Computer Systems Engineering
ENCS5341
Machine Learning and Data Science
2023-2024

Course Project: Machine Learning Models

Students:

Hana Kafri 1190084

Hadeel Abdellatif 1190451

Instructor: Dr.Yazan Abu Farha

Sec.1 26/1 /2024

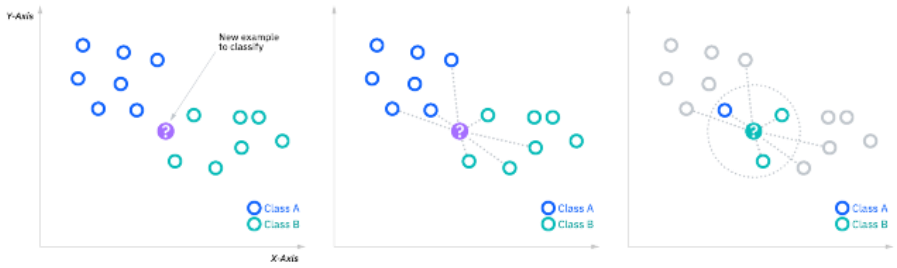
Table of Contents

1. Introduction

This report presents the practical implementation and comparative analysis of three pivotal machine learning models: k-Nearest Neighbors, Logistic Regression, and Random Forest, each enhanced with Cross-Validation techniques. Our study meticulously explores the unique attributes and performance metrics of these models in handling classification tasks. Through comparative analysis and real-world examples, the report aims to provide a clear understanding of how these models operate and their significant role in data analysis.

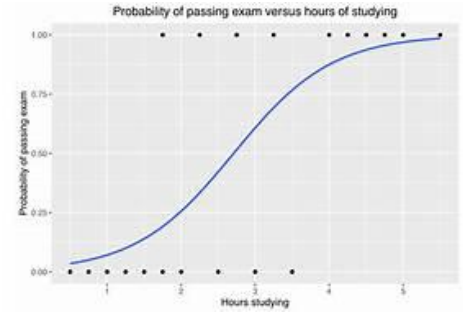
1.1. k-Nearest Neighbors

KNN stands for k-nearest neighbors algorithm, which is a non-parametric, supervised learning classifier used for classification or regression tasks [1]. It works on the principle of similarity, where the algorithm predicts the label or value of a new data point by considering its k closest neighbors in the training data[2]. The value of k is crucial in the KNN algorithm to define the number of neighbors in the algorithm [3]. Larger values of k are often more robust to outliers and produce more stable decision boundaries, which might produce undesirable results. KNN has been utilized within a variety of applications, including data preprocessing, recommendation engines, and pattern recognition[4].



1.2. Logistic Regression

Logistic regression is a supervised machine learning algorithm that is mainly used for binary classification problems, where the goal is to predict the probability that an instance belongs to a given class or not. It is based on a logistic function, also known as a sigmoid function, that maps any real value into a value between 0 and 1. The output of the logistic function can be interpreted as the probability of the instance belonging to the positive class (usually labeled as 1)[5]. Logistic regression is a widely used and powerful classification technique, as it can provide probabilities and classify new data using both continuous and discrete features. Logistic regression is easy to implement and interpret, and can be trained efficiently using various optimization algorithms. Logistic regression is also a good baseline model to compare with other more complex classifiers, such as decision trees, support vector machines, or Random forest [6].



1.3. Random Forest

Random forest is a popular machine learning algorithm that combines the outputs of multiple decision trees to make a single prediction [7]. It is used for both classification and regression tasks. The algorithm is made up of a collection of decision trees, and each tree in the ensemble is trained on a different subset of examples, which ensures the relative independence of the trees and corrects overfitting [8]. Random forest is known for its ease of use, flexibility, and ability to handle noisy datasets. It is widely used in various fields such as finance, healthcare, e-commerce, and more due to its ability to produce reliable results without extensive hyperparameter tuning [7]. The algorithm is also used to determine stock behavior, identify medicine components, analyze patient data, and make product recommendations in e-commerce. Random forest is a versatile and powerful algorithm that is commonly used for both classification and regression tasks [9].

2. Experiments and Results

2.1. Task 1

In this project, we used the Brain Stroke Dataset from Kaggle, which contains data on various risk factors and outcomes of stroke among 5,110 patients. The dataset has 12 columns, including demographic, behavioral, and medical attributes of the patients, as well as a binary variable indicating whether they had a stroke or not.

- chosen dataset: https://www.kaggle.com/datasets/jillanisofttech/brain-stroke-dataset

2.2. Task 2

We note from the results shown in the figure below, that the nearest neighbor (KNN) model exhibits an increase in accuracy as we transition from k = 1 to k = 3, underscoring the significance of considering multiple neighbors during the prediction process. By involving the three nearest neighbors, the model benefits from a broader perspective on the data, which in turn reduces the impact of occasional erroneous data points and introduces a smoothing effect. However, it's important to acknowledge that selecting the optimal k value involves a trade-off between bias and variance. Smaller k values (e.g., k = 1) are more sensitive to local variations but may risk overfitting, whereas larger k values (e.g., k = 3) offer more stability but could potentially underfit the data if chosen excessively.

```
KNN, k=1: Accuracy = 0.9178, Precision = 0.9046, Recall = 0.9178, F1-Score = 0.9109
KNN, k=3: Accuracy = 0.9388, Precision = 0.9058, Recall = 0.9388, F1-Score = 0.9193
Mean of each feature:
age                               43.419859
hypertension                       0.096165
heart_disease                      0.055210
avg_glucose_level                105.943562
bmi                               28.498173
stroke                             0.049789
gender_Male                        0.416382
ever_married_Yes                   0.658502
work_type_Private                  0.574182
work_type_Self-employed            0.161413
work_type_children                 0.135113
Residence_type_Urban               0.508332
smoking_status_formerly smoked     0.174061
smoking_status_never smoked        0.369002
smoking_status_smokes              0.155792
Name: mean, dtype: float64
```

When working with a dataset, we first read it into a pandas DataFrame and pre-processed the data by handling missing values by filling them with the mean of each numeric column. converting categorical variables into a numerical format that can be fed into machine learning algorithms by identifying the categorical values and encode them using one-hot encoding. Then, we split the dataset into features (X) and the target variable (y), with 'stroke' as the target variable indicating whether a person had a stroke or not. The data is then split into training and testing sets using an 80-20 split ratio. We apply a k-nearest neighbors (KNN) classifier with two different values of k (1 and 3) to the training data and evaluate the models' performance on the test data using accuracy, precision, recall, and F1-score metrics to assess the quality of the classification models. Additionally, we compute and display statistics for each feature in the dataset, including the mean, standard deviation, mode, skewness, and kurtosis, providing insights into the central tendency, spread, distribution, and shape of the

data. Finally, we compute and display the correlation matrix of the dataset to understand the relationships between variables and potential multicollinearity in the data.

2.3. Task 3

In our analysis, we compared the performance of Logistic Regression and Random Forest models using cross-validation to determine the most effective approach for our predictive needs. In the case of Logistic Regression, we experimented with varying regularization strengths denoted by the parameter C. We observed that with a lower regularization strength (C = 0.1), and as shown in the figure below, the model achieved an accuracy of 78.17%, a precision of 94.51%, and a recall and F1-score both at 73.17%. Increasing the regularization strength to C = 1 resulted in a slight improvement in all metrics, notably accuracy, which increased to 73.34%, and precision, which increased to 94.59%. Further increases in C to 10 and 100 did not significantly alter these results, with accuracy, precision, recall, and F1-score all plateauing around 73.39%, 94.60%, 73.39%, and 80.95%, respectively. This indicates that while the model benefits marginally from increased regularization strength, there is a limit to the improvement it offers, especially noticeable in the high precision values across all configurations, suggesting the model's proficiency in minimizing false positives.

Contrastingly, the Random Forest model, tested across various numbers of estimators, consistently outperformed the Logistic Regression model. The accuracy of the Random Forest model is consistently high, ranging from approximately 95.03% to 95.13%. This indicates that the Random Forest model is performing well in correctly predicting the target variable. The precision is also consistently high, ranging from approximately 90.50% to 95.37%. This suggests that when the model predicts a positive outcome (stroke), it is generally correct. The recall values are also consistently high, ranging from approximately 95.03% to 95.13%. This means that the model is not capturing a large proportion of actual positive cases. And the F1-Score is high, ranging from approximately 92.71% to 92.76%, indicating a good balance between precision and recall. Overall, the Random Forest model is providing reliable and consistent performance across different numbers of estimators.

In conclusion, the Random Forest model, with either 50, 100, or 200 estimators, is preferable due to its superior performance metrics. It exhibits robustness and effectiveness in predictive accuracy, which is crucial for our requirements.

```
Model 1: Logistic Regression with Cross-Validation
Logistic Regrision (C=0.1): Accuracy = 0.7317, Precision = 0.9451, Recall = 0.7317, F1-Score = 0.8079
Logistic Regrision (C=1): Accuracy = 0.7334, Precision = 0.9459, Recall = 0.7334, F1-Score = 0.8092
Logistic Regrision (C=10): Accuracy = 0.7339, Precision = 0.9460, Recall = 0.7339, F1-Score = 0.8095
Logistic Regrision (C=100): Accuracy = 0.7339, Precision = 0.9460, Recall = 0.7339, F1-Score = 0.8095

Model 2: Random Forest with Cross-Validation
Random Forest (n_estimators=50): Accuracy = 0.9508, Precision = 0.9050, Recall = 0.9508, F1-Score = 0.9273
Random Forest (n_estimators=100): Accuracy = 0.9503, Precision = 0.9049, Recall = 0.9503, F1-Score = 0.9271
Random Forest (n_estimators=150): Accuracy = 0.9511, Precision = 0.9050, Recall = 0.9511, F1-Score = 0.9274
Random Forest (n_estimators=200): Accuracy = 0.9513, Precision = 0.9537, Recall = 0.9513, F1-Score = 0.9276
```

2.4. Task 4

In this task, the best model was chosen among the previous models based on the results, and random forest was chosen since it had the highest accuracy. The model was trained using the best n estimator which is 200, using the entire training set.

As shown in the figure below, the model achieved an accuracy of approximately 89.87%. Which is the proportion of correctly classified instances among the total instances. It shows that the model performs well in terms of overall correctness. The model suggests that when the model predicts a positive class, it is correct around 89.41% of the time, which is the precision. It also achieved a recall of 89.87%, which is the ratio of correctly predicted positive observations to all observations in the actual class; the recall for the positive class (stroke) is relatively low. This means that the model is not capturing a large proportion of actual positive cases. And achieved a F1-score of 89.64%, which is the harmonic mean of precision and recall. It provides a balance between precision and recall. The AUC-ROC is a metric that evaluates the model's ability to distinguish between positive and negative classes. A value of 0.7704 indicates a reasonable performance, but the interpretation depends on the specific context and the desired trade-off between true positive and false positive rates.

```
Model: Random Forest with Cross-Validation
Random Forest (Parameter=50): Accuracy = 0.9533, Precision = 0.9536, Recall = 0.9533, F1-Score = 0.9533
Random Forest (Parameter=100): Accuracy = 0.9526, Precision = 0.9531, Recall = 0.9526, F1-Score = 0.9526
Random Forest (Parameter=150): Accuracy = 0.9529, Precision = 0.9534, Recall = 0.9529, F1-Score = 0.9529
Random Forest (Parameter=200): Accuracy = 0.9536, Precision = 0.9540, Recall = 0.9536, F1-Score = 0.9536
Best parameter: 200
Random Forest (using the entire training set) (Parameter=200): Accuracy = 0.8987, Precision = 0.8941, Recall = 0.8987, F1-Score = 0.8964
AUC-ROC: 0.7704
```

The results also shows the following:

- Confusion Matrix:

The confusion matrix breaks down the number of true positives, true negatives, false positives, and false negatives.

- ➔ True Positives (1): The number of instances where the model correctly predicted the positive class.
- ➔ True Negatives (895): The number of instances where the model correctly predicted the negative class.
- ➔ False Positives (48): The number of instances where the model predicted the positive class, but it was actually negative.
- ➔ False Negatives (53): The number of instances where the model predicted the negative class, but it was actually positive.

It reveals that the model is performing well in correctly identifying instances of class 0, but struggles with class 1. There are a small number of true positives and a relatively high number of false negatives.

Given these metrics, the model appears to have high specificity (ability to correctly identify negatives) but low precision and recall for positive instances. This suggests that the model is good at identifying negatives but struggles with identifying positives. Further

investigation and potential adjustments to the model may be necessary to improve its performance on positive instances.

- Classification Report:

➔ The classification report provides a detailed breakdown of precision, recall, and F1-Score for each class (0 and 1). In this case, class 0 (no stroke) has high precision, recall, and F1-Score, while class 1 (stroke) has lower values, suggesting challenges in predicting the positive class.

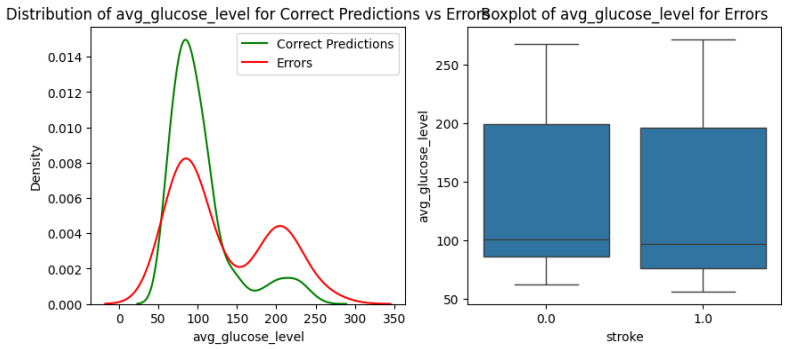
```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.95      0.95       943
           1       0.02      0.02      0.02        54

    accuracy                           0.90       997
   macro avg       0.48      0.48      0.48       997
weighted avg       0.89      0.90      0.90       997


Confusion Matrix:
[[895  48]
 [ 53   1]]
True Positives: 1
True Negatives: 895
False Positives: 48
False Negatives: 53
Feature Importances:
```

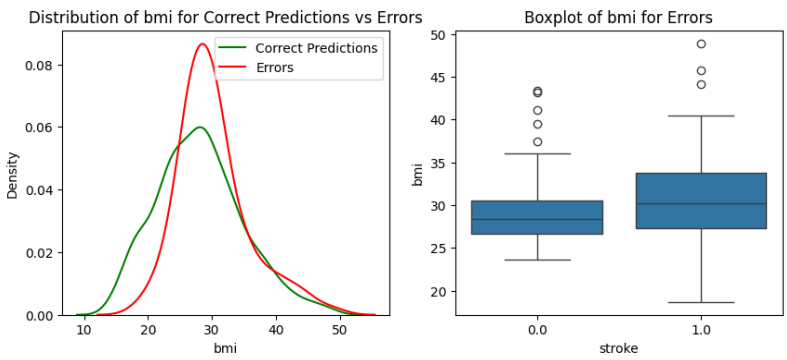
- Error instances summary:

➔ The model seems to perform well in terms of overall accuracy but faces challenges in predicting the positive class (stroke, class 1), as reflected in the low precision, recall, and F1-score for class 1. This suggests that the model might struggle to identify instances of stroke effectively.

The results also plot the distribution of features for correct predictions vs errors (KDE) and boxplot of features for errors, as shown in the figures below, to compare the distributions of selected features (‘age’, ‘avg_glucose_level’, and ‘bmi’) between correct predictions and errors made by the Random Forest model.

The KDE plot shows the distribution of the selected feature for instances where the model made correct predictions (labeled as 'Correct Predictions' in green) and where errors were made (labeled as 'Errors' in red). While if the distributions for correct predictions and errors overlap, such in the ‘age’ and ‘bmi’ features, it suggests that the feature alone may not be a strong discriminator between the two classes. If there's a noticeable difference such in the ‘avg_glucose_level’ feature, it indicates that the feature has some discriminatory power.





3. Analysis

When choosing between machine learning algorithms, we chose *Random Forest* since it is a powerful algorithm for classification and regression tasks. It is known for its ability to handle large amounts of data with high dimensionality, works well with both numerical and categorical features, and is less prone to overfitting compared to decision trees. Additionally, Random Forest provides a feature importance score, which can be helpful in understanding the impact of different features on the model's predictions. Also, we chose *Logistic Regression* since it is a simple and interpretable algorithm that is easy to implement and provides probabilities for outcomes. It works well when the relationship between the dependent variable and the independent variables is linear. Logistic Regression is also less computationally intensive compared to Random Forest, making it faster to train and use for predictions.

After trying the 3 different machine learning algorithms, KNN, Random forest, and Logistic Regression, the Random Forest algorithm got the highest results, it outperformed KNN in terms of accuracy and Logistic Regression in terms of accuracy, precision, recall, and F1-Score. The highest accuracy of KNN, Random Forest, and Logistic Regression are 93.88%, 95.13%, and 73.39% in order. And we notice that the Random FOrest got the highest accuracy.

On the other hand, when comparing the models, Random Forest generally outperforms KNN with k=1 and k=3, as well as Logistic Regression. The Random Forest model with n_estimators=200 achieved the highest accuracy, precision, recall, and F1-Score among all the models compared. Therefore, based on the provided results, the Random Forest model with n_estimators=200 is the better model for this classification task. In conclusion, Random Forest is a suitable choice when working with complex datasets and aiming for high predictive accuracy. However, it's important to consider the specific characteristics of the dataset and the interpretability of the model when making a final decision.

Although the Random Forest model performs well in terms of overall accuracy, its ability to correctly identify instances of the positive class (stroke) is limited. The imbalanced nature of the classes (with a small number of positive cases) may contribute to the challenges in predicting strokes. It is good at identifying negatives but struggles with identifying positives. So that we conclude that the model is not enough to be trained on an imbalanced dataset we provided, it needs additional steps to address challenges associated with the imbalanced dataset and improve its ability to predict the positive class. In the case of imbalanced datasets, achieving high precision might be easier because the model can achieve high accuracy by simply predicting the majority class most of the time.

4. Conclusion

In conclusion, we compared three different machine learning algorithms, KNN, logistic regression and random forest, to see which one is better for predicting outcomes based on data. Logistic regression was pretty accurate and especially good at not making false predictions, but it reached a point where making it stronger didn't really improve its predictions. On the other side, the random forest method was better than logistic regression in every important way - it was more accurate, made fewer false predictions, and was better at identifying correct outcomes. The random forest was particularly good because it balanced well between not making false predictions and correctly identifying true ones. Additionally, when compared with KNN models (k=1 and k=3), Random Forest again demonstrated superior performance. While KNN models showed competitive results, especially with k=3, their performance was not on par with Random Forest in the evaluation metrics. This could be attributed to KNN's sensitivity to the choice of k and its susceptibility to the curse of dimensionality in complex datasets. This project really showed us how important it is to pick the right method and adjust it properly to get the best results. The random forest method was really good at dealing with complicated data and making the right predictions, but its ability to correctly identify instances of the positive class in the imbalanced dataset was limited. What we learned from comparing these three methods is that it is really useful for future projects like this, where we have to choose and adjust methods for predicting outcomes.

5. References

[1]: https://www.ibm.com/topics/knn
[2]: https://www.geeksforgeeks.org/k-nearest-neighbours/
[3]: https://www.w3schools.com/python/python_ml_knn.asp
[4]: https://learn.g2.com/k-nearest-neighbor
[5]: https://www.geeksforgeeks.org/understanding-logistic-regression/
[6]:https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[7]: https://www.ibm.com/topics/random-forest
[8]: https://en.wikipedia.org/wiki/Random_forest
[9]: https://builtin.com/data-science/random-forest-algorithm

6. Appendix

6.1. Task 2:

```
from google.colab import drive
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.preprocessing import OneHotEncoder
from scipy.stats import mode, skew, kurtosis
from sklearn.metrics import precision_score, recall_score, f1_score

# Mount Google Drive
drive.mount('/content/drive')

file_path = '/content/drive/MyDrive/ML_Project/brain_stroke.csv'
df = pd.read_csv(file_path)

# Pre-prossing the data:

# Handling Missing Values: Fill missing values with the mean of the column
df.fillna(df.mean(numeric_only=True), inplace=True)
#df.fillna(df.mean(), inplace=True)

# Identify categorical columns
categorical_columns = df.select_dtypes(include=['object']).columns

# One-hot encode categorical columns
df = pd.get_dummies(df, columns=categorical_columns, drop_first=True)

# Display the first few rows of the dataset to verify
df.head()

# Split data
X = df.drop('stroke', axis=1)
y = df['stroke']

# Split the encoded dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

for k in [1, 3]:
    model = KNeighborsClassifier(n_neighbors=k, metric='euclidean')
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
```

```
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    print(f"KNN, k={k}: Accuracy = {accuracy:.4f}, Precision = {precision:.4f}, Recall = {recall:.4f}, F1-Score = {f1:.4f}")

# Display statistics for each feature in the dataset
feature_statistics = df.describe()

# Display the mean, standard deviation, and other statistics
print("Mean of each feature:")
print(feature_statistics.loc['mean'])

print("\nStandard deviation of each feature:")
print(feature_statistics.loc['std'])

# Display mode, skewness, and kurtosis for each feature in the dataset
print("Mode of each feature:")
print(df.mode().iloc[0])

print("\nSkewness of each feature:")
print(df.apply(skew))

print("\nKurtosis of each feature:")
print(df.apply(kurtosis))

# Display correlation matrix
correlation_matrix = df.corr()
print("\nCorrelation Matrix:")
print(correlation_matrix)
```

6.2. Task 3:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_predict
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')
```

```
file_path = '/content/drive/MyDrive/ML_Project/brain_stroke.csv'
df = pd.read_csv(file_path)

# Handling Missing Values: Fill missing values with the mean of the column
df.fillna(df.mean(numeric_only=True), inplace=True)

# Identify categorical columns excluding the target variable
categorical_columns = df.select_dtypes(include=['object']).columns
df = pd.get_dummies(df, columns=categorical_columns, drop_first=True)

# Display the first few rows of the dataset to verify
df.head()

# Split data
X = df.drop('stroke', axis=1)
y = df['stroke']

# Scale the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

def evaluate_model(y_train, y_pred, model_name, Parameter_name, parameter):
    accuracy = accuracy_score(y_train, y_pred)
    precision = precision_score(y_train, y_pred, average='weighted', zero_division=1)
    recall = recall_score(y_train, y_pred, average='weighted', zero_division=1)
    f1 = f1_score(y_train, y_pred, average='weighted', zero_division=1)
    print(f"{model_name} ({Parameter_name}={parameter}): Accuracy = {accuracy:.4f}, Precision =
{precision:.4f}, Recall = {recall:.4f}, F1-Score = {f1:.4f}")
    return accuracy, y_pred

# Model 1: Logistic Regression with Cross-Validation
print("Model 1: Logistic Regression with Cross-Validation")
for C in [0.1, 1, 10, 100]:
    model1 = LogisticRegression(C=C, class_weight='balanced', penalty='l2', solver='lbfgs') # add Loss
Function: penalty='l2' to indicate L2 regularization
    model1.fit(X_train, y_train)
    y_pred = cross_val_predict(model1, X_train, y_train, cv=3)  # 3-fold cross-validation
    accuracy, _ = evaluate_model(y_train, y_pred, "Logistic Regrision", "C", C)
```

```
# Model 2: Random Forest with Cross-Validation
print("\nModel 2: Random Forest with Cross-Validation")
for n_estimators in [50, 100, 150, 200]:
    model2 = RandomForestClassifier(n_estimators=n_estimators, class_weight='balanced', random_state=42)
    model2.fit(X_train, y_train)
    y_pred = cross_val_predict(model2, X_train, y_train, cv=3)  # 3-fold cross-validation
    accuracy, _ = evaluate_model(y_train, y_pred, "Random Forest", "n_estimators", n_estimators)
```

6.3. Task 4:

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_predict
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
from google.colab import drive
from sklearn.metrics import roc_auc_score
from imblearn.over_sampling import SMOTE
# Mount Google Drive
drive.mount('/content/drive')

file_path = '/content/drive/MyDrive/ML_Project/brain_stroke.csv'
df = pd.read_csv(file_path)

# Handling Missing Values: Fill missing values with the mean of the column
df.fillna(df.mean(numeric_only=True), inplace=True)

# Identify categorical columns excluding the target variable
categorical_columns = df.select_dtypes(include=['object']).columns
df = pd.get_dummies(df, columns=categorical_columns, drop_first=True)
# Display the first few rows of the dataset to verify
df.head()

# Split data
```

```
X = df.drop('stroke', axis=1)
y = df['stroke']

# Scale the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Apply SMOTE to oversample the minority class
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)

def train_random_forest(X_train, y_train, n_estimators):
    model = RandomForestClassifier(n_estimators=n_estimators, class_weight='balanced',
random_state=42)
    y_pred = cross_val_predict(model, X_train, y_train, cv=3)  # 3-fold cross-validation
    return y_pred, model

def evaluate_model(y_true, y_pred, model_name, parameter):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted', zero_division=1)
    recall = recall_score(y_true, y_pred, average='weighted', zero_division=1)
    f1 = f1_score(y_true, y_pred, average='weighted', zero_division=1)
    print(f"{model_name} (Parameter={parameter}): Accuracy = {accuracy:.4f}, Precision =
{precision:.4f}, Recall = {recall:.4f}, F1-Score = {f1:.4f}")
    return accuracy, y_pred

# Model: Random Forest with Cross-Validation
best_n_estimators = None
best_accuracy = 0

print("\nModel: Random Forest with Cross-Validation")
for n_estimators in [50, 100, 150, 200]:
    y_pred, _ = train_random_forest(X_train, y_train, n_estimators)
    accuracy, _ = evaluate_model(y_train, y_pred, "Random Forest", n_estimators)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_n_estimators = n_estimators
```

```
print(f"Best parameter: {best_n_estimators}")
# Train the final model using the entire training set with the best parameter
model2 = RandomForestClassifier(n_estimators=best_n_estimators, class_weight='balanced',
random_state=42)
model2.fit(X_train, y_train)

# Evaluate on the Test Set
accuracy_test, y_pred_test = evaluate_model(y_test, model2.predict(X_test), "Random Forest (using the
entire training set)", best_n_estimators)

# Analyze errors
errors = y_test != y_pred_test
error_indices = errors[errors].index
error_instances = pd.DataFrame(X_test[errors, :], columns=X.columns)

# Predict probabilities for the positive class (stroke)
y_pred_proba = model2.predict_proba(X_test)[:, 1]  # Get probabilities for the positive class

# Example of evaluating AUC-ROC
y_pred_proba_rf = model2.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, y_pred_proba_rf)
print(f"AUC-ROC: {roc_auc:.4f}")
print ("----------------------------------------------------------------------------------------------------")
# Classification Report and Confusion Matrix
print("Classification Report:")
print(classification_report(y_test, y_pred_test))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred_test))

# Extract confusion matrix values
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_test).ravel()

# Print values
print(f"True Positives: {tp}")
print(f"True Negatives: {tn}")
print(f"False Positives: {fp}")
print(f"False Negatives: {fn}")

# Feature Importance Analysis
feature_importances = model2.feature_importances_
importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})
```

```
importance_df = importance_df.sort_values(by='Importance', ascending=False)
print("Feature Importances:")
print(importance_df)
print("")

print ("error instances:")
# Print first few rows of error instances
print(error_instances.head())
print ("error instances summary:")

# Explore summary statistics of error instances
print(error_instances.describe())

# Analyzing Error Instances
error_stats = error_instances.describe()
print(error_stats)

# Compare the distributions of errors vs correct predictions for key features
selected_features = ['age', 'avg_glucose_level', 'bmi']  # replace with features of interest
for feature_name in selected_features:
    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    sns.kdeplot(df.loc[y_test.index[~errors], feature_name], color='green', label='Correct Predictions')
    sns.kdeplot(df.loc[y_test.index[errors], feature_name], color='red', label='Errors')
    plt.title(f'Distribution of {feature_name} for Correct Predictions vs Errors')
    plt.legend()

    plt.subplot(1, 2, 2)
    sns.boxplot(x=y_test[errors], y=df.loc[y_test.index, feature_name])
    plt.title(f'Boxplot of {feature_name} for Errors')
    plt.show()
# Visualize class distribution in errors
plt.figure(figsize=(6, 4))
sns.countplot(x=y_test[errors])
plt.title('Distribution of Errors by Class')
plt.show()
```