# Prediction of Loan Approval Using ML Techniques

Zachary Matheson & Anshuman Srivastava

# Outline

I. Background Information

I. Code Build-Up

I. Results

I. Conclusion

# Scope of Project

Objective: Predict loan approval using machine learning techniques

Dataset: Contains applications who previously applied for a property loan

- Income, loan amount, credit history, co-applicant income, education, marital status, dependents, etc

Approach: Utilizing the *loan_data.csv* for data preprocessing, featuring scaling, and model training

Goal: Develop a model to predict loan approval decision and enhance efficiency and accuracy in the loan approval process

# Previous Work

Dataset from [Kaggle.com](Kaggle.com), works done include:

I. Decision tree classifier
   A. Algorithm that splits the data into branches based on features, assigning a class label to each instance
II. Predictive Project
   A. SVM, Ada, Gradient Boosting, and Random Forest models
III. Loan Status Prediction
   A. KNN and GaussianNB models
      1. KNN: accuracy of 71% with f1-score of 0.34
      2. GaussianNB: accuracy of 32% with f1-score of 0.49

# I.
# Background Information

# Cost Function

- Explains how well a model explains the training data

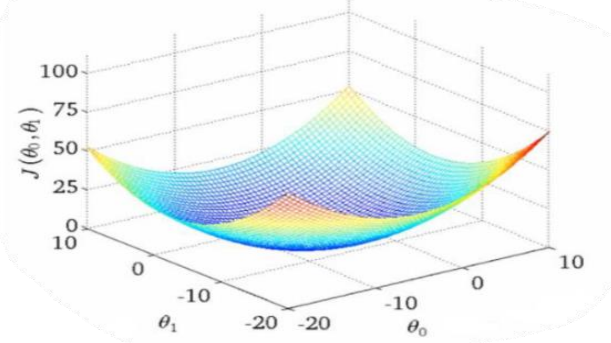



Figure. Visualization of Cost Function

Linear regression: Goal is to minimize cost function

```python
# Cost function
def compute_cost_train(X, y, theta):
    predictions = X.dot(theta)
    errors = np.subtract(predictions, y)
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)
    return J
```

Figure. Code Implementation of Cost Function

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_i (h_\theta(x_i) - y_i)^2$

$\emptyset_0, \emptyset_1$: Parameters

m: total # of training samples

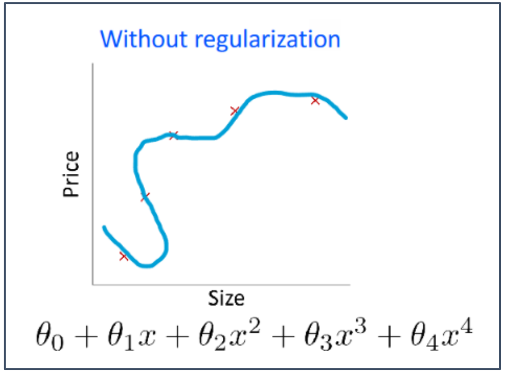
Hypothesis: $h_\emptyset(x) = \emptyset_0 + \emptyset_1(x)$

$(x_i, y_i)$: $i_{th}$ training sample pair

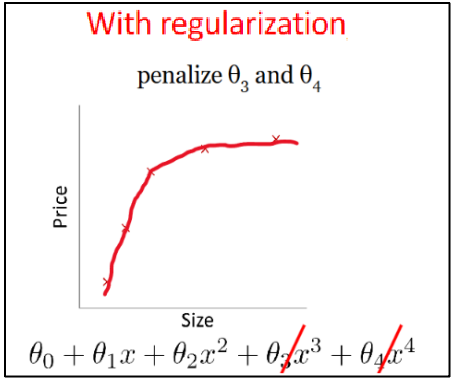
# Linear Regression <L2-Regularization>

A method for automatically controlling complexity of the learning hypothesis

- Penalize large values of Øj
  - Addresses overfitting




$$J(\theta) = \frac{1}{2m}\left[\underbrace{\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2}_{\text{Model to fit to data}} + \underbrace{\lambda \sum_{j=1}^{n}\theta_j^2}_{\text{Regularization}}\right]$$



# Gradient Descent

Gradient is a vector, so it has:

- A direction
- A magnitude

```
# Gradient descent
def gradient_descent(X, Xtest, y, y_test, theta, alpha, iterations):
    m_train = len(y_train)
    m_test = len(y_test)
    cost_history_train = np.zeros(iterations)
    cost_history_test = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta)
        errors = np.subtract(predictions, y)
        gradients = (2/m_train) * X.transpose().dot(errors)
        theta -= alpha * gradients

        train_loss = (1/m_train) * np.sum(np.square(X.dot(theta) - y))
        cost_history_train[i] = train_loss

        val_loss = (1/m_test) * np.sum(np.square(Xtest.dot(theta) - y_test))
        cost_history_test[i] = val_loss

    return theta, cost_history_train, cost_history_test
```

Figure. Code Implementation of Gradient Descent

It’s a iterative optimization algorithm to find the minimum of any function

$$\theta_j := \theta_j' - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \frac{1}{m} \boldsymbol{X}^T (\mathbf{X}.\boldsymbol{\theta} - \boldsymbol{y})$$

Øj: New value

Øj’: Current value

α: Learning rate

$\frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$ :

Gradient(parti $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{1}{m} \boldsymbol{X}^T (\mathbf{X}.\boldsymbol{\theta} - \boldsymbol{y})$

# Feature Scaling

StandardScaler()

- Rescalling features to have zero mean and unit variance

Mean feature of j: $\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_j^{(i)}$

Replacing each value: $x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$

$s_j$: Standard deviation of

σ: standard deviation (pop,
X: datapoint value
μ: Population mean
N: Population size

$$\sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{N}}$$

MinMaxScaler()

- Values are shifted and rescaled so they end up in ranging between 0-1

Normalization: $X' = \frac{x - \min(x)}{\max(x) - \min(x)}$

X': New value
x: Original Value

# Logistic Function

A smooth, convex cost function

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}\right) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

$$g(z) = \frac{1}{1+e^{-z}}$$

Logistic regression cost function:

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Compact form:

$$Cost\ (h_\theta(x), y) = -y\log\big(h_\theta(x)\big) - (1-y)\log(1 - h_\theta(x))$$

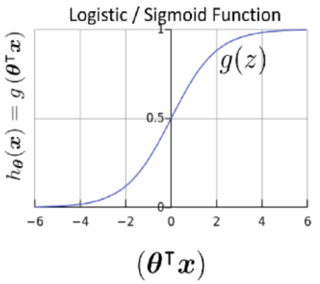


Figure. Visualization of Logistic Function

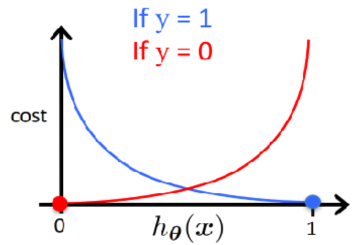


Figure. Visualization of Logistic Cost Function

```
# Logistic Regression
def perform_logistic_regression(x_train, y_train, x_test, y_test):
    classifier = LogisticRegression(random_state=100)
    classifier.fit(x_train, y_train)

    # Make predictions
    y_pred = classifier.predict(x_test)

    # Find confusion matrix
    cnf_matrix = confusion_matrix(y_test, y_pred)
    cnf_matrix

    # Find accuracy, precision, and recall
    print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
    print("Precision:", metrics.precision_score(y_test, y_pred))
    print("Recall:", metrics.recall_score(y_test, y_pred))
    print("F1 Score:", metrics.f1_score(y_test, y_pred))

    return cnf_matrix
```

# Confusion Matrix

Represents the predicted summary in matrix form

**Accuracy** = (TP+TN)/(TP+TN+FP+FN)
**Precision** = TP/(TP+FP)
**Recall** = TP/(TP+FN)
**Specificity** = TN/(TN+FP)
**F1 Score** =
2*(Precision*Recall)/(Precision+Recall)

A high F1 score indicates a higher accuracy >=1

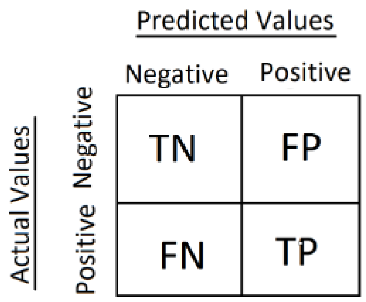



Figure. Confusion Matrix

```
# Confusion Matrix
def plot_confusion_matrix(confusion_matrix, neg_label, pos_label):
    # Visualize the confusion matrix using a heatmap
    class_names=[neg_label, pos_label] # Name  of classes
    fig, ax = plt.subplots()
    tick_marks = np.arange(len(class_names))
    plt.xticks(tick_marks)
    plt.yticks(tick_marks)

    # Create heatmap
    sns.heatmap(pd.DataFrame(confusion_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
    ax.xaxis.set_label_position("top")
    ax.set_xticklabels(class_names)
    ax.set_yticklabels(class_names)
    plt.tight_layout()
    plt.title('Confusion matrix', y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
```

# Naive Bayes Classifier

Probably of 2 events (A & B) happening P(A∩B) is the same as P(A) * P(B) given that A has occurred P(B|A)

Bayes Rule

$$\boxed{P(A|B)} = \boxed{P(A)} * \left( \boxed{P(B|A)} / \boxed{P(B)} \right)$$

Probability of occuring

Likelihood

Prior

Posterior Probability

Log-probabilities to prevent underflow: $\arg\max_{y_k} \; \log P(Y = y_k) + \sum_{j=1}^{n} \log P(X_j = x_j \mid Y = y_k)$

# II.
# Code Build-Up

# Plan of Code

- Preprocess the information:
  - Cleaning the data
    - Handling missing data or outliers
    - Providing categorical variables for column variables
- Split the dataset into training and test sets
  - Size of **training is 70%** and **testing is 30%**
  - Training set includes all columns excluding **Loan_ID** and **Loan_Status**
  - Testing set is only the **Loan_Status** indication if the applicant was approved or rejected

# Preprocessing Code

```
# Mapping
loan['Loan_Status'] = loan['Loan_Status'].map({'yes': 1, 'no': 0})
loan['Property_Area'] = loan['Property_Area'].map({'Semirural': 2, 'Rural': 1, 'Urban': 0})
loan['Self_Employed'] = loan['Self_Employed'].map({'yes': 1, 'no': 0})
loan['Education'] = loan['Education'].map({'Graduate': 1, 'Not Graduate': 0})
loan['Married'] = loan['Married'].map({'yes': 1, 'no': 0})
loan['Gender'] = loan['Gender'].map({'Male': 1, 'Female': 0})

# Cleaning dataset
loan = loan.dropna()
#Resetting index
loan.reset_index(drop=True, inplace=True)

num_elements = loan.shape[0]
print("Number of elements remaining: ", num_elements) # We lose half of the elements
loan.head()
```

Label Encoding

Handling missing values

```
Mounted at /content/drive
Number of elements remaining:  185
```

| | Loano_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History | Property_Area | Loan_Status |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | LP001003 | 1.0 | 1 | 1.0 | 1 | 0.0 | 4583 | 1508.0 | 128 | 360.0 | 1.0 | 1.0 | 0 |
| 1 | LP001005 | 1.0 | 1 | 0.0 | 1 | 1.0 | 3000 | 0.0 | 66 | 360.0 | 1.0 | 0.0 | 1 |
| 2 | LP001006 | 1.0 | 1 | 0.0 | 0 | 0.0 | 2583 | 2358.0 | 120 | 360.0 | 1.0 | 0.0 | 1 |
| 3 | LP001008 | 1.0 | 0 | 0.0 | 1 | 0.0 | 6000 | 0.0 | 141 | 360.0 | 1.0 | 0.0 | 1 |
| 4 | LP001013 | 1.0 | 1 | 0.0 | 0 | 0.0 | 2333 | 1516.0 | 95 | 360.0 | 1.0 | 0.0 | 1 |

# Training and Testing Split

```
# Splitting the dataset into training and testing sets
np.random.seed(0)
df_train, df_test = train_test_split(loan, train_size = 0.7, test_size = 0.3, random_state = 0)
```

Splitting the data

```
# Numerical variables that will be used for training
num_varsa = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status' ]
```

Variables used

```
# Separate features and labels for training
X_train = df_train[num_varsa].values[:, :-1]
y_train = df_train['Loan_Status'].values
m_train = len(y_train)
n_train = len(X_train)
# Seperate features and labels for test
X_test = df_test[num_varsa].values[:, :-1]
y_test = df_test['Loan_Status'].values
#print(y_train)
m_test = len(y_test)
n_test = len(X_test)
```

Separating the data into training and testing

```
# Initializing
X_0train = np.ones((m_train,1))
X_0test = np.ones((m_test,1))

X_1train = X_train.reshape(m_train, 11)
X_1test = X_test.reshape(m_test, 11)

Xtrain = np.hstack((X_0train, X_1train))
Xtest = np.hstack((X_0test, X_1test))

theta = np.zeros(12)

m = len(loan)
```

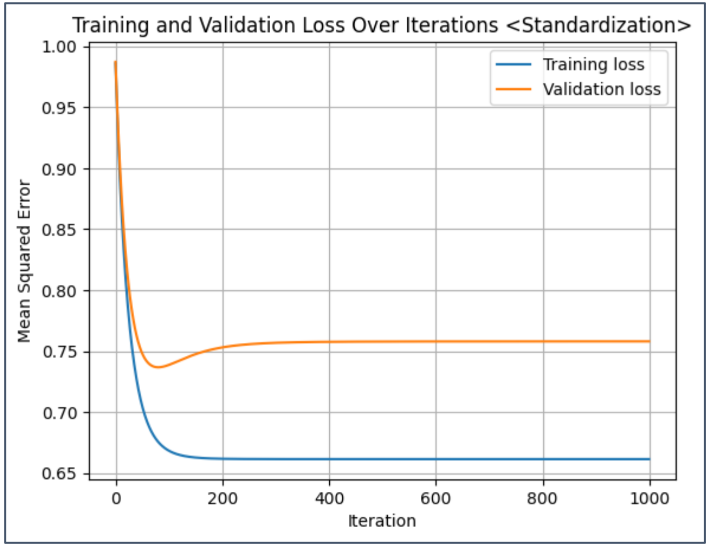
Initializing values

# III.
# Results

# Standardization & Normalization

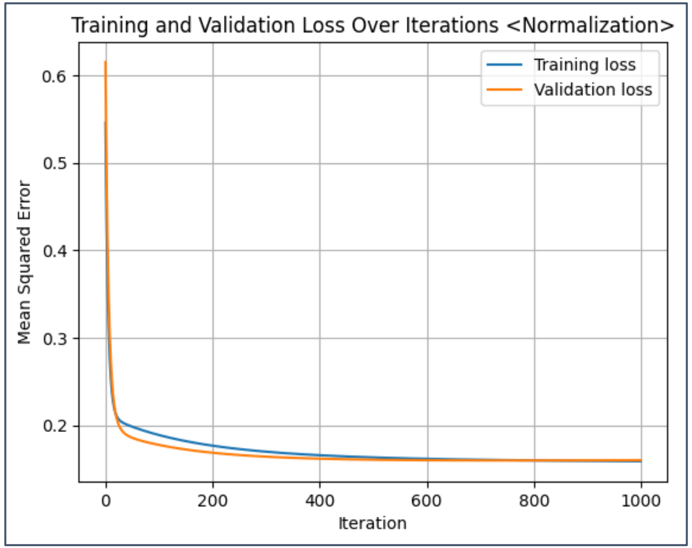
α (step size) = 0.01

Exhibits overfitting




α (step size)= 0.01

Exhibits an optimized system



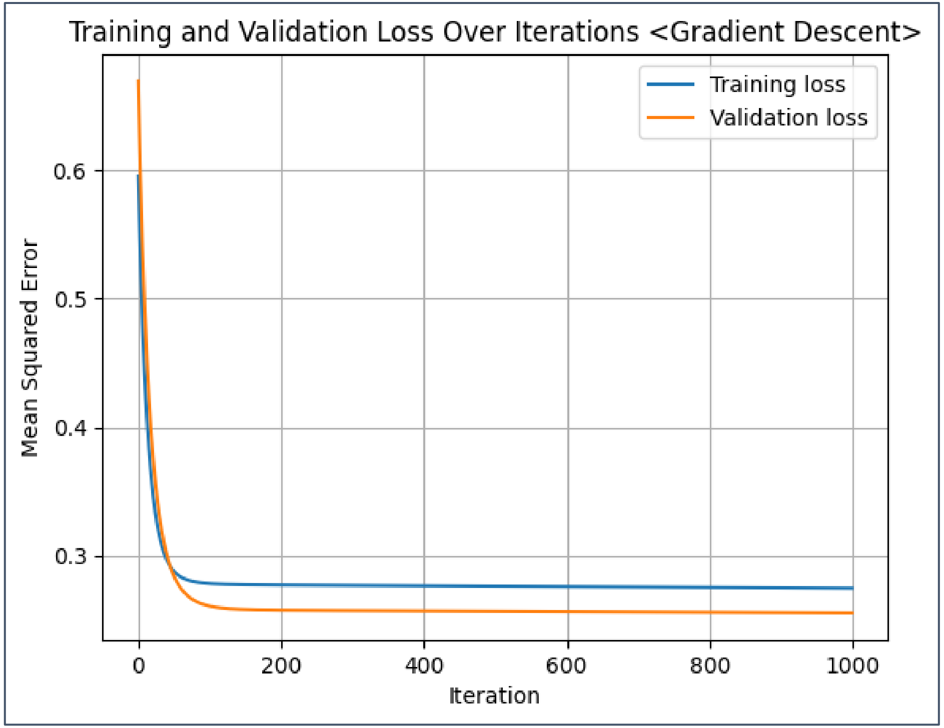

UNIVERSITY OF NORTH CAROLINA CHARLOTTE

# Gradient Descent

α (step size)= 0.000000001

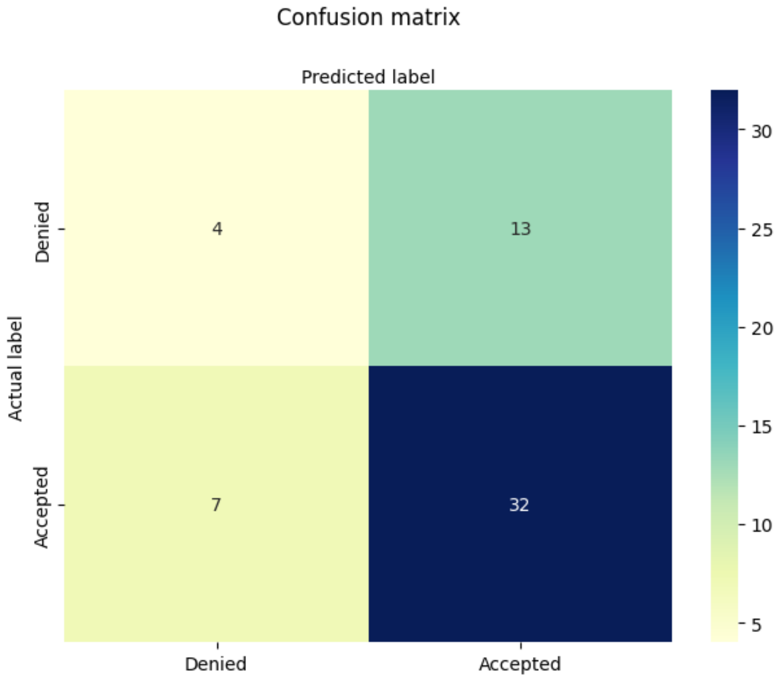
Exhibits data leakage

**Note:** A very small alpha value ensures that updates are gradual over each iteration, reducing the risk of overshooting the minimum of the cost function.



# Logistic Regression

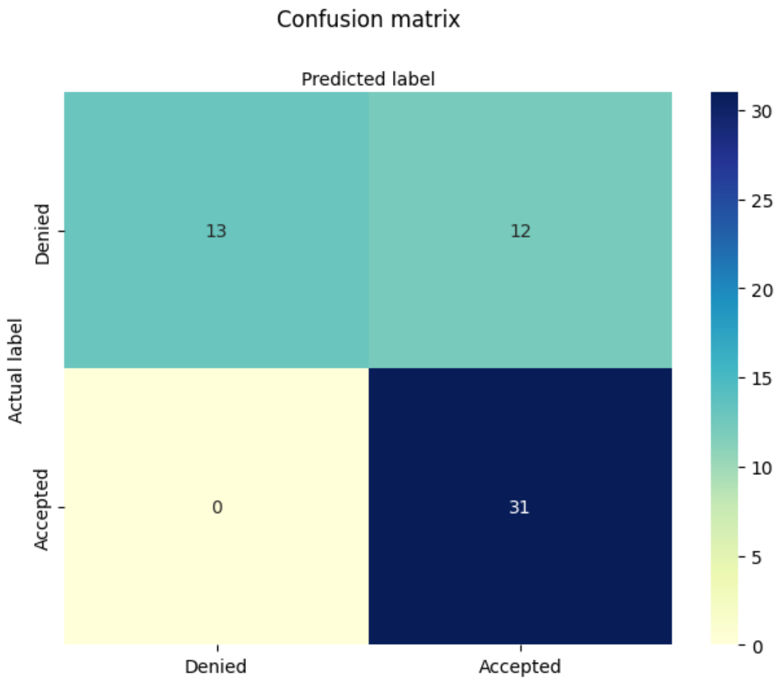
| | |
|---|---|
| Accuracy | 64.29% |
| Precision | 71.11% |
| Recall | 82.05% |
| F1-Score | 76.19% |



# Logistic Regression with penalty

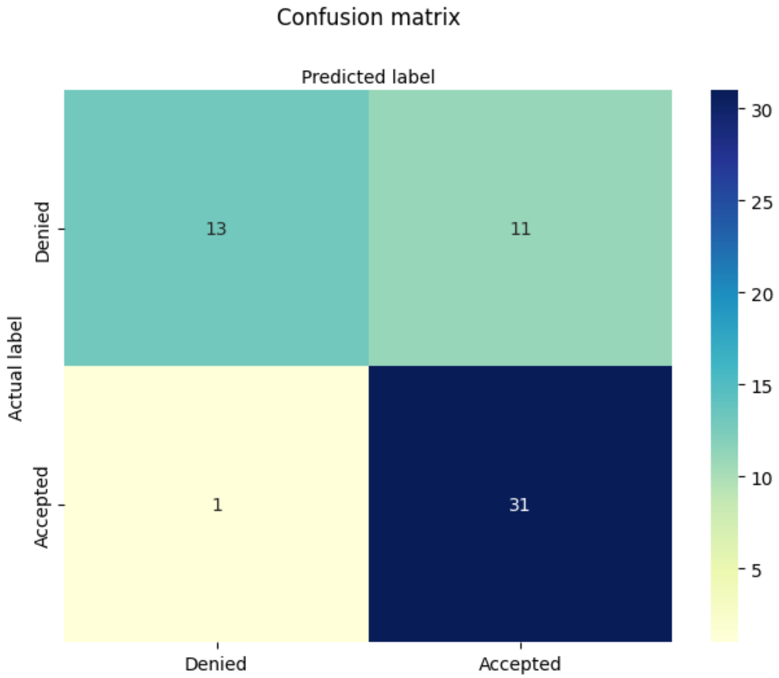
Penalty: C=1

| | |
|---|---|
| Accuracy | 78.57% |
| Precision | 72.09% |
| Recall | 100% |
| F1-Score | 83.78% |



# Naive Bayesian

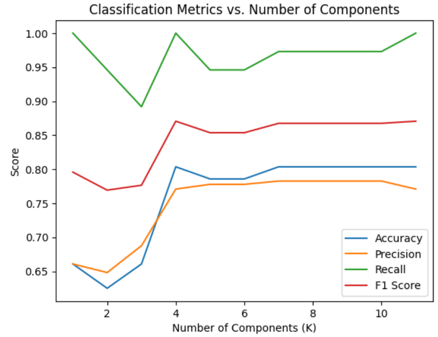
| | |
|---|---|
| Accuracy | 78.57% |
| Precision | 73.81% |
| Recall | 96.88% |
| F1-Score | 83.78% |

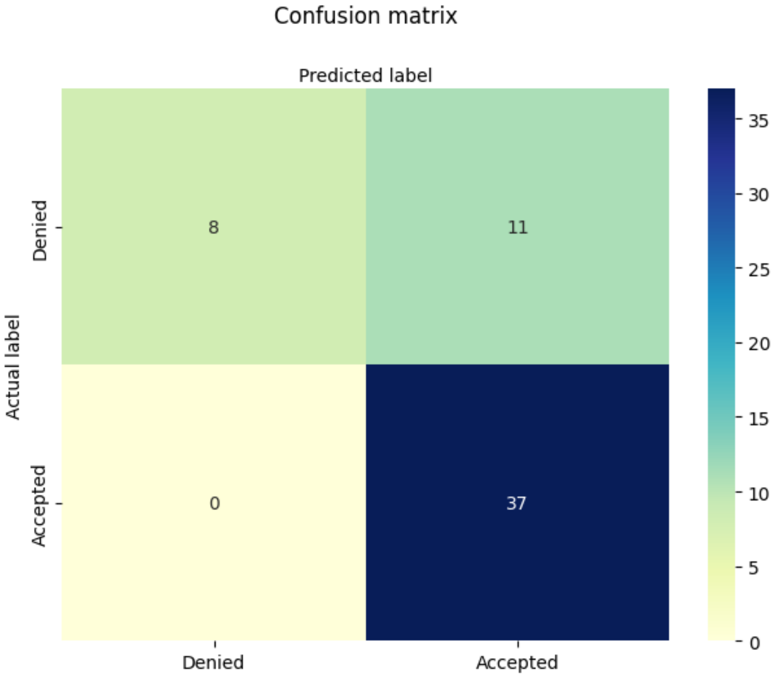


# Naive Bayesian with PCA feature extraction (Normalized)

K = 11

| | |
|---|---|
| Accuracy | 80.36% |
| Precision | 77.08% |
| Recall | 100% |
| F1-Score | 87.06% |

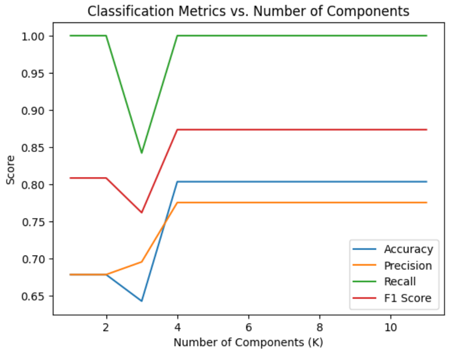


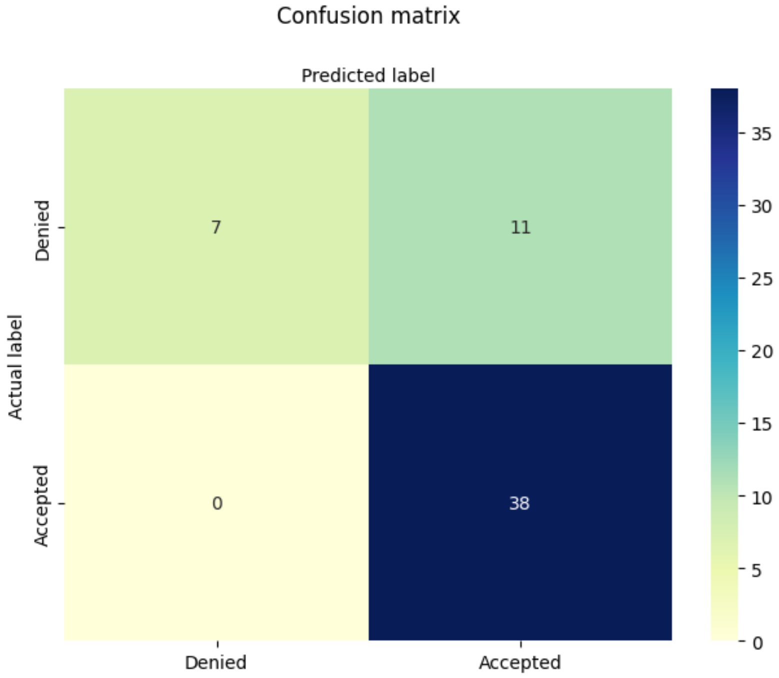



# Logistic Regression with PCA feature extraction (Normalized)

K=4-11

| | |
|---|---|
| Accuracy | 80.36% |
| Precision | 77.55% |
| Recall | 100% |
| F1-Score | 87.36% |



# IV.
# Conclusion

# Outcome

Logistic Regression and Naive Bayesian using PCA extraction (normalized) both perform well.

Logistic Regression

| | |
|---|---|
| Accuracy | 80.36% |
| Precision | 77.55% |
| Recall | 100% |
| F1-Score | 87.36% |

Naive Bayesian

| | |
|---|---|
| Accuracy | 80.36% |
| Precision | 77.08% |
| Recall | 100% |
| F1-Score | 87.06% |

# Predictive Model

```python
# Function to collect user input for questionnaire
def get_user_input():
    gender = input("Gender (Male/Female): ").capitalize()
    married = input("Married (Yes/No): ").capitalize()
    dependents = int(input("Number of Dependents: "))
    education = input("Education (Graduate/Not Graduate): ").capitalize()
    self_employed = input("Self Employed (Yes/No): ").capitalize()
    applicant_income = float(input("Applicant Income: "))
    coapplicant_income = float(input("Coapplicant Income: "))
    loan_amount = float(input("Loan Amount: "))
    loan_amount_term = float(input("Loan Amount Term: "))
    credit_history = float(input("Credit History (0 or 1): "))
    property_area = input("Property Area (Semirural/Rural/Urban): ").capitalize()

    return gender, married, dependents, education, self_employed, applicant_income,
    coapplicant_income, loan_amount, loan_amount_term, credit_history, property_area

# Collect user input
gender, married, dependents, education, self_employed, applicant_income,
coapplicant_income, loan_amount, loan_amount_term, credit_history, property_area = get_user_input()

# Convert user input to numeric values based on mappings
gender = 1 if gender == 'Male' else 0
married = 1 if married == 'Yes' else 0
education = 1 if education == 'Graduate' else 0
self_employed = 1 if self_employed == 'Yes' else 0
property_area = {'Semirural': 2, 'Rural': 1, 'Urban': 0}[property_area]

# Perform PCA on user input
input_data = [[gender, married, dependents, education, self_employed,
               applicant_income, coapplicant_income, loan_amount, loan_amount_term, credit_history, property_area]]
input_data_pca = pca.transform(input_data)

# Predict using Gaussian Naive Bayes classifier
prediction = classifier.predict(input_data_pca)

# Output prediction
print("Loan Approval Prediction:", "Approved" if prediction[0] == 1 else "Denied")
```

LoanID: LP001138

```
Gender (Male/Female): Male
Married (Yes/No): Yes
Number of Dependents: 1
Education (Graduate/Not Graduate): Graduate
Self Employed (Yes/No): No
Applicant Income: 5649
Coapplicant Income: 0
Loan Amount: 44
Loan Amount Term: 360
Credit History (0 or 1): 1
Property Area (Semirural/Rural/Urban): Urban
Loan Approval Prediction: Approved
```

LoanID: LP001813

```
Gender (Male/Female): Male
Married (Yes/No): No
Number of Dependents: 0
Education (Graduate/Not Graduate): Graduate
Self Employed (Yes/No): Yes
Applicant Income: 6050
Coapplicant Income: 4333
Loan Amount: 120
Loan Amount Term: 180
Credit History (0 or 1): 1
Property Area (Semirural/Rural/Urban): Urban
Loan Approval Prediction: Denied
```

# Future Work

Future work could explore advanced machine learning algorithms like:

- Random Forests, neural networks or SVMs to enhance accuracy.

Also, an interactive application could be developed for real-time loan approval predictions and improved user experience by integrating APIs for automatic data retrieval.

# Thank You!
# Questions?