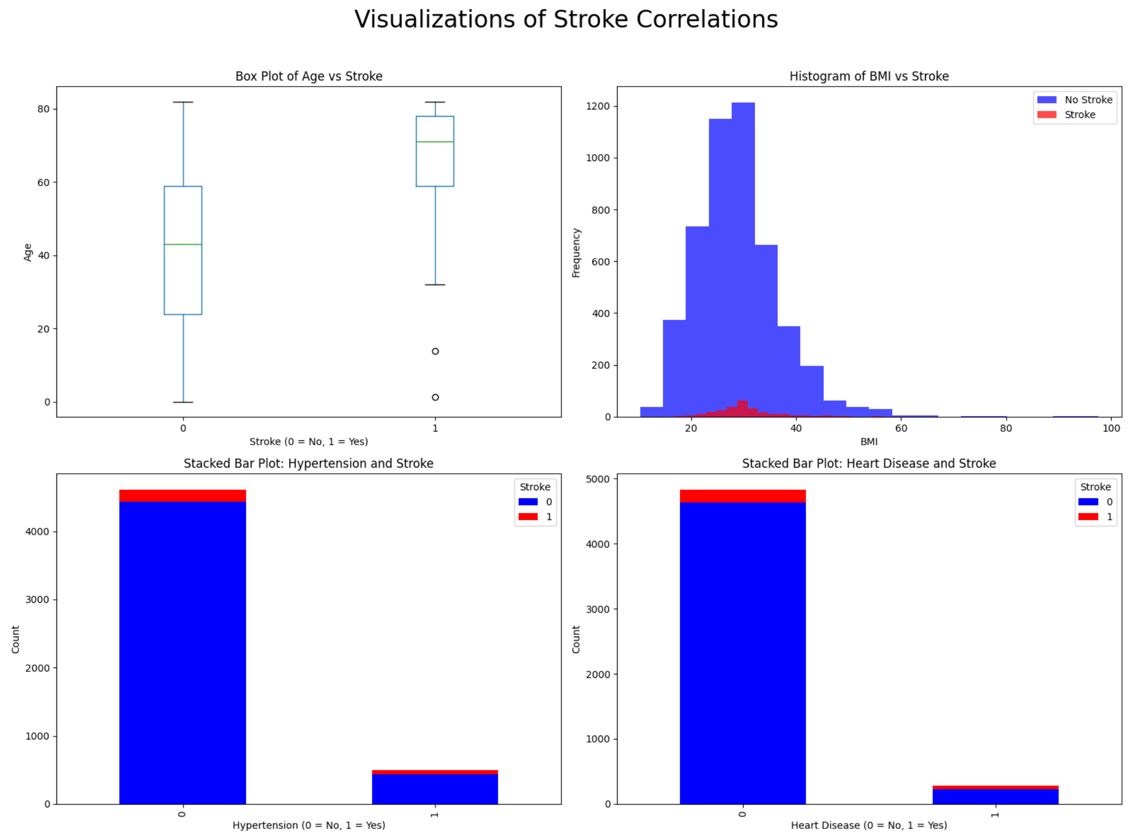
[ ]:
```
# Replace BMI Null values by their mean
mean_bmi = df['bmi'].mean()
df['bmi'].fillna(mean_bmi, inplace = True)
df['bmi'] = df['bmi'].round(2)

#Categorize features gender, residence_type, work_type
df['gender'] = df['gender'].replace({'Male':0,'Female':1,'Other':-1})
df['Residence_type'] = df['Residence_type'].replace({'Rural':0,'Urban':1})
df['work_type'] = df['work_type'].replace({'Private':0,'Self-employed':
 ↪1,'Govt_job':2,'children':3,'Never_worked':4})
df['smoking_status'] =  df['smoking_status'].replace({'never smoked':0,␣
 ↪'formerly smoked':1, 'smokes':2, 'Unknown': 0 })
```

[672]:
```
df = df.sample(frac=1).reset_index(drop=True)
df.head()
```

[672]:
```
      id  gender   age  hypertension  heart_disease ever_married  work_type  \
0  48922       0  55.0             1              1          Yes          0
1  22590       0   5.0             0              0           No          3
2  53882       0  74.0             1              1          Yes          0
3  46436       0  13.0             0              0           No          3
4  15791       0  77.0             0              0          Yes          0
```

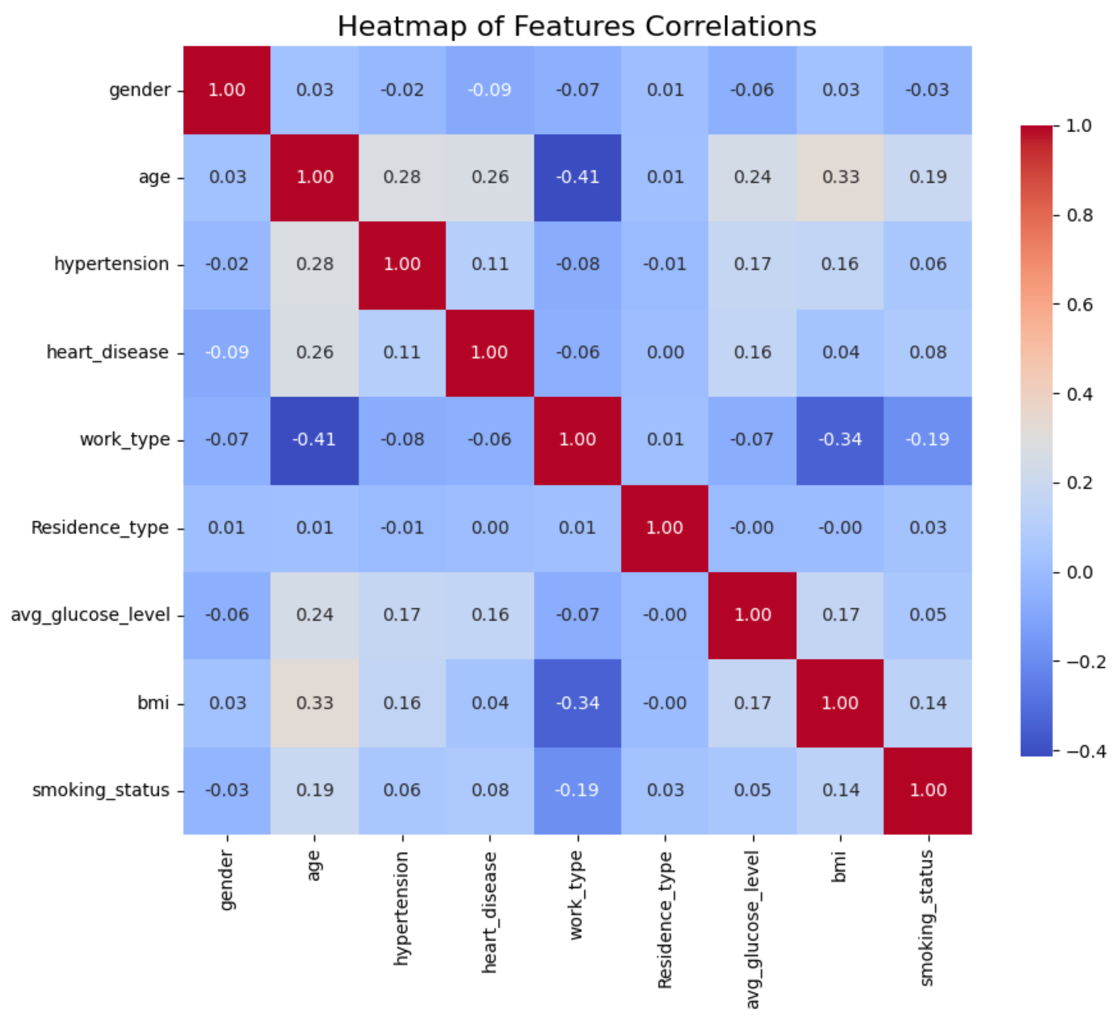
```
   Residence_type  avg_glucose_level   bmi  smoking_status  stroke
0               0              64.92  32.1               2       0
1               1              83.75  18.1               0       0
2               0              70.09  27.4               0       1
3               1             122.31  15.3               0       0
4               1             193.83  26.5               0       0
```

[673]:
```
# Feature engineering by using visualization
fig, axs = plt.subplots(2, 2, figsize=(16, 12))

# Box plot of Age vs Stroke
df.boxplot(column='age', by='stroke', grid=False, ax=axs[0, 0])
axs[0, 0].set_title('Box Plot of Age vs Stroke')
axs[0, 0].set_xlabel('Stroke (0 = No, 1 = Yes)')
axs[0, 0].set_ylabel('Age')

# Histogram of BMI vs Stroke
axs[0, 1].hist(df[df['stroke'] == 0]['bmi'], bins=20, alpha=0.7, label='No␣
 ↪Stroke', color='blue')
axs[0, 1].hist(df[df['stroke'] == 1]['bmi'], bins=20, alpha=0.7,␣
 ↪label='Stroke', color='red')
axs[0, 1].set_title('Histogram of BMI vs Stroke')
axs[0, 1].set_xlabel('BMI')
axs[0, 1].set_ylabel('Frequency')
axs[0, 1].legend()

# Stacked Bar Plot: Hypertension and Stroke
hypertension_stroke = pd.crosstab(df['hypertension'], df['stroke'])
hypertension_stroke.plot(kind='bar', stacked=True, ax=axs[1, 0], color=['blue',␣
 ↪'red'])
axs[1, 0].set_title('Stacked Bar Plot: Hypertension and Stroke')
axs[1, 0].set_xlabel('Hypertension (0 = No, 1 = Yes)')
axs[1, 0].set_ylabel('Count')
axs[1, 0].legend(title='Stroke')

# Stacked Bar Plot: Heart Disease and Stroke
heart_disease_stroke = pd.crosstab(df['heart_disease'], df['stroke'])
heart_disease_stroke.plot(kind='bar', stacked=True, ax=axs[1, 1],␣
 ↪color=['blue', 'red'])
axs[1, 1].set_title('Stacked Bar Plot: Heart Disease and Stroke')
axs[1, 1].set_xlabel('Heart Disease (0 = No, 1 = Yes)')
axs[1, 1].set_ylabel('Count')
axs[1, 1].legend(title='Stroke')

# Adjust layout and display the figure
plt.suptitle('Visualizations of Stroke Correlations', fontsize=24)
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```



[674]:
```
X  = df.drop(columns = ['id', 'ever_married', 'stroke'])
y = df['stroke']
```

[679]:
```
correlation_matrix = X.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm',␣
 ↪square=True, cbar_kws={"shrink": .8})
plt.title('Heatmap of Features Correlations', fontsize=16)
plt.show()
```



	gender	age	hypertension	heart_disease	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
gender	1.00	0.03	-0.02	-0.09	-0.07	0.01	-0.06	0.03	-0.03
age	0.03	1.00	0.28	0.26	-0.41	0.01	0.24	0.33	0.19
hypertension	-0.02	0.28	1.00	0.11	-0.08	-0.01	0.17	0.16	0.06
heart_disease	-0.09	0.26	0.11	1.00	-0.06	0.00	0.16	0.04	0.08
work_type	-0.07	-0.41	-0.08	-0.06	1.00	0.01	-0.07	-0.34	-0.19
Residence_type	0.01	0.01	-0.01	0.00	0.01	1.00	-0.00	-0.00	0.03
avg_glucose_level	-0.06	0.24	0.17	0.16	-0.07	-0.00	1.00	0.17	0.05
bmi	0.03	0.33	0.16	0.04	-0.34	-0.00	0.17	1.00	0.14
smoking_status	-0.03	0.19	0.06	0.08	-0.19	0.03	0.05	0.14	1.00

```
[676]: X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,␣
        ↪random_state=42)
       smote = SMOTE(random_state=42)
       X_train_resh, y_train_resh = smote.fit_resample(X_train, y_train)

       scaler = StandardScaler()
       X_train_scaled = scaler.fit_transform(X_train_resh)

       #Replace hyperparameters to optain the best result
       rf_model =␣
        ↪RandomForestClassifier(max_features=2,n_estimators=100,max_depth=5,random_state=42)
       logreg_model = LogisticRegression(C=0.1,penalty='l2',random_state=42)


       rf_scores = cross_val_score(rf_model, X_train_scaled, y_train_resh, cv=10,␣
        ↪scoring='recall')
```

```
logreg_scores = cross_val_score(logreg_model, X_train_scaled, y_train_resh,␣
 ↪cv=10, scoring='recall')

print('Mean Recall Scores:')
print('Random Forest mean:', rf_scores.mean())
print('Logistic Regression mean:', logreg_scores.mean())
```

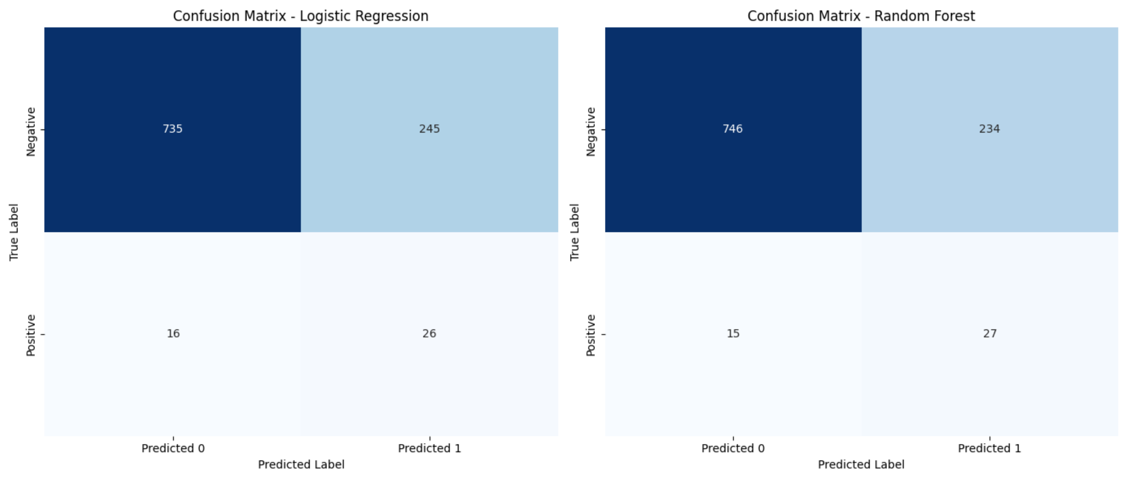
```
Mean Recall Scores:
Random Forest mean: 0.9072337211459466
Logistic Regression mean: 0.8294205337502982
```

[677]:
```
rf_model.fit(X_train_scaled, y_train_resh)
logreg_model.fit(X_train_scaled, y_train_resh)

X_test_scaled = scaler.transform(X_test)

y_predict_rf = rf_model.predict(X_test_scaled)
y_predict_lr = logreg_model.predict(X_test_scaled)

print('Recall score:')
print(f'Random Forest: {recall_score(y_test, y_predict_rf, pos_label=1)}')
print(f'Logistic Regression: {recall_score(y_test, y_predict_lr, pos_label=1)}')

print('\nLog_loss:')
print(f'Random Forest: {log_loss(y_test, y_predict_rf)}')
print(f'Logistic Regression: {log_loss(y_test, y_predict_lr)}')
```

```
Recall score:
Random Forest: 0.6428571428571429
Logistic Regression: 0.6190476190476191

Log_loss:
Random Forest: 8.781672890303494
Logistic Regression: 9.204886041643423
```

[678]:
```
cm_log_reg = confusion_matrix(y_test, y_predict_lr)
cm_rf = confusion_matrix(y_test, y_predict_rf)

fig, axes = plt.subplots(1, 2, figsize=(14, 6))


sns.heatmap(cm_log_reg, annot=True, fmt='d', cmap='Blues', cbar=False,␣
 ↪ax=axes[0],
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Negative', 'Positive'])
axes[0].set_xlabel('Predicted Label')
axes[0].set_ylabel('True Label')
```

```
axes[0].set_title('Confusion Matrix - Logistic Regression')

sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', cbar=False, ax=axes[1],
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Negative', 'Positive'])
axes[1].set_xlabel('Predicted Label')
axes[1].set_ylabel('True Label')
axes[1].set_title('Confusion Matrix - Random Forest')

plt.tight_layout()
plt.show()
```



```
[ ]:
```