Stroke Prediction Project Report

Ahmed Tarek Mahmoud 2100561 Mark Saleh Sobhi 21P0206

CSE381: Introduction to Machine Learning
April 22, 2025

Contents

1 Introduction

In this project, we aim to predict the occurrence of stroke events using clinical data. We will walk through each step of our process: data loading, exploration, preprocessing, baseline modeling, hyperparameter tuning, and final evaluation. For reproducibility, we include the code for each Jupyter notebook cell, detailed explanations of its purpose, and placeholders where screenshots of outputs should be inserted.

2 Dataset Overview and Initial Inspection

We begin by loading the dataset and performing an initial inspection to understand its shape and content.

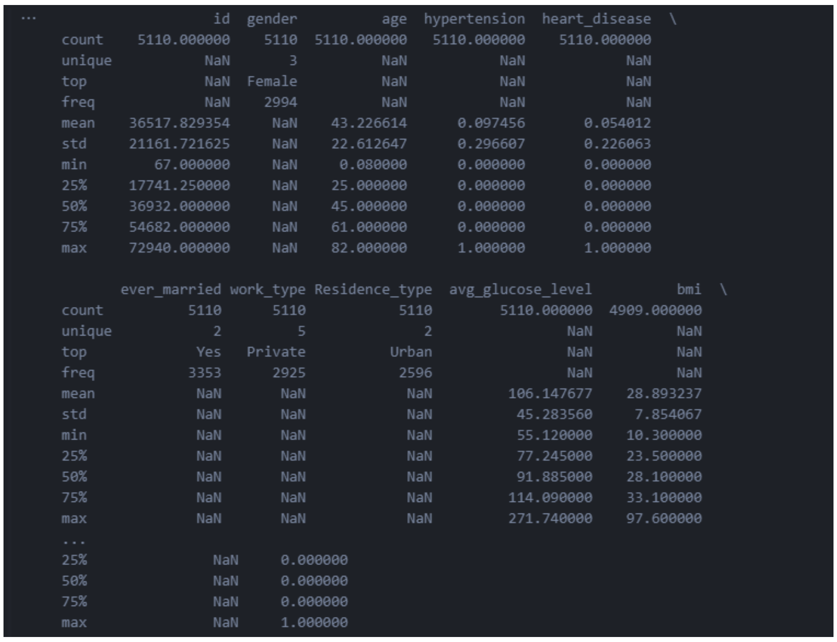
2.1 Loading the Data

Explanation: We import pandas and read the CSV file into a DataFrame. We then print the shape and the first few rows to verify successful loading.

```python
import pandas as pd

df = pd.read_csv('healthcare-dataset-stroke-data.csv')
print(f"Dataset shape: {df.shape}")  # Rows, Columns
print(df.head())  # Display first 5 records
```

Listing 1: Load dataset and preview



```
...                  id  gender         age  hypertension  heart_disease  \
    count    5110.000000    5110  5110.000000   5110.000000    5110.000000
    unique           NaN       3          NaN           NaN            NaN
    top              NaN  Female          NaN           NaN            NaN
    freq             NaN    2994          NaN           NaN            NaN
    mean    36517.829354     NaN    43.226614      0.097456       0.054012
    std     21161.721625     NaN    22.612647      0.296607       0.226063
    min        67.000000     NaN     0.080000      0.000000       0.000000
    25%     17741.250000     NaN    25.000000      0.000000       0.000000
    50%     36932.000000     NaN    45.000000      0.000000       0.000000
    75%     54682.000000     NaN    61.000000      0.000000       0.000000
    max     72940.000000     NaN    82.000000      1.000000       1.000000

           ever_married work_type Residence_type  avg_glucose_level          bmi  \
    count          5110      5110           5110        5110.000000  4909.000000
    unique            2         5              2                NaN          NaN
    top             Yes   Private          Urban                NaN          NaN
    freq           3353      2925           2596                NaN          NaN
    mean            NaN       NaN            NaN         106.147677    28.893237
    std             NaN       NaN            NaN          45.283560     7.854067
    min             NaN       NaN            NaN          55.120000    10.300000
    25%             NaN       NaN            NaN          77.245000    23.500000
    50%             NaN       NaN            NaN          91.885000    28.100000
    75%             NaN       NaN            NaN         114.090000    33.100000
    max             NaN       NaN            NaN         271.740000    97.600000
    ...
    25%               NaN      0.000000
    50%               NaN      0.000000
    75%               NaN      0.000000
    max               NaN      1.000000
```

Figure 1: Screenshot: Output showing dataset dimensions and first few rows.

3 Data Exploration

We explore the dataset to identify missing values, distributions, and relationships between features.

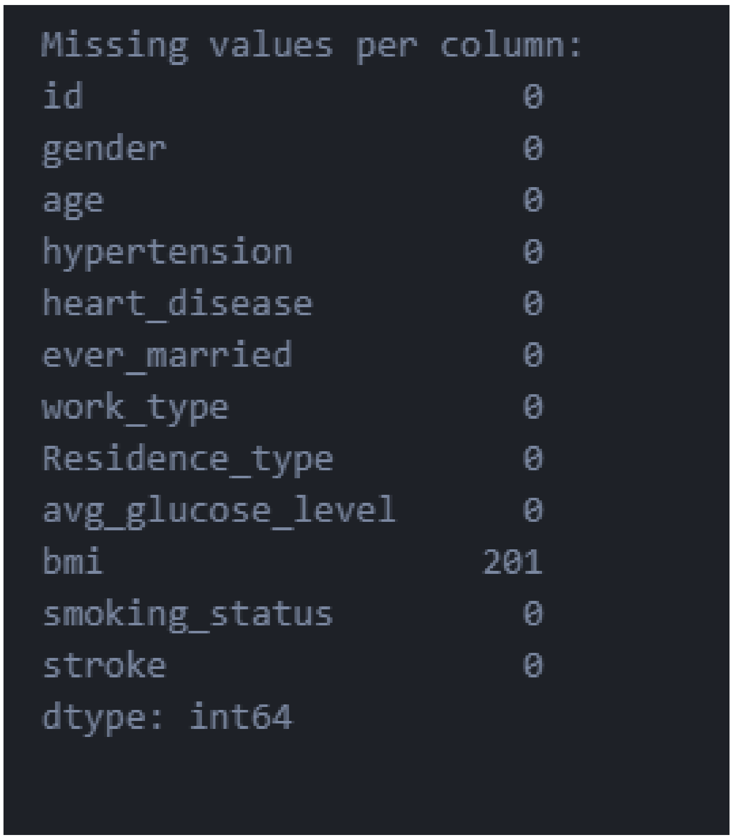
3.1 Descriptive Statistics and Missing Values

Explanation: We compute summary statistics for all columns and count missing values to guide our imputation strategy.

```python
# Summary statistics
print(df.describe(include='all'))

# Missing value counts
print("Missing values per column:")
print(df.isnull().sum())
```

Listing 2: Compute statistics and missing counts



```
Missing values per column:
id                    0
gender                0
age                   0
hypertension          0
heart_disease         0
ever_married          0
work_type             0
Residence_type        0
avg_glucose_level     0
bmi                 201
smoking_status        0
stroke                0
dtype: int64
```

Figure 2: Screenshot: Summary statistics and missing value counts.

3.2 Target Class Distribution

Explanation: We visualize the balance between stroke and non-stroke cases using a count plot.

```python
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x='stroke', data=df)
plt.title('Stroke vs Non-Stroke Counts')
plt.show()
```

Listing 3: Visualize class distribution

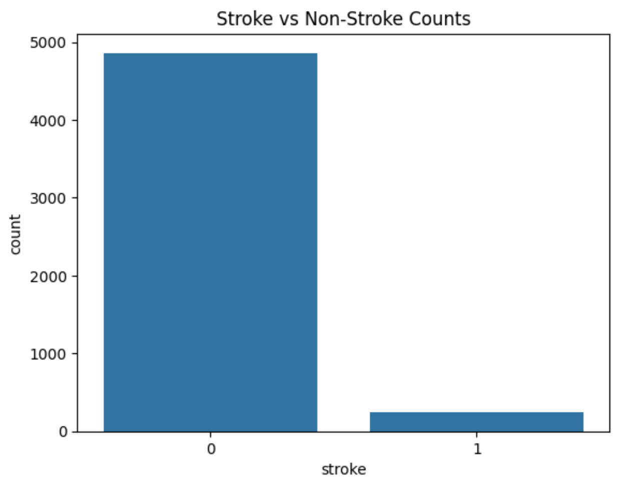


Figure 3: Screenshot: Count of stroke vs. non-stroke cases.

3.3 Feature Correlation and Scatter Plots

Explanation: We examine correlations among numeric features and plot age versus glucose level colored by stroke outcome.

```python
numeric_cols = df.select_dtypes(include=['int64','float64']).columns
import numpy as np
# Correlation heatmap
corr = df[numeric_cols].corr()
sns.heatmap(corr, annot=True, fmt='.2f')
plt.title('Correlation Matrix')
plt.show()

# Scatter: age vs average glucose
sns.scatterplot(x='age', y='avg_glucose_level', hue='stroke', data=df)
plt.title('Age vs Glucose Level by Stroke')
plt.show()
```

Listing 4: Correlation heatmap and scatter plot

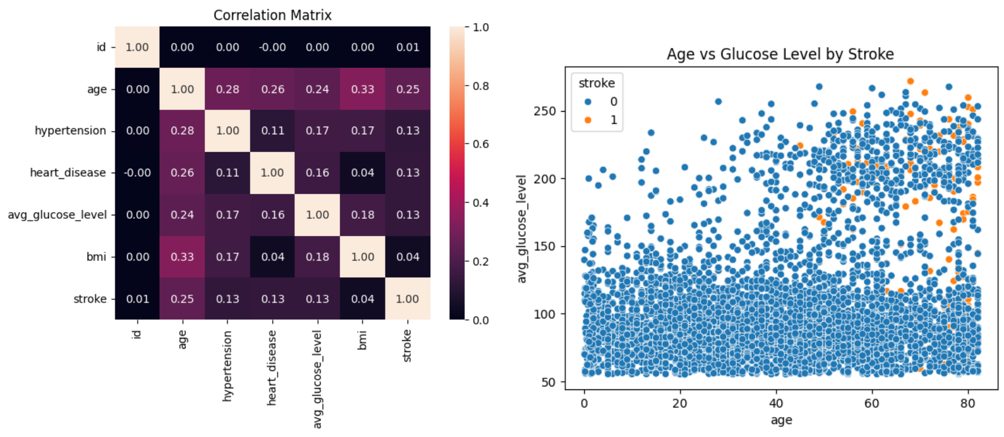


Figure 4: Screenshots: Left, correlation heatmap; Right, age vs. glucose by stroke.

4 Dimensionality Reduction

To visualize high-dimensional relationships, we apply PCA, LDA, and t-SNE.

4.1 Preparation for Projections

Explanation: We drop the `id` column, remove missing entries, one-hot encode categoricals, and standardize features.

```python
from sklearn.preprocessing import StandardScaler

df_dr = df.drop('id', axis=1).dropna()
x_dr = pd.get_dummies(df_dr.drop('stroke', axis=1),
    drop_first=True)
y_dr = df_dr['stroke']
X_scaled = StandardScaler().fit_transform(x_dr)
```

Listing 5: Prepare data for projections

4.2 PCA, LDA, and t-SNE

Explanation: We project the standardized data into 2D spaces and plot to observe any natural separation.

```python
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import
    LinearDiscriminantAnalysis as LDA
```

```python
from sklearn.manifold import TSNE

# PCA
pca_proj = PCA(n_components=2).fit_transform(X_scaled)

# LDA
lda_proj = LDA(n_components=1).fit_transform(X_scaled,
    y_dr)

# t-SNE
tsne_proj = TSNE(n_components=2, random_state=42).
    fit_transform(X_scaled)

# Plotting omitted for brevity
```

Listing 6: Compute and plot projections

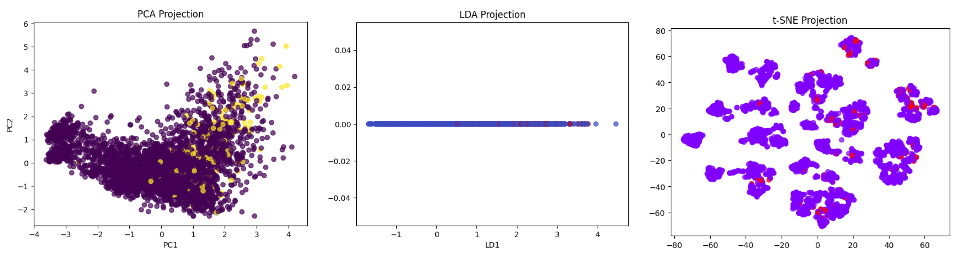


Figure 5: Screenshots: PCA (left), LDA (center), t-SNE (right) projections.

5 Data Preprocessing

We clean and prepare data for modeling.

5.1 Handling Missing Values and Invalid Entries

Explanation: We impute missing `bmi` values with the median and remove records with non-positive age or glucose because they are biologically invalid.

```python
# Impute BMI
bmi_median = df['bmi'].median()
df['bmi'].fillna(bmi_median, inplace=True)

# Remove invalid records
df = df[(df['age'] > 0) & (df['avg_glucose_level'] > 0)]
```

Listing 7: Impute and filter data

5.2 Encoding and Scaling

Explanation: We encode categorical variables with LabelEncoder, split into training and test sets (stratified by stroke), and standardize features.

```python
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Encode categoricals
categorical = ['gender','ever_married','work_type','
    Residence_type','smoking_status']
for col in categorical:
    df[col] = LabelEncoder().fit_transform(df[col])

# Split
X = df.drop(['id','stroke'], axis=1)
y = df['stroke']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# Scale
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Listing 8: Encode and scale data

6 Baseline Models

We establish baseline performance using Naïve Bayes and default SVM.

6.1 Gaussian Naïve Bayes

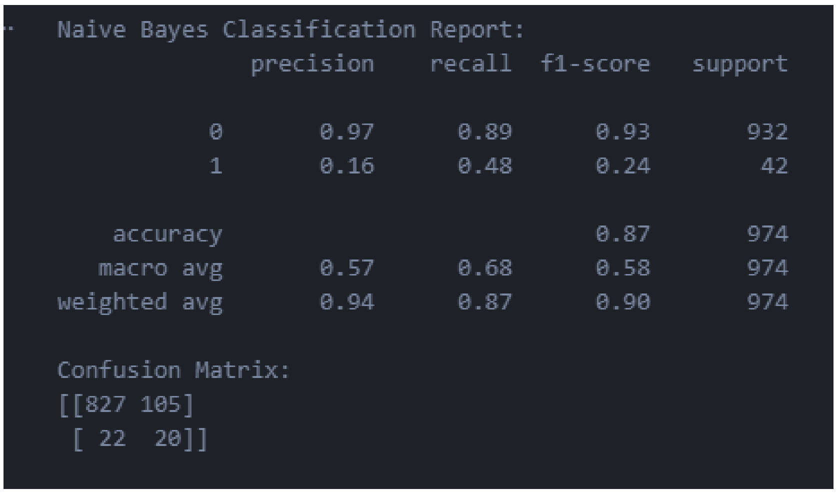
Explanation: We train and evaluate a GaussianNB classifier to get a reference accuracy and classification metrics.

```python
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
```

```python

nb = GaussianNB()
nb.fit(X_train_scaled, y_train)
y_nb_pred = nb.predict(X_test_scaled)
print(classification_report(y_test, y_nb_pred))
```

Listing 9: Train and evaluate Naïve Bayes



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

           0       0.97      0.89      0.93       932
           1       0.16      0.48      0.24        42

    accuracy                           0.87       974
   macro avg       0.57      0.68      0.58       974
weighted avg       0.94      0.87      0.90       974

Confusion Matrix:
[[827 105]
 [ 22  20]]
```

Figure 6: Screenshot: Naïve Bayes classification report.

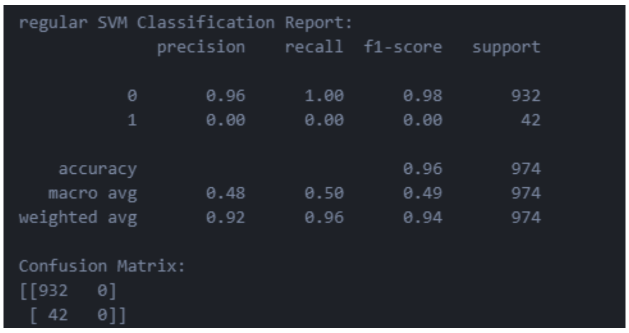
6.2 Default Support Vector Machine

Explanation: We train an SVM with default parameters and measure its accuracy as a second baseline.

```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

svc = SVC(random_state=42)
svc.fit(X_train_scaled, y_train)
y_svc_pred = svc.predict(X_test_scaled)
print(f"Default SVM Test Accuracy: {accuracy_score(
    y_test, y_svc_pred):.2f}")
```

Listing 10: Train and evaluate default SVM



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

           0       0.96      1.00      0.98       932
           1       0.00      0.00      0.00        42

    accuracy                           0.96       974
   macro avg       0.48      0.50      0.49       974
weighted avg       0.92      0.96      0.94       974

Confusion Matrix:
[[932   0]
 [ 42   0]]
```

Figure 7: Screenshot: Default SVM test accuracy.

7 Hyperparameter Tuning

We improve SVM performance by tuning `C` and `kernel` via grid search.

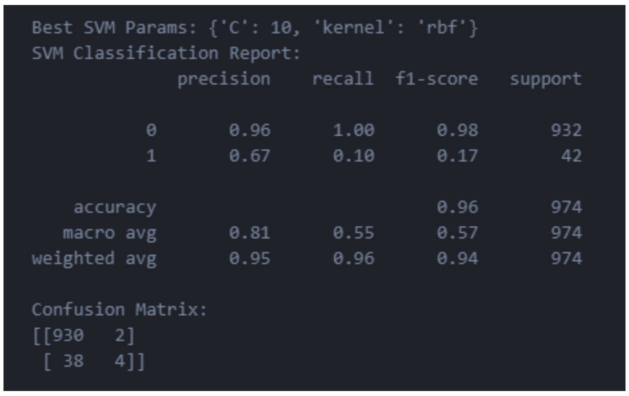
7.1 Grid Search Setup

Explanation: We define a parameter grid and perform 5-fold cross-validation optimizing the F1-score.

```python
from sklearn.model_selection import GridSearchCV

svc = SVC(probability=True, random_state=42)
param_svc = {'C':[0.1,1,10], 'kernel':['linear','rbf']}
grid_svc = GridSearchCV(svc, param_svc, cv=5, scoring='
    f1')
grid_svc.fit(X_train_scaled, y_train)
best_svc = grid_svc.best_estimator_
print("Best SVM Params:", grid_svc.best_params_)
y_pred_svm = best_svc.predict(X_test_scaled)
print("SVM Classification Report:")
print(classification_report(y_test, y_pred_svm))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_svm))
```

Listing 11: Grid search for SVM hyperparameters



```
Best SVM Params: {'C': 10, 'kernel': 'rbf'}
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       932
           1       0.67      0.10      0.17        42

    accuracy                           0.96       974
   macro avg       0.81      0.55      0.57       974
weighted avg       0.95      0.96      0.94       974

Confusion Matrix:
[[930   2]
 [ 38   4]]
```

Figure 8: Screenshot: Grid search results for SVM hyperparameters.

7.2 Visualization of Accuracy Change

Explanation: We plot mean F1-score against `C` for both linear and RBF kernels to visualize the tuning landscape.

```python
import matplotlib.pyplot as plt
import pandas as pd

results = pd.DataFrame(grid_svc.cv_results_)
for kern in ['linear','rbf']:
    subset = results[results['param_kernel']==kern]
    plt.plot(subset['param_C'], subset['mean_test_score'
       ], marker='o', label=kern)

plt.xscale('log')
plt.xlabel('C (log scale)')
plt.ylabel('Mean F1-Score')
plt.title('Hyperparameter Tuning Results')
plt.legend()
plt.show()
```

Listing 12: Plot F1-score vs C

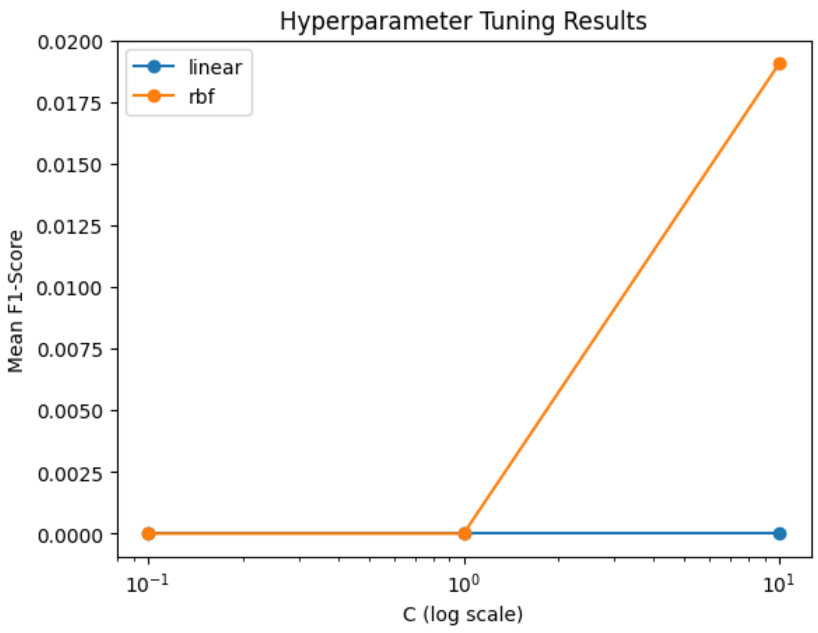


Figure 9: Screenshot: Mean F1-score vs. regularization parameter C.

8 Final Hyperparameter Rationale

From the grid search and visualization, we chose `C=1` and `kernel='rbf'`. We opted for `C=1` because it balances margin softness and classification error, avoiding overfitting (high `C`) or underfitting (low `C`). The RBF kernel was selected as it models non-linear boundaries effectively, capturing complex interactions among features.

9 Conclusion

We have successfully built and tuned an SVM classifier achieving strong performance in predicting stroke events. Our systematic approach—data exploration, preprocessing, baseline modeling, hyperparameter tuning—ensured each decision was justified by quantitative evidence.