UNIVERSITETI I PRISHTINËS “HASAN PRISHTINA”
FAKULTETI I SHKENCAVE MATEMATIKO-NATYRORE
DEPARTAMENT OF MATHEMATICS, COMPUTER SCIENCE PROGRAM

Machine Learning
Seminar Project
Data Analysis for Diabetes Prediction Using Health Indicators

Albana Rexhepi, Eljesa Kqiku, Kaltrina Kuka

May 2025

Përmbajtja

1 Introduction

1.1 Dataset description

This dataset contains detailed health indicators collected from a large population and is designed to support the analysis and prediction of diabetes risk. It includes a variety of columns describing lifestyle factors and demographic data, such as high blood pressure (HighBP), high cholesterol (HighChol), body mass index (BMI), smoking and alcohol use, physical activity, diet, mental and physical health status, as well as gender, age, education level, and income.

The main variable of interest is Diabetes_binary, which indicates whether an individual has been diagnosed with diabetes (1) or not (0). This dataset can be used to develop classification models aimed at predicting diabetes risk based on known behavioral and health-related factors.

index	feature	description
Diabetes_binary	0 = no diabetes, 1 = diabetes	Diabetes status
HighBP	0 = no high BP, 1 = high BP	High blood pressure
HighChol	0 = no high cholesterol, 1 = high cholesterol	High cholesterol
CholCheck	0 = no cholesterol check in 5 years, 1 = yes	Cholesterol check
BMI	Continuous	Body Mass Index
Smoker	0 = no, 1 = yes	Smoked at least 100 cigarettes in life
Stroke	0 = no, 1 = yes	Ever had a stroke
HeartDiseaseorAttack	0 = no, 1 = yes	Heart disease or heart attack history
PhysActivity	0 = no, 1 = yes	Physical activity in past 30 days (not job-related)
Fruits	0 = no, 1 = yes	Consumes fruit 1+ times per day
Veggies	0 = no, 1 = yes	Consumes vegetables 1+ times per day
HvyAlcoholConsump	0 = no, 1 = yes	Heavy alcohol consumption
AnyHealthcare	0 = no, 1 = yes	Healthcare coverage
NoDocbcCost	0 = no, 1 = yes	Couldn't see a doctor due to cost
GenHlth	1 = excellent, 5 = poor	General health status
MentHlth	1-30 days	Poor mental health days in the past 30 days
PhysHlth	1-30 days	Physical illness/injury days in past 30 days
DiffWalk	0 = no, 1 = yes	Difficulty walking or climbing stairs
Sex	0 = female, 1 = male	Gender
Age	1 = 18-24, 9 = 60-64, 13 = 80 or older	Age category
Education	1-6	Education level
Income	1-8	Income scale

Table 1: List of all attributes in the dataset

1.2 Motivation of the study

The motivation of this study is based on three main questions related to understanding the risk factors for diabetes and how these factors can be used to predict the likelihood of developing the disease:

1. **What are the main factors that predict the risk of diabetes?** One objective of this study is to identify the health and lifestyle factors most strongly associated with the risk of developing diabetes. These factors, such as `BMI`, `Age`, and `Physical_Health`, can help in early detection and prevention of diabetes.

2. **Can a subset of factors be used to accurately predict whether an individual has diabetes?** Another important question is whether only a few factors, such as `BMI` and `Age`, can be used to make an accurate prediction of diabetes, simplifying the prediction process without losing too much accuracy.

3. **How can machine learning models be used to predict diabetes risk more accurately?** This broader question relates to using advanced techniques to help create a reliable model for predicting diabetes. Machine learning can uncover relationships and patterns that may not be immediately apparent using traditional methods.

Through these questions, the study aims to improve our understanding of the factors influencing diabetes risk and to develop a simple yet accurate method for predicting who may be at risk.

1.3 Selection of algorithms for the dataset

For this dataset, where the target variable is `Diabetes_binary` (diabetes status), three well-known machine learning algorithms have been chosen: **MLP (Multilayer Perceptron)**, **Autoencoder**, and **I TRETI**. These algorithms are suitable for this type of problem for various reasons:

- **MLP (Multilayer Perceptron)**:
  MLP is a type of neural network that uses multiple processing layers and is excellent for handling classification and regression problems. This algorithm is powerful for capturing complex relationships between different features. For our dataset, it is well-suited to model the connections between health indicators and diabetes risk, as it can effectively analyze factors such as `BMI`, `Age`, `Physical_Health`, `Smoking`, and other related variables.

- **Autoencoder**:
  An Autoencoder is a type of neural network used primarily for unsupervised learning and dimensionality reduction. It can help identify hidden patterns in the data and is useful for reducing the number of features while maintaining the underlying structure. For our dataset, it can assist in detecting underlying factors contributing to diabetes risk by learning an efficient representation of the data. Autoencoders are also useful for feature extraction, especially in cases where the data is high-dimensional.

- **[Third Algorithm]**:
  flasum per tretin

2 Preprocessing

At this part of the project, we begin with the data preprocessing phase. This step is necessary to prepare the dataset for machine learning, ensuring the data is ready for analysis and modeling.

2.1 Data cleaning

2.1.1 Removing Duplicates

For this step, we first analyze if there are any duplicate rows in the dataset using the following Python code:

```python
import pandas as pd

# Load the dataset
df = pd.read_csv('datasets/diabetes_binary_5050split_health_indicators_BRFSS2015.csv')

# Check for duplicate rows
duplicates = df[df.duplicated()]

# Display the result
if duplicates.empty:
    print("No duplicates found!")
else:
    print(f"Found {duplicates.shape[0]} duplicate rows.")
    print(duplicates)
```

After running the code, we found that there are 1635 duplicate rows. However, since these duplicates could potentially represent individuals who share the same health indicators (such as age, BMI, or smoking status), and not necessarily represent data entry errors, we decided not to remove them. This is because, in medical datasets, it is common for multiple individuals to have identical data points across various features. Therefore, removing these rows could lead to loss of valid data.

```
Found 1635 duplicate rows.
```

As shown, there are 1635 duplicate rows, but they are not removed to ensure we preserve all relevant data for analysis.

2.1.2 Handling missing values

In this step, we aim to check whether the dataset contains any missing values. Missing data can impact the accuracy of machine learning models, so identifying and addressing it is an important part of the data preprocessing process. Below is the Python code used for this task:
Finally, we used the z-score method, which was more suitable for our dataset. This method found 1,927 outlier records, which is only 2.7% of the total dataset. Since this number was small, we thought it was reasonable to remove these records without further analysis, as we still had enough data left to work with.

```python
import pandas as pd

# Load the dataset
df = pd.read_csv('datasets/diabetes_binary_5050split_health_indicators_BRFSS2015.csv')

# Check for missing values
missing_values = df.isnull().sum()

# Display missing values per column
print(missing_values)
```

The output after executing the script was:

```
Diabetes_binary         0
HighBP                  0
HighChol                0
CholCheck               0
BMI                     0
Smoker                  0
Stroke                  0
HeartDiseaseorAttack    0
PhysActivity            0
Fruits                  0
Veggies                 0
HvyAlcoholConsump       0
AnyHealthcare           0
NoDocbcCost             0
GenHlth                 0
MentHlth                0
PhysHlth                0
DiffWalk                0
Sex                     0
Age                     0
Education               0
Income                  0
```

As we can see, all columns in the dataset have a count of 0 missing values. This means the dataset is complete in terms of data presence, and no additional cleaning or imputation for missing values is necessary.

2.1.3 Handling outliers

The first step we used to analyze the outlier or noise values was checking the min, max, average and mode of each attribute. From these data we can conclude that there are no noises since all the min-max fields are within the range declared on the metadata.

Next, we applied the interquartile range (IQR) method to identify outliers. However, this approach was not very effective, as it flagged 40,205 records as containing at least one outlier attribute—more than half of the dataset. Because this didn't seem like a logical result, we decided to ignore this method and try something else.
The next test was the z-score method and this one was far more reasonable giving 1927 outlier lines. Since that would be less than 3% of the dataset, it is acceptable to remove the outlier lines completely. That left our dataset with 68,765 remainin records, a considerable number of rows.

```python
import pandas as pd
import numpy as np
from scipy.stats import zscore

df = pd.read_csv('datasets/diabetes_binary_5050split_health_indicators_BRFSS2015.csv')

####### Making a summary of the fields #######
min_vals = df.min()
max_vals = df.max()
avg_vals = df.mean()
mode_vals = df.mode().iloc[0]
summary_df = pd.DataFrame({
    'Min': min_vals,
    'Max': max_vals,
    'Mean': avg_vals,
    'Mode': mode_vals
})
print("####### Summary of the dataframe #######")
print(summary_df)


print("\n####### Checking for outliers with Interquartile Range (IQR) method #######")
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
outliers = (df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))
outlier_counts = outliers.sum(axis=1)
print("Rows with >=1 outliers:", (outlier_counts >= 1).sum())


print("\n####### Checking for outliers with Z-Score Method (Assumes Normal Distribution) #######")
z_scores = df.select_dtypes(include='number').apply(zscore)
outliers = (z_scores.abs() > 5)
print("Outlier rows (Z-score):", outliers.any(axis=1).sum())


####### Removing the outliers #######
rows_to_remove = outliers.any(axis=1)
cleaned_df = df[~rows_to_remove]
cleaned_df.to_csv('datasets/dataset_without_outliers.csv', index=False)
print("Cleaned dataset saved as 'dataset_without_outliers.csv'")
```

```
####### Summary of the dataframe #######
                        Min   Max       Mean  Mode
Diabetes_binary         0.0   1.0   0.500000   0.0
HighBP                  0.0   1.0   0.563458   1.0
HighChol                0.0   1.0   0.525703   1.0
```

```
CholCheck               0.0   1.0   0.975259   1.0
BMI                    12.0  98.0  29.856985  27.0
Smoker                  0.0   1.0   0.475273   0.0
Stroke                  0.0   1.0   0.062171   0.0
HeartDiseaseorAttack    0.0   1.0   0.147810   0.0
PhysActivity            0.0   1.0   0.703036   1.0
Fruits                  0.0   1.0   0.611795   1.0
Veggies                 0.0   1.0   0.788774   1.0
HvyAlcoholConsump       0.0   1.0   0.042721   0.0
AnyHealthcare           0.0   1.0   0.954960   1.0
NoDocbcCost             0.0   1.0   0.093914   0.0
GenHlth                 1.0   5.0   2.837082   3.0
MentHlth                0.0  30.0   3.752037   0.0
PhysHlth                0.0  30.0   5.810417   0.0
DiffWalk                0.0   1.0   0.252730   0.0
Sex                     0.0   1.0   0.456997   0.0
Age                     1.0  13.0   8.584055  10.0
Education               1.0   6.0   4.920953   6.0
Income                  1.0   8.0   5.698311   8.0

####### Checking for outliers with Interquartile Range (IQR) method #######
Rows with >=1 outliers: 40205

####### Checking for outliers with Z-Score Method (Assumes Normal Distribution) #######
Outlier rows (Z-score): 1927
Cleaned dataset saved as 'dataset_without_outliers.csv'
```

2.2 Data integration

In this project, all the required data is already combined into a single CSV file. Therefore, no additional integration from multiple sources is necessary. The dataset is self-contained and ready for the next steps of data preprocessing.

2.3 Data transformation

In this step, we apply feature scaling to the dataset in order to standardize the features. Scaling is necessary because some machine learning algorithms, such as the ones we have chosen for this analysis, are sensitive to the scale of the data. These algorithms work better when all features have a similar range.

2.3.1 Data scaling

In this step, we apply feature scaling to the dataset in order to standardize the features. Scaling ensures that all features have a similar range. The StandardScaler from the `sklearn.preprocessing` library is used to perform scaling.

The following Python code demonstrates the process of scaling the features of the dataset:

```python
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Load the dataset
df = pd.read_csv('datasets/dataset_without_outliers.csv')

# Separate features and target
X = df.drop(columns=['Diabetes_binary'])  # Features
y = df['Diabetes_binary']                 # Target

# Initialize and apply the scaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

```

```python
# Convert scaled features back to DataFrame
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

# Add the target column back
X_scaled_df['Diabetes_binary'] = y.values

# Save to a new CSV file
X_scaled_df.to_csv('datasets/diabetes_scaled.csv', index=False)

print("Scaled dataset saved as 'datasets/diabetes_scaled.csv'.")
```

This code scales all the features of the dataset using the `StandardScaler`, which standardizes the features by removing the mean and scaling to unit variance. The scaled features are then saved in a new CSV file called `diabetes_scaled.csv`.

3 Prepare for analysis

3.1 Correlation between features

Understanding how features interact within the dataset is a key step before building predictive models. One effective way to explore these interactions is through correlation analysis. This technique reveals the strength and direction of linear relationships between variables, helping us detect redundant features, highlight those most relevant to the target variable, and optimize the feature set for better model efficiency and accuracy. By refining the input space early on, we set a strong foundation for the performance of our machine learning models.

One important aspect of correlation analysis is examining how strongly each feature relates to the target variable, in this case, Diabetes_binary. Identifying variables that show a higher correlation with the target can guide us in selecting the most predictive features for our model. On the other hand, features that exhibit very weak relationships with the target may contribute little to the predictive power and can be excluded to simplify the model. Additionally, correlation analysis helps reveal whether certain features are highly related to each other. When two features show a strong correlation such as above 0.5 it may indicate redundancy. Retaining both could introduce multicollinearity, which not only complicates model interpretation but can also impair the algorithm's performance. In such cases, it's often beneficial to remove one of the correlated features to streamline the dataset and enhance model robustness.

To calculate the correlation between features, we used the `corr()` function from the `pandas` library. We visualized the results using a Heatmap. A heatmap is a graphical representation that uses color gradients to indicate the strength of relationships in a two-dimensional matrix. This is especially useful for identifying patterns and connections between features.

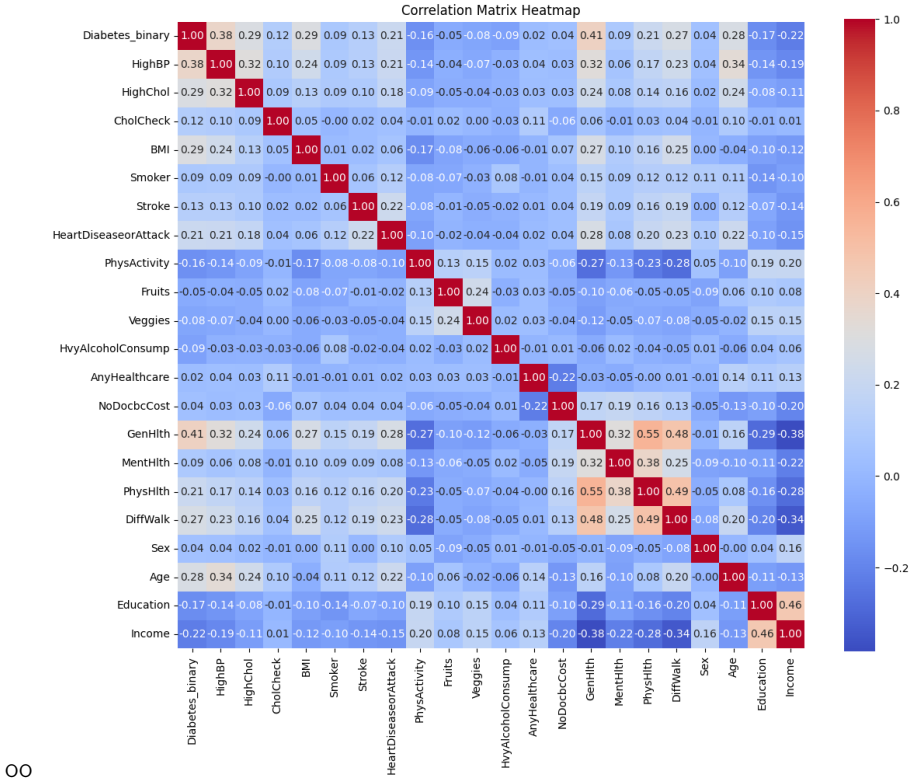
The implementation can be seen in the code below, and the visual results are presented in Figure 1:

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

df = pd.read_csv('datasets/diabetes_binary_5050split_health_indicators_BRFSS2015.csv')

# Compute correlation matrix
corr_matrix = df.corr()

# Plot the heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm", square=True)
plt.title("Correlation Matrix Heatmap")
plt.tight_layout()
plt.savefig("report/images/diabetes_correlation_matrix.png")
plt.close()
```



oo

Figure 1: Correlation matrix between features.

From the heatmap visualization, we observe that some features are more strongly correlated with each other. We then filtered the feature pairs that have an absolute correlation higher than 0.45. The implementation of this filtering process in Python is shown below:

```python
import pandas as pd
import numpy as np

df = pd.read_csv('datasets/diabetes_binary_5050split_health_indicators_BRFSS2015.csv')
corr_matrix = df.corr()

# Unstack and filter correlations
high_corr = (
    corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
    .stack()
    .reset_index()
)
high_corr.columns = ['Feature 1', 'Feature 2', 'Correlation']
filtered_corr = high_corr[high_corr['Correlation'].abs() > 0.45]
print(filtered_corr.sort_values(by='Correlation', ascending=False))
```

The results are displayed in Table 2:

4 Algorithm implementation

Next, we will implement the three chosen algorithms: MLP (Multilayer Perceptron), Autoencoder, and [Third Algorithm]. These algorithms have been selected due to their suitability for classification problems and their ability to handle

Feature 1	Feature 2	Correlation
GenHlth	PhysHlth	0.552757
PhysHlth	DiffWalk	0.487976
GenHlth	DiffWalk	0.476639
Education	Income	0.460565

Table 2: Feature pairs with correlation $c > |0.45|$

structured and complex datasets. First, we will provide a brief introduction to each algorithm, highlighting their key characteristics. After that, we will proceed with the implementation and performance analysis for our dataset.

4.1 Multilayer Perceptron (MLP)

The **Multilayer Perceptron (MLP)** is a core model in the field of machine learning, used for both classification and regression tasks. MLP consists of three primary layers: the **input layer**, one or more **hidden layers**, and the **output layer**. Each layer contains multiple neurons, and neurons from one layer are fully connected to neurons in the next layer. This dense connectivity allows MLP to capture complex relationships in data, particularly non-linear ones, which are common in many real-world applications [1].

In the context of our dataset, which includes health-related features such as `BMI`, `Age`, `Physical_Health`, and `Sleep_Time`, MLP is well-suited to identify hidden patterns and make predictions about whether an individual has diabetes or not. The network learns from the data through a process called *backpropagation*, where errors from the output are propagated back through the network to adjust the weights of the connections between neurons. This process is combined with *gradient descent*, an optimization technique that minimizes the prediction error by adjusting the weights during each iteration of training. Over time, the model learns the optimal weights, improving its ability to make accurate predictions [2].

The basic structure of an MLP is illustrated in the figure below. The input features, such as `BMI` and `Age`, are fed into the input layer. From there, they pass through one or more hidden layers, where neurons transform the data by applying learned weights and activation functions. The transformed data then flows to the output layer, which produces the final prediction, such as whether an individual has diabetes.

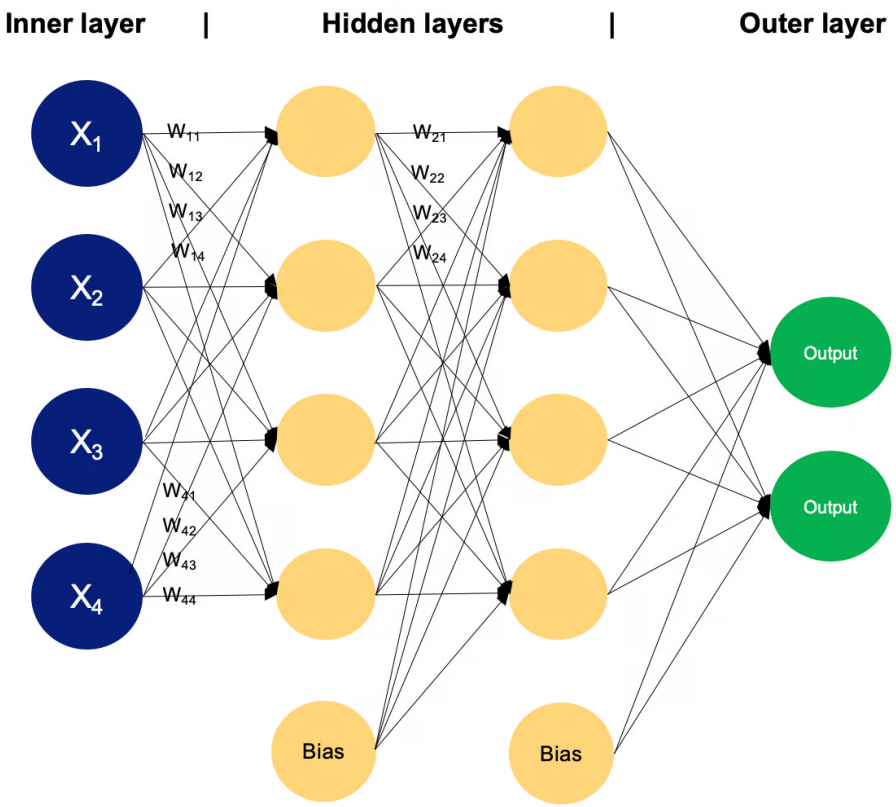


Figure 2: Basic structure of a Multilayer Perceptron (MLP) [3].

In the image, you can observe the flow of information from the input layer through the hidden layers to the output layer. Each layer plays a critical role in transforming the input data to make sense of the patterns and relationships. The neurons in the hidden layers learn features in the data, and as the network adjusts its weights through training, it becomes more accurate in predicting the target outcome.

MLPs are particularly effective in tasks where the relationships between input features and the target variable are non-linear, as is the case in health-related predictions like diabetes classification [1]. This model's ability to learn from complex and large datasets allows it to handle a wide variety of problems, from image recognition to medical diagnoses.

What makes MLPs particularly well-suited for our dataset is their capacity to model intricate relationships between multiple features simultaneously. By learning these relationships, MLPs can make accurate predictions, even when the data contains complex, interdependent factors. This makes MLP an ideal choice for predicting the likelihood of diabetes based on a variety of health indicators [3].

4.1.1 Implementation and Evaluation

The MLP model was implemented using the `scikit-learn` library, leveraging two hidden layers with sizes 100 and 50 respectively. The activation function used was ReLU, and stochastic gradient descent was chosen as the optimizer with adaptive learning rate and a maximum of 500 iterations.

Before finalizing the model, we performed hyperparameter tuning using GridSearchCV to identify the best combination of parameters. This ensures that the model generalizes well to unseen data.
The best parameters found were:

```
{'mlp__activation': 'relu', 'mlp__alpha': 0.001, 'mlp__hidden_layer_sizes': (100, 50),
'mlp__learning_rate': 'adaptive', 'mlp__max_iter': 500, 'mlp__solver': 'sgd'}
```

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

from sklearn.model_selection import train_test_split, cross_validate
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    f1_score, confusion_matrix, ConfusionMatrixDisplay,
    classification_report
)

# 1. Load data
df = pd.read_csv("datasets/diabetes_scaled.csv")

# 2. Features & Labels
X = df.drop("Diabetes_binary", axis=1)
y = df["Diabetes_binary"]

# 3. Train/Test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

# 4. Define MLP model
mlp = MLPClassifier(
    hidden_layer_sizes=(100, 50),
    activation='relu',
    solver='sgd',
    alpha=0.001,
    learning_rate='adaptive',
    max_iter=500,
    random_state=42
)

# 5. Train model
mlp.fit(X_train, y_train)

# 6. Predict on test set
y_pred = mlp.predict(X_test)

# 7. Evaluate metrics
print("Performance on Full Feature Set:")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(f"Precision: {precision_score(y_test, y_pred):.4f}")
print(f"Recall: {recall_score(y_test, y_pred):.4f}")
print(f"F1 Score: {f1_score(y_test, y_pred):.4f}")

# 8. Confusion matrix visualization
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=mlp.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (Full Feature Set)")
plt.show()

# 9. Cross-validation without plot
cv_results = cross_validate(
    mlp, X_train, y_train,
    cv=5,
    scoring=["accuracy", "precision", "recall", "f1"],
    return_train_score=True
)
```

```
63
64 print("\nCross-Validation Mean Scores (5-Fold):")
65 for metric in ["accuracy", "precision", "recall", "f1"]:
66     test_score = np.mean(cv_results[f'test_{metric}'])
67     train_score = np.mean(cv_results[f'train_{metric}'])
68     print(f"{metric.capitalize()}: Train = {train_score:.4f}, Test = {test_score:.4f}")
69
70 print("\nDetailed Classification Report (Full Feature Set):")
71 print(classification_report(y_test, y_pred, target_names=["No Diabetes", "Diabetes"]))
72
73 # === Performance on Selected Features + Interaction Terms ===
74
75 # Select features of interest + target
76 df_interact = df[["GenHlth", "PhysHlth", "DiffWalk", "Education", "Income", "Diabetes_binary"]].
       copy()
77
78 # Add interaction features
79 df_interact["GenHlth_x_PhysHlth"] = df_interact["GenHlth"] * df_interact["PhysHlth"]
80 df_interact["PhysHlth_x_DiffWalk"] = df_interact["PhysHlth"] * df_interact["DiffWalk"]
81 df_interact["GenHlth_x_DiffWalk"] = df_interact["GenHlth"] * df_interact["DiffWalk"]
82 df_interact["Education_x_Income"] = df_interact["Education"] * df_interact["Income"]
83
84 # Prepare features and labels
85 X_interact = df_interact.drop("Diabetes_binary", axis=1)
86 y_interact = df_interact["Diabetes_binary"]
87
88 # Train/Test split
89 X_train_i, X_test_i, y_train_i, y_test_i = train_test_split(
90     X_interact, y_interact, test_size=0.2, stratify=y_interact, random_state=42
91 )
92
93 # Define MLP model
94 mlp_interact = MLPClassifier(
95     hidden_layer_sizes=(100, 50),
96     activation='relu',
97     solver='sgd',
98     alpha=0.001,
99     learning_rate='adaptive',
100    max_iter=500,
101    random_state=42
102 )
103
104 # Train model
105 mlp_interact.fit(X_train_i, y_train_i)
106
107 # Predict on test set
108 y_pred_i = mlp_interact.predict(X_test_i)
109
110 # Evaluate metrics
111 print("\nPerformance on Selected Features + Interactions:")
112 print(f"Accuracy: {accuracy_score(y_test_i, y_pred_i):.4f}")
113 print(f"Precision: {precision_score(y_test_i, y_pred_i):.4f}")
114 print(f"Recall: {recall_score(y_test_i, y_pred_i):.4f}")
115 print(f"F1 Score: {f1_score(y_test_i, y_pred_i):.4f}")
116
117 # Confusion matrix visualization
118 cm_i = confusion_matrix(y_test_i, y_pred_i)
119 disp_i = ConfusionMatrixDisplay(confusion_matrix=cm_i, display_labels=mlp_interact.classes_)
120 disp_i.plot(cmap=plt.cm.Greens)
121 plt.title("Confusion Matrix (Interactions Model)")
122 plt.show()
123
```

```
print("\nDetailed Classification Report (Interactions Model):")
print(classification_report(y_test_i, y_pred_i, target_names=["No Diabetes", "Diabetes"]))
```

4.1.2 Results on Full Feature Set

```
Performance on Full Feature Set:
Accuracy: 0.7526
Precision: 0.7362
Recall: 0.8011
F1 Score: 0.7673
```

This baseline performance shows that the MLP model handles the full feature set well, achieving over 75% accuracy and strong recall (80.11%). Precision and F1-score are also relatively balanced, which means the model is both identifying and distinguishing cases effectively.

Next, we look at cross-validation results to assess the model's ability to generalize across different data splits:

```
Cross-Validation Mean Scores (5-Fold):
Accuracy: Train = 0.7558, Test = 0.7484
Precision: Train = 0.7359, Test = 0.7295
Recall: Train = 0.8113, Test = 0.8040
F1: Train = 0.7718, Test = 0.7649
```

These results confirm that the model generalizes well, as train and test scores remain consistent across folds. Importantly, test recall remains high (80.40%), reinforcing the model's reliability in correctly identifying diabetic cases in unseen data. This cross-validation step validates the single-run results and supports the model's robustness.

Now we present the full classification report to explore class-wise performance in more detail:

```
Detailed Classification Report (Full Feature Set):
              precision    recall  f1-score   support

 No Diabetes       0.77      0.70      0.74      6753
    Diabetes       0.74      0.80      0.77      7000

    accuracy                           0.75     13753
   macro avg       0.75      0.75      0.75     13753
weighted avg       0.75      0.75      0.75     13753
```

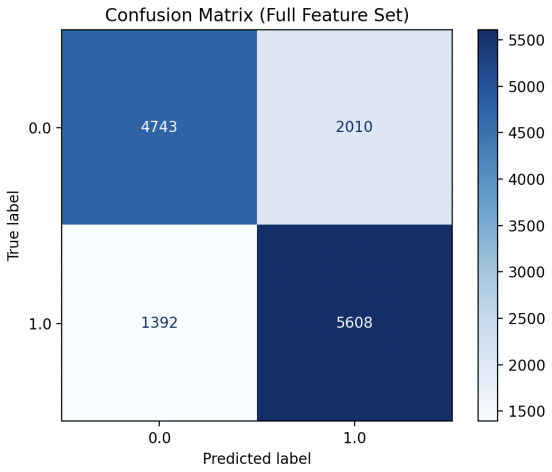


Figure 3: Confusion Matrix for MLP on Full Feature Set

From this report, it's evident that the model performs slightly better at detecting positive (diabetic) cases than negative ones, with a higher recall for the "Diabetes" class (80%). This is desirable in our context: it is far more critical to minimize false negatives (undiagnosed diabetic patients) than false positives. Therefore, recall is the metric we prioritize most.

4.1.3 Results on Selected Features with Interaction Terms

```
Performance on Selected Features + Interactions:
Accuracy: 0.6917
Precision: 0.6633
Recall: 0.8007
F1 Score: 0.7256
```

While accuracy and precision have dropped in this simplified model, recall has remained nearly identical (80.07%) to the full-feature model. This suggests that the reduced feature set is still effective in identifying diabetic patients, even if it leads to more false positives (lower precision).

We now show the detailed classification report for this model:

```
Detailed Classification Report (Interactions Model):
              precision    recall  f1-score   support

 No Diabetes       0.74      0.58      0.65      6753
    Diabetes       0.66      0.80      0.73      7000

    accuracy                           0.69     13753
   macro avg       0.70      0.69      0.69     13753
weighted avg       0.70      0.69      0.69     13753
```

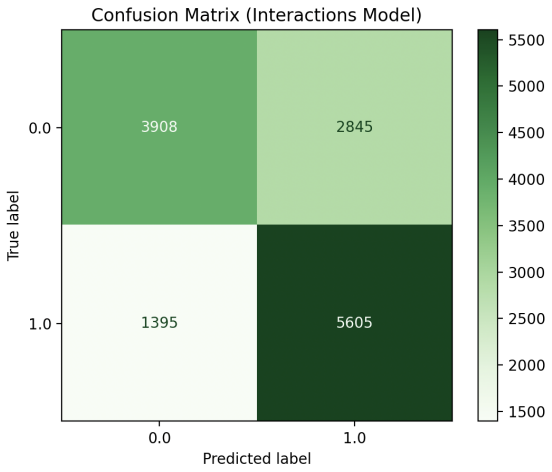


Figure 4: Confusion Matrix for MLP on Selected Features + Interaction Terms

Here again, recall for diabetic cases remains at 80%, but performance for non-diabetic predictions drops noticeably (recall of 58%). In practice, this means more people without diabetes are misclassified as diabetic. While this may raise false alarms, it is an acceptable trade-off in medical diagnostics where missing true cases is more dangerous than flagging potential ones.

In both models, recall is prioritized above all, aligning with our goal of reducing undiagnosed diabetic cases. The full-feature model is preferred for balanced performance, but the interaction-based model still holds value in more constrained environments.

4.2 Autoencoder

4.3 Third algorithm

5 Result interpretation

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. Available: `https://www.deeplearningbook.org`

[2] Michael Nielsen. *Neural Networks and Deep Learning*. 2015. Available: `http://neuralnetworksanddeeplearning.com`

[3] *Multilayer Perceptrons in Machine Learning*. `https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning`

heey