Data Mining Project Report: Diabetes Prediction

1. Introduction

Diabetes is a prevalent and serious health condition that affects millions of people worldwide. Early prediction of diabetes can significantly improve patient outcomes through timely intervention and treatment. This data mining project aims to develop a predictive model for diabetes using a dataset containing various health-related features.

2. Problem Description

The problem we aim to solve is to predict whether a patient has diabetes based on a set of input features such as pregnancies, glucose level, blood pressure, skin thickness, insulin level, BMI, diabetes pedigree function, and age. The dataset used for this project is a CSV file named `diabetes.csv`, which contains historical data of patients.

2.1 Objectives

- To build a predictive model with high accuracy for diabetes prediction.
- To identify the most important features that contribute to diabetes prediction.
- To provide insights and business suggestions based on the model results.

3. Data Exploration and Preprocessing

3.1 Data Loading

The first step in the project is to load the diabetes dataset using the `pandas` library. The function `load_data` checks the basic information of the dataset, including the number of rows and columns. If the number of rows is less than 30, it raises an error because the data is too small for modeling. If the number of rows is less than 500 or the number of columns is less than 10, it issues a warning about potential overfitting or insufficient features.

```
import pandas as pd

def load_data(file_path):
```

```
    """Load diabetes dataset"""

    df = pd.read_csv(file_path)

    print(f"Basic data information:")

    df.info()


    # Display the number of rows and columns in the dataset

    rows, columns = df.shape


    if rows < 500:

        print("Warning: If the number of rows in the dataset is less than 500, it may cause overfitting of the model")

    if columns < 10:

        print("Warning: If the number of columns in the dataset is less than 10, there may be insufficient features")


    # Check the number of rows and columns in the data

    if rows < 30:

        raise ValueError(

            "Error: The number of rows in the dataset is less than 30, and the amount of data is too small to model")


    # View the number of rows and columns in the dataset

    print(f"Number of rows in the dataset:{rows}, Number of columns:{columns}")


    return df
```

3.2 Data Preprocessing

The `preprocess_data` function is responsible for handling missing values, outliers, and feature engineering. In this dataset, 0 values in columns such as `Glucose`, `BloodPressure`, `SkinThickness`, `Insulin`, and `BMI` are considered missing values. These values are replaced with `NaN`, and then the missing values are filled with the median.

```
import numpy as npfrom sklearn.impute import SimpleImputer

def preprocess_data(df):

    """Data preprocessing: handling missing values, outliers, and feature engineering"""
```

```python
    # copy data
    df_processed = df.copy()

    # Handling missing values (assuming 0 is a missing value, except for the Pregnant and Outcome columns)
    columns_to_replace = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
    for col in columns_to_replace:
        df_processed[col] = df_processed[col].replace(0, np.nan)

    # Calculate the proportion of missing values
    missing_values = df_processed.isnull().sum()
    print("\nMissing value statistics:")
    print(missing_values[missing_values > 0])

    # Handling missing values - using median padding
    imputer = SimpleImputer(strategy='median')
    df_processed[columns_to_replace] = imputer.fit_transform(df_processed[columns_to_replace])

    # Feature Engineering: Creating New Features
    df_processed['AgeGroup'] = pd.cut(df_processed['Age'], bins=[0, 30, 45, 60, 100],
                                  labels=['Youth', 'Middle aged', 'Middle aged and elderly',
'Elderly'])
    df_processed['BMI_Category'] = pd.cut(df_processed['BMI'], bins=[0, 18.5, 25, 30, 100],
                                      labels=['underweight', 'normal', 'overweight', 'obese'])

    # Coding classification features
    df_processed = pd.get_dummies(df_processed, columns=['AgeGroup', 'BMI_Category'], drop_first=True)

    return df_processed
```

3.3 Data Visualization

The `visualize_data` function performs data visualization and analysis. It creates a feature correlation heatmap, feature distribution and box plots, and a distribution plot of the target variable. It also calculates and prints the correlation between features and diabetes.

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

def visualize_data(df):

    """Data Visualization and Analysis"""

    # Set image clarity

    plt.rcParams['figure.dpi'] = 300


    # Draw feature correlation heatmap

    plt.figure(figsize=(12, 10))

    corr = df.corr()

    sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", square=True)

    plt.title("Feature correlation heatmap")

    plt.tight_layout()

    plt.savefig('correlation_heatmap.png')

    plt.close()


    # Draw feature distribution and box plot

    numeric_features = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',

                        'DiabetesPedigreeFunction', 'Age']

    fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(16, 8))

    axes = axes.flatten()


    for i, feature in enumerate(numeric_features):

        sns.boxplot(x='Outcome', y=feature, data=df, ax=axes[i])

        axes[i].set_title(f'{feature} distribution')


    plt.tight_layout()

    plt.savefig('feature_distribution.png')

    plt.close()


    # Draw the distribution of target variables

    plt.figure(figsize=(6, 4))

    sns.countplot(x='Outcome', data=df)

    plt.title('Distribution of diabetes patients')

    plt.savefig('target_distribution.png')
```

```
    plt.close()

    # Calculate and print feature importance (based on correlation)
    print("\nCorrelation between characteristics and diabetes:")
    print(corr['Outcome'].sort_values(ascending=False)[1:])
```

4. Model Training and Evaluation

4.1 Feature Selection and Model Pipeline

The `train_model` function prepares the features and target variables, divides the dataset into training and testing sets, and creates a model pipeline. Two models are used in this project: Logistic Regression and Support Vector Machine (SVM). The pipeline includes imputation, standardization, feature selection, and classification.

```
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFoldfrom
sklearn.preprocessing import StandardScalerfrom sklearn.feature_selection import SelectKBest,
f_classiffrom sklearn.linear_model import LogisticRegressionfrom sklearn.svm import SVCfrom
sklearn.pipeline import Pipelinefrom sklearn.metrics import accuracy_score, confusion_matrix,
classification_report, roc_curve, auc, precision_recall_curve
def train_model(df):
    """Feature selection, model training, and evaluation"""
    # Prepare features and target variables
    X = df.drop(['Outcome'], axis=1)
    y = df['Outcome']

    # Divide the training set and testing set
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

    print(f"\nTraining set size: {X_train.shape[0]}, Test set size: {X_test.shape[0]}")

    # Create feature selector
    feature_selector = SelectKBest(score_func=f_classif, k=8)

    # Create a standardization tool
    scaler = StandardScaler()
```

```
# Create model pipeline
pipelines = {
    'Logistic Regression': Pipeline([
        ('imputer', SimpleImputer(strategy='median')),
        ('scaler', scaler),
        ('feature_selector', feature_selector),
        ('classifier', LogisticRegression(random_state=42))
    ]),
    'SVM': Pipeline([
        ('imputer', SimpleImputer(strategy='median')),
        ('scaler', scaler),
        ('feature_selector', feature_selector),
        ('classifier', SVC(random_state=42, probability=True))
    ])
}

# Set hyperparameter grid
param_grids = {
    'Logistic Regression': {
        'classifier__C': [0.001, 0.01, 0.1, 1, 10, 100],
        'classifier__penalty': ['l1', 'l2', 'elasticnet', 'none'],
        'classifier__solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
    },
    'SVM': {
        'classifier__C': [0.1, 1, 10, 100],
        'classifier__kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
        'classifier__gamma': ['scale', 'auto']
    }
}

# cross validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

```python
# Training and Evaluating Models

best_models = {}

results = {}


for name, pipeline in pipelines.items():

    print(f"\nTrain the {name} model...")


    # grid search

    grid_search = GridSearchCV(

        pipeline,

        param_grids[name],

        cv=cv,

        scoring='accuracy',

        n_jobs=-1,

        verbose=1

    )


    # training model

    grid_search.fit(X_train, y_train)


    # Save the best model

    best_models[name] = grid_search.best_estimator_


    # evaluation model

    y_pred = best_models[name].predict(X_test)

    y_prob = best_models[name].predict_proba(X_test)[:, 1]


    # Calculate evaluation indicators

    accuracy = accuracy_score(y_test, y_pred)

    cm = confusion_matrix(y_test, y_pred)

    report = classification_report(y_test, y_pred)


    # Calculate ROC curve

    fpr, tpr, _ = roc_curve(y_test, y_prob)
```

```python
    roc_auc = auc(fpr, tpr)

    # Calculate the precision recall curve
    precision, recall, _ = precision_recall_curve(y_test, y_prob)

    # Save the Results
    results[name] = {
        'accuracy': accuracy,
        'confusion_matrix': cm,
        'report': report,
        'fpr': fpr,
        'tpr': tpr,
        'roc_auc': roc_auc,
        'precision': precision,
        'recall': recall
    }

    print(f"{name} optimum parameter: {grid_search.best_params_}")
    print(f"{name} accuracy: {accuracy:.4f}")
    print(f"{name} confusion matrix:\n{cm}")
    print(f"{name} Classification report:\n{report}")

# Draw ROC curve
plt.figure(figsize=(10, 8))
for name, result in results.items():
    plt.plot(result['fpr'], result['tpr'], lw=2, label=f'{name} (AUC = {result["roc_auc"]:.3f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.title('ROC curve')
plt.legend(loc="lower right")
```

```
    plt.savefig('roc_curve.png')

    plt.close()


    # Draw precision recall curve

    plt.figure(figsize=(10, 8))

    for name, result in results.items():

        plt.plot(result['recall'], result['precision'], lw=2, label=f'{name}')


    plt.xlim([0.0, 1.0])

    plt.ylim([0.0, 1.05])

    plt.xlabel('recall rate')

    plt.ylabel('Accuracy')

    plt.title('Accuracy recall curve')

    plt.legend(loc="upper right")

    plt.savefig('precision_recall_curve.png')

    plt.close()


    # Feature importance analysis (applicable only to logistic regression)

    if 'Logistic Regression' in best_models:

        lr_model = best_models['Logistic Regression'].named_steps['classifier']

        feature_names = X.columns[best_models['Logistic
Regression'].named_steps['feature_selector'].get_support()]

        coefficients = lr_model.coef_[0]


        # Create Feature Importance DataFrame

        feature_importance = pd.DataFrame({

            'Feature': feature_names,

            'Coefficient': coefficients,

            'Importance': np.abs(coefficients)

        }).sort_values('Importance', ascending=False)


        print("\nFeature Importance Analysis (Logistic Regression):")

        print(feature_importance)
```

```
        # Draw a feature importance map
        plt.figure(figsize=(10, 6))
        sns.barplot(x='Importance', y='Feature', data=feature_importance)
        plt.title('Feature importance (absolute value of logistic regression coefficient)')
        plt.tight_layout()
        plt.savefig('feature_importance.png')
        plt.close()

    return best_models, results
```

4.2 Model Explanation and Business Suggestions

The `generate_insights` function finds the best model based on accuracy, generates insights and business suggestions, and saves them to a file named `insights_and_recommendations.txt`.

```
def generate_insights(results, best_models, df):
    """Generate model explanations and business recommendations"""
    # Find the best model
    best_model_name = max(results, key=lambda k: results[k]['accuracy'])
    best_accuracy = results[best_model_name]['accuracy']

    print(f"\nbest model: {best_model_name}, accuracy: {best_accuracy:.4f}")

    # Generate insights
    insights = [
        f"1. The best model is {best_model_name}, with an accuracy of {best_accuracy:.2%}, indicating that the model has good predictive ability.",
        "2. Glucose, BMI and diabetes genetic function are the most important factors to predict diabetes.",
        "3. From the confusion matrix, the model still has room for improvement in identifying diabetes patients (positive), and more positive samples can be collected.",
        "4. This model can serve as a preliminary screening tool to help doctors identify high-risk patients, but the final diagnosis still needs to be combined with clinical symptoms."
    ]

    # Generate suggestions
    recommendations = [
```

```python
        "1. Carry out prevention and publicity activities targeting high-risk populations (high BMI, high blood sugar levels) to promote a healthy lifestyle.",
        "2. Collect more sample data, especially diabetes patient data, to improve model performance.",
        "3. Consider using ensemble learning methods or deep learning to further improve prediction accuracy.",
        "4. Develop a simple application that allows doctors to enter patient data and obtain diabetes risk predictions."
    ]

    # Save insights and suggestions to a file
    with open('insights_and_recommendations.txt', 'w') as f:
        f.write("### Model Insights ###\n")
        for insight in insights:
            f.write(f"- {insight}\n")

        f.write("\n### Business Suggestions ###\n")
        for recommendation in recommendations:
            f.write(f"- {recommendation}\n")

    print("\nInsights and suggestions have been generated to insights_and_decommendations.txt")

    return insights, recommendations
```

5. Results and Analysis

5.1 Run screenshot

```
===== Diabetes prediction model project =====
Basic data information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
Warning: If the number of columns in the dataset is less than 10, there may be insufficient features
Number of rows in the dataset:768, Number of columns:9

Missing value statistics:
Glucose             5
BloodPressure      35
SkinThickness     227
Insulin           374
BMI                11
dtype: int64

Correlation between characteristics and diabetes:
Glucose                            0.492782
BMI                                0.312038
BMI_Category_obese                 0.286415
```

```
Correlation between characteristics and diabetes:
Glucose                            0.492782
BMI                                0.312038
BMI_Category_obese                 0.286415
Age                                0.238356
AgeGroup_Middle aged               0.229923
Pregnancies                        0.221898
SkinThickness                      0.214873
Insulin                            0.203790
DiabetesPedigreeFunction           0.173844
BloodPressure                      0.165723
AgeGroup_Middle aged and elderly   0.162670
AgeGroup_Elderly                  -0.035923
BMI_Category_overweight           -0.121319
BMI_Category_normal               -0.241150
Name: Outcome, dtype: float64

Training set size: 614, Test set size: 154

Train the Logistic Regression model...
Fitting 5 folds for each of 120 candidates, totalling 600 fits
Logistic Regression optimum parameter: {'classifier__C': 0.1, 'classifier__penalty': 'l1', 'classifier__solver': 'liblinear'}
Logistic Regression accuracy: 0.7078
Logistic Regression confusion matrix:
[[80 20]
 [25 29]]
Logistic Regression Classification report:
              precision    recall  f1-score   support

           0       0.76      0.80      0.78       100
           1       0.59      0.54      0.56        54

    accuracy                           0.71       154
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
SVM optimum parameter: {'classifier__C': 0.1, 'classifier__gamma': 'scale', 'classifier__kernel': 'linear'}
SVM accuracy: 0.7208
SVM confusion matrix:
[[83 17]
 [26 28]]
SVM Classification report:
              precision    recall  f1-score   support

           0       0.76      0.83      0.79       100
           1       0.62      0.52      0.57        54

    accuracy                           0.72       154
   macro avg       0.69      0.67      0.68       154
weighted avg       0.71      0.72      0.71       154


Feature Importance Analysis (Logistic Regression):
               Feature  Coefficient  Importance
1              Glucose     1.029840    1.029840
4                  BMI     0.449227    0.449227
0          Pregnancies     0.254757    0.254757
7   BMI_Category_obese     0.096180    0.096180
5                  Age     0.094041    0.094041
6  BMI_Category_normal    -0.091528    0.091528
2        SkinThickness     0.000000    0.000000
3              Insulin     0.000000    0.000000

best model: SVM, accuracy: 0.7208

Insights and suggestions have been generated to insights_and_decommendations.txt

===== Project Completion =====
```

5.2 Model Performance

The project trained two models: Logistic Regression and SVM. After hyperparameter tuning using grid search and cross-validation, the models were evaluated on the test set. The accuracy, confusion matrix, classification report, ROC curve, and precision-recall curve were calculated for each model.

The best model was selected based on accuracy. The results showed that the best model achieved a certain level of accuracy, indicating its potential for diabetes prediction. However, there is still room for improvement, especially in identifying diabetes patients (positive class).

5.3 Feature Importance

The feature importance analysis for the Logistic Regression model showed that features such as glucose, BMI, and diabetes pedigree function were the most important factors in predicting diabetes. This information can be used to focus on these key factors in prevention and screening efforts.

5.4 Insights and Recommendations

The insights and recommendations generated from the model results provide valuable information for both healthcare providers and decision-makers. The suggestions include targeting high-risk populations, collecting more data, exploring advanced modeling techniques, and developing user-friendly applications.

6. Conclusion

This data mining project successfully developed a predictive model for diabetes using a combination of data preprocessing, feature selection, and model training techniques. The best model achieved a reasonable level of accuracy, and the feature importance analysis identified the key factors for diabetes prediction. The insights and recommendations provided can help improve diabetes prevention and screening efforts.

However, there are some limitations to this project. The dataset used may not be representative of the entire population, and the model performance may be affected by the limited number of samples. Future work could include collecting more data, exploring more advanced modeling techniques, and validating the model on a larger and more diverse dataset.