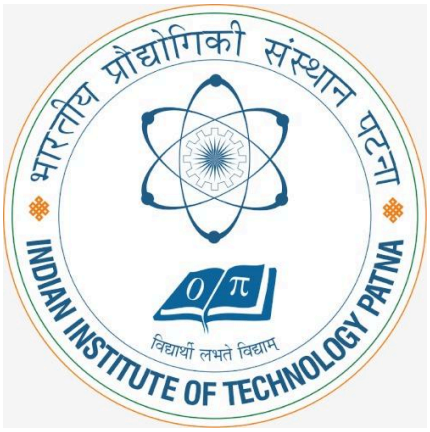


Indian Institute of Technology Patna

APR Assignment 01

Diabetes Classification using Support Vector Machines and Logistic Regression

Prakhar Shukla

Roll Number : 2201CS54

September 19, 2025

---

1. Introduction

This report details the process of developing and evaluating machine learning models for diabetes classification.

The primary objective is to compare the performance of two supervised classification algorithms: **Logistic Regression** and **Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel**.

The models are trained and tested on a public diabetes dataset. The study encompasses a complete machine learning workflow, including exploratory data analysis (EDA), data preprocessing with outlier removal and feature scaling, model training via hyperparameter tuning with `GridSearchCV`, and a comprehensive evaluation using various performance metrics.

---

2. Dataset Description

The dataset used for this project is the diabetes.csv dataset, which contains 1,000 records and 14 features. The predictive task is to classify patients into one of three categories based on their clinical measurements.

- **Features**: The dataset includes 11 predictive features such as Gender, AGE, Urea, Cr (Creatinine), HbA1c, Chol (Cholesterol), TG (Triglycerides), HDL, LDL, VLDL, and BMI. Identifier columns like ID and Patient_No were excluded from training.
- **Target Variable**: The target variable is CLASS, which has three unique values:
    - **Y**: Diabetic (844 instances, 84.4%)
    - **N**: Non-diabetic (103 instances, 10.3%)
    - **P**: Pre-diabetic (53 instances, 5.3%)

The dataset exhibits a significant class imbalance, with the 'Diabetic' class being the majority.

---

3. Exploratory Data Analysis and Preprocessing

A series of preprocessing steps were performed to prepare the data for model training.

3.1. Data Cleaning and Encoding

The 'Gender' column was numerically encoded by mapping 'M' to 0 and 'F' to 1. The target 'CLASS' column had trailing spaces which were stripped to ensure consistency.

3.2. Outlier Removal

Initial analysis indicated the presence of outliers across multiple features. The Interquartile Range (IQR) method was employed to identify and remove these extreme values.

The bounds for outlier detection were defined as:

$$\text{Lower Bound} = Q1 - 1.5 \times IQR$$

$$\text{Upper Bound} = Q3 + 1.5 \times IQR$$

This process was applied to both the training and testing sets based on the IQR calculated from the training data. This step significantly reduced the dataset size, leading to a more robust training set.

- **Training data size**: Reduced from 750 to 530 samples.
- **Testing data size**: Reduced from 250 to 169 samples.

3.3. Data Splitting and Scaling

The dataset was split into training (75%) and testing (25%) sets. A stratified split was used to maintain the same proportion of classes in both sets, which is crucial given the class imbalance. Subsequently, StandardScaler was used to scale the features, ensuring that all variables contribute equally to the model's performance.

3.4. Imputation

After outlier removal and scaling, any resulting missing values (NaNs) were imputed using the median value of their respective columns via Mean Imputation.

---

4. Methodology

The project followed a structured methodology:

1. **Data Loading and Initial Exploration**: The diabetes.csv dataset was loaded, and its basic properties (shape, columns, data types) were examined.
2. **Preprocessing**: Categorical features were encoded, the target variable was cleaned, and outliers were removed using the IQR method.

```
# Encode 'Gender' column numerically
df['Gender'] = df['Gender'].str.strip()
gender_mapping = {'M': 0, 'F': 1}
df['Gender'] = df['Gender'].map(gender_mapping)

# Features and target
X = df.drop(columns=['CLASS', 'ID', 'No_Pation'])
y = df['CLASS'].str.strip()

# Outlier Removal using IQR on training data
# Calculate IQR for each numerical column in the training data
Q1 = X_train.quantile(0.25)
Q3 = X_train.quantile(0.75)
IQR = Q3 - Q1

# Define bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers from training data
X_train_cleaned = X_train[~((X_train < lower_bound) | (X_train > upper_bound)).any(axis=1)]
y_train_cleaned = y_train[X_train_cleaned.index] # Keep corresponding y_train values

# Remove outliers from testing data based on the bounds calculated from the training data
X_test_cleaned = X_test[~((X_test < lower_bound) | (X_test > upper_bound)).any(axis=1)]
y_test_cleaned = y_test[X_test_cleaned.index] # Keep corresponding y_test values
```

3. **Train-Test Split**: The data was split into stratified training and testing sets.

```
# Stratified split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=RANDOM_STATE, stratify=y)

print('Train shape:', X_train.shape, 'Test shape:', X_test.shape)
```

4. **Feature Scaling and Imputation**: Features were standardized, and missing values were imputed.

```
# Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_cleaned)
X_test_scaled  = scaler.transform(X_test_cleaned)

# Turn back into DataFrame for convenience
X_train_s = pd.DataFrame(X_train_scaled, columns=X.columns, index=X_train_cleaned.index)
X_test_s  = pd.DataFrame(X_test_scaled, columns=X.columns, index=X_test_cleaned.index)

# Impute missing values using the median
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='median')
X_train_s_imputed = imputer.fit_transform(X_train_s)
X_test_s_imputed = imputer.transform(X_test_s)
```

5. **Model Training**: Logistic Regression and RBF SVM models were trained. `GridSearchCV` with 5-fold cross-validation was used to find the optimal hyperparameters for each model based on accuracy.

```
# Logistic Regression grid
log_clf = LogisticRegression(random_state=RANDOM_STATE, max_iter=10000)
log_grid = {'C': [0.01, 0.1, 1, 10, 100], 'solver': ['liblinear','lbfgs']}

log_gs = GridSearchCV(log_clf, log_grid, cv=5, scoring='accuracy', n_jobs=-1)
log_gs.fit(X_train_s_imputed, y_train_cleaned)
print('Logistic best:', log_gs.best_params_, 'CV best score:', round(log_gs.best_score_, 4))

# SVM grid
svm_clf = SVC(kernel='rbf', probability=True, random_state=RANDOM_STATE)
svm_grid = {'C': [0.1, 1, 10, 100], 'gamma': ['scale', 0.01, 0.001]}

svm_gs = GridSearchCV(svm_clf, svm_grid, cv=5, scoring='accuracy', n_jobs=-1)
svm_gs.fit(X_train_s_imputed, y_train_cleaned)
print('SVM best:', svm_gs.best_params_, 'CV best score:', round(svm_gs.best_score_, 4))
```

6. **Model Evaluation**: The performance of the best estimators was evaluated on the unseen test set using accuracy, confusion matrices, and detailed classification reports.

```
# Evaluate on test set
models = {
    'LogisticRegression': log_gs.best_estimator_,
    'SVM': svm_gs.best_estimator_
}

results = {}
for name, model in models.items():
    y_pred = model.predict(X_test_s_imputed)
    acc = accuracy_score(y_test_cleaned, y_pred)
    cm = confusion_matrix(y_test_cleaned, y_pred)
    cr = classification_report(y_test_cleaned, y_pred, digits=4)
    results[name] = dict(acc=acc, cm=cm, cr=cr) # Updated results dictionary
    print(f'=== {name} ===')
    print('Accuracy:', round(acc,4))
    print('\nConfusion matrix:\n', cm)
    print('\nClassification report:\n', cr)
```

7. **Decision Boundary Visualization**: Principal Component Analysis (PCA) was used to reduce the data to two dimensions to visualize the approximate decision boundaries of the trained models.

```
# Plot decision boundaries
def plot_decision_boundary(X, y, model, title):
    # Map string labels to numerical for plotting
    class_mapping = {label: idx for idx, label in enumerate(np.unique(y))}
    y_numeric = y.map(class_mapping)

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    # Convert predicted labels to numerical
    Z_numeric = np.array([class_mapping[label] for label in Z])
    Z_numeric = Z_numeric.reshape(xx.shape)

    plt.contourf(xx, yy, Z_numeric, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y_numeric, edgecolors='k', marker='o', s=20)
    plt.title(title)
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
```

5. Results and Discussion

The final models were evaluated on the preprocessed test set of 169 samples.

5.1. Logistic Regression Results

The GridSearchCV identified the best hyperparameters as {'C': 100, 'solver': 'lbfgs'} with a cross-validation accuracy of 92.83%.

On the test set, the model achieved an accuracy of **94.08%**.

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

           N     0.7692    0.7692    0.7692        13
           P     0.6250    0.5556    0.5882         9
           Y     0.9730    0.9796    0.9763       147

    accuracy                         0.9408       169
   macro avg     0.7891    0.7681    0.7779       169
weighted avg     0.9388    0.9408    0.9397       169
```

5.2. Support Vector Machine (RBF) Results

The best hyperparameters found were {'C': 10, 'gamma': 'scale'} with a cross-validation accuracy of 95.47%.

On the test set, the RBF SVM achieved a higher accuracy of **97.04%**.

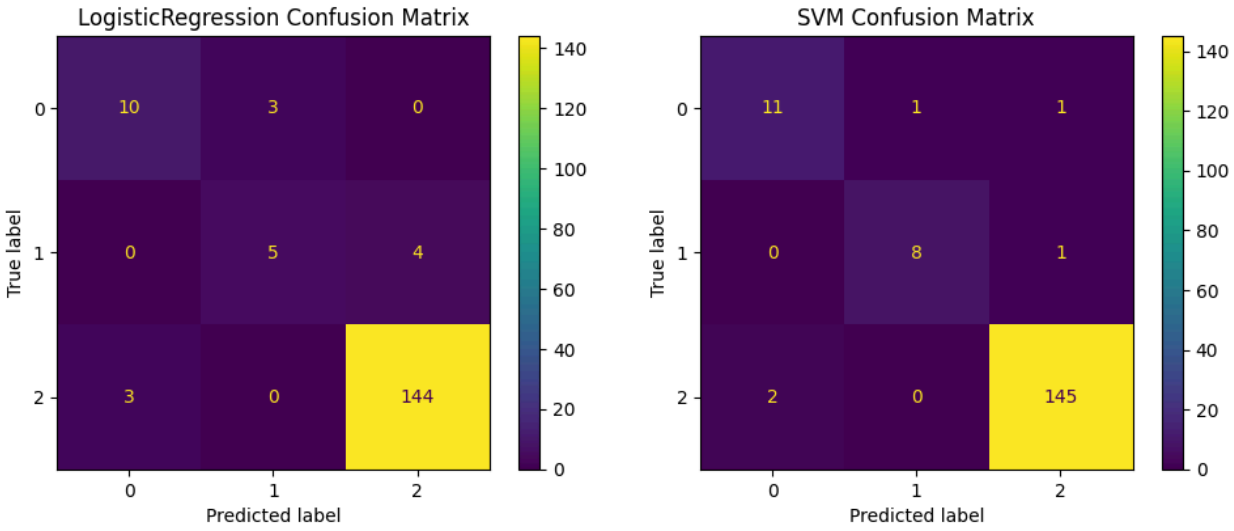
```
Classification report:
              precision    recall  f1-score   support

           N     0.8462    0.8462    0.8462        13
           P     0.8889    0.8889    0.8889         9
           Y     0.9864    0.9864    0.9864       147

    accuracy                         0.9704       169
   macro avg     0.9071    0.9071    0.9071       169
weighted avg     0.9704    0.9704    0.9704       169
```

5.3. Comparison and Visualizations

The **RBF SVM (97.04%)** outperformed the **Logistic Regression (94.08%)** in overall accuracy. The SVM also showed superior precision, recall, and F1-scores for the minority classes ('N' and 'P'), indicating it was better at handling the class imbalance.

Confusion Matrices:

The confusion matrices below provide a visual comparison of the models' predictions :



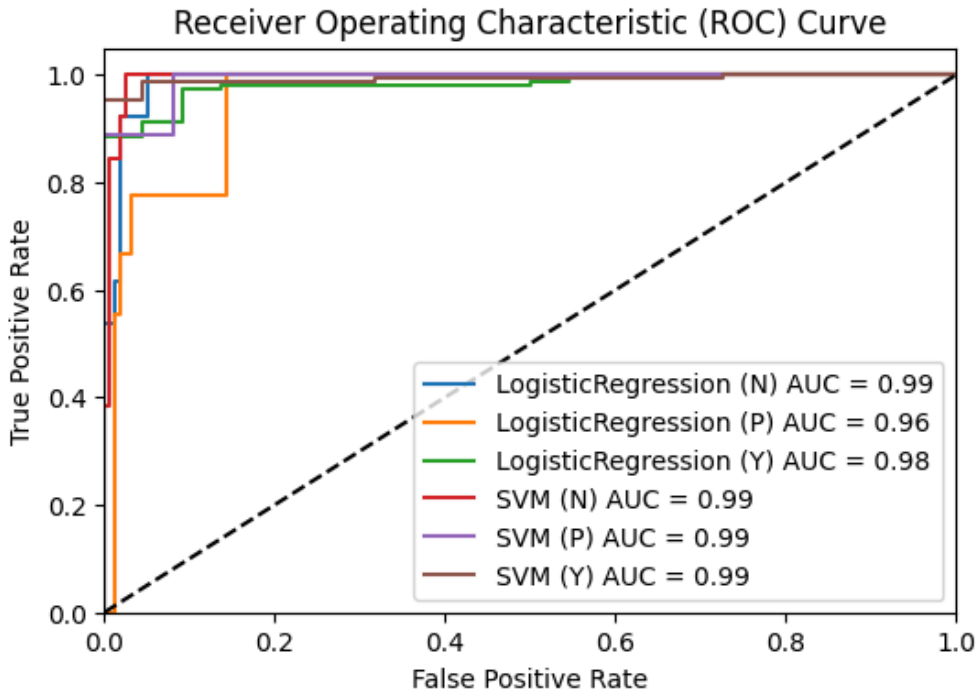
The SVM model had fewer overall misclassifications (5) compared to the Logistic Regression model (10).

Notably, the SVM correctly classified more instances of the pre-diabetic class 'P'.

ROC Curve and AUC Score:

To further assess the models' ability to discriminate between classes, one-vs-rest Receiver Operating Characteristic (ROC) curves were generated, and the Area Under the Curve (AUC) was calculated for each class.

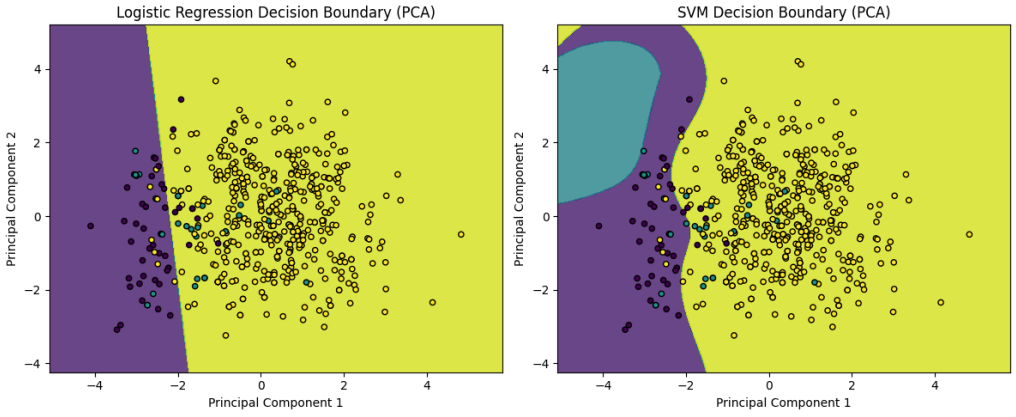
The SVM model achieved higher AUC scores across the board, particularly for the most challenging minority class, 'P' (Pre-diabetic), where it scored 0.99 compared to Logistic Regression's 0.96. These excellent AUC values for the SVM model further confirm its superior diagnostic ability for this classification problem.



Decision Boundary Visualization using PCA

To better understand how the models separate the data, PCA was used to reduce the feature space to two principal components.

The models were retrained on this 2D data. While this reduction led to a slight drop in performance (CV scores: LR = 91.5%, SVM = 90.2%), it allowed for the visualization of their decision boundaries, confirming the non-linear nature of the SVM's separation compared to the linear boundary of the Logistic Regression model.



6. Conclusion

This analysis successfully applied and compared Logistic Regression and RBF SVM for diabetes classification.

After a rigorous preprocessing pipeline that included outlier removal and feature scaling, both models performed well.

However, the **RBF SVM demonstrated superior performance**, achieving a test accuracy of 97.04% and proving more effective at classifying the minority classes. This suggests that the non-linear decision boundary provided by the RBF kernel was better suited for capturing the complex relationships within this dataset.