

VIT BHOPAL UNIVERSITY

School of Computing Science and Engineering

Bhopal-Indore Highway, Kothrikalan, Sehore

Madhya Pradesh - 466114

CSA4008 - APPLIED MACHINE LEARNING

REG.NO : 23BAI10165

NMAE : Suyash Soni

BRANCH : CSE – AI & ML

SEMESTER: Fall Semester 2025-26

INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| Ex.NO | DATE | EXPERIMENT NAME | PAGE NO. |
|  |  | **Implement Decision Tree learning** |  |
|  |  | **Implement Logistic Regression** |  |
|  |  | **Implement classification using Multilayer perceptron** |  |
|  |  | **Implement classification using Multilayer perceptron** |  |
|  |  | **Implement Adaboost Algorithm** |  |
|  |  | **Implement Bagging using Random Forests** |  |
|  |  | **Implement K-means Clustering to Find Natural Patterns in Data** |  |
|  |  | **Implement Principle Component Analysis for Dimensionality Reduction** |  |
|  |  | **Evaluating ML algorithm with balanced and unbalanced datasets** |  |
|  |  | **Evaluating ML algorithm with balanced and unbalanced datasets** |  |

|  |  |
| --- | --- |
| **EXP.NO: 02** | **Implement Logistic Regression** |
| **DATE:** |

**AIM : To implement and evaluate a logistic regression model for predicting stock price direction (up/down) using historical stock market data and analyze model performance using various evaluation metrics and visualization techniques.**

**PROCEDURE**

**Logistic Regression**

Logistic regression is a statistical method used for binary classification problems. Unlike linear regression, it uses the logistic function (sigmoid) to map any real-valued input to a value between 0 and 1, making it suitable for probability estimation.

Mathematical Foundation:

* Sigmoid Function: σ(z) = 1/(1 + e^(-z))
* Linear Combination: z = β₀ + β₁x₁ + β₂x₂ + ... + βₙxₙ
* Probability: P(y=1|x) = σ(z)

Key Characteristics:

* Output ranges from 0 to 1
* Uses maximum likelihood estimation
* Assumes linear relationship between features and log-odds
* No assumption of normality for features

**Dataset Description**

Source: Stock market historical data Features:

* Date: Trading date
* Close/Last: Closing price
* Volume: Number of shares traded
* Open: Opening price
* High: Highest price of the day
* Low: Lowest price of the day

Dataset Statistics:

* Total Records: 2,518
* Date Range: 2020-02-28 to [end date]
* Missing Values: None

**Methodology**

4.1 Data Preprocessing

1. Data Type Conversion:
   * Convert date column to datetime format
   * Ensure price columns are numeric (float64)
   * Handle any missing values
2. Feature Engineering:
   * Create target variable: Price\_Up (1 if next day's price > current day's price, 0 otherwise)
   * Price\_Range = High - Low
   * Price\_Change = Close - Open
   * Price\_Change\_Pct = ((Close - Open) / Open) × 100
   * Volume\_MA\_5 = 5-day moving average of volume
   * Volume\_Above\_Avg = Binary indicator for above-average volume

4.2 Model Development

1. Feature Selection:
   * Selected features: Volume, Price\_Range, Price\_Change, Price\_Change\_Pct, Volume\_MA\_5, Volume\_Above\_Avg
2. Data Splitting:
   * Training set: 80%
   * Test set: 20%
   * Stratified split to maintain class balance
3. Feature Scaling:
   * StandardScaler to normalize features
   * Mean = 0, Standard deviation = 1
4. Model Training:
   * Algorithm: Logistic Regression
   * Solver: Default (liblinear for small datasets)
   * Random state: 42 for reproducibility

**PROGRAM**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import \*

import warnings

warnings.filterwarnings('ignore')

df = pd.read\_csv('D:\\ML PROJECTS\\AML lab\\dataset\\HistoricalQuotes.csv')

print("Column names:")

print(df.columns.tolist())

print("\nColumn names with quotes to see spaces:")

for col in df.columns:

print(f"'{col}'")

df.columns = df.columns.str.strip()

df['Close/Last'] = pd.to\_numeric(df['Close/Last'].str.replace('$', ''))

df['Open'] = pd.to\_numeric(df['Open'].str.replace('$', ''))

df['High'] = pd.to\_numeric(df['High'].str.replace('$', ''))

df['Low'] = pd.to\_numeric(df['Low'].str.replace('$', ''))

df['Date'] = pd.to\_datetime(df['Date'])

def clean\_price\_column(column):

if column.dtype == 'object':

return pd.to\_numeric(column.astype(str).str.replace('$', '').str.replace(',', ''))

else:

return column

df['Close/Last'] = clean\_price\_column(df['Close/Last'])

df['Open'] = clean\_price\_column(df['Open'])

df['High'] = clean\_price\_column(df['High'])

df['Low'] = clean\_price\_column(df['Low'])

df['Date'] = pd.to\_datetime(df['Date'])

print("After conversion:")

print(df.dtypes)

print(df.head())

df['Price\_Up'] = (df['Close/Last'].shift(-1) > df['Close/Last']).astype(int)

df['Price\_Range'] = df['High'] - df['Low']

df['Price\_Change'] = df['Close/Last'] - df['Open']

df['Price\_Change\_Pct'] = ((df['Close/Last'] - df['Open']) / df['Open']) \* 100

df['Volume\_MA\_5'] = df['Volume'].rolling(window=5).mean()

df['Close\_MA\_5'] = df['Close/Last'].rolling(window=5).mean()

df['Volume\_Above\_Avg'] = (df['Volume'] > df['Volume\_MA\_5']).astype(int)

print("New features created:")

print(df[['Date', 'Close/Last', 'Price\_Up', 'Price\_Range', 'Price\_Change', 'Volume\_MA\_5']].head(10))

feature\_columns = ['Volume', 'Price\_Range', 'Price\_Change', 'Price\_Change\_Pct', 'Volume\_MA\_5', 'Volume\_Above\_Avg']

df\_clean = df.dropna()

X = df\_clean[feature\_columns]

y = df\_clean['Price\_Up']

print(f"Dataset shape: {X.shape}")

print(f"Target distribution:")

print(y.value\_counts())

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

model = LogisticRegression(random\_state=42)

model.fit(X\_train\_scaled, y\_train)

y\_pred = model.predict(X\_test\_scaled)

y\_pred\_proba = model.predict\_proba(X\_test\_scaled)[:, 1]

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.4f}")

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

print("\nConfusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

feature\_importance = pd.DataFrame({

'feature': feature\_columns,

'coefficient': model.coef\_[0],

'abs\_coefficient': abs(model.coef\_[0])

}).sort\_values('abs\_coefficient', ascending=False)

print("Feature Importance:")

print(feature\_importance)

results\_df = pd.DataFrame({

'Actual': y\_test.values,

'Predicted': y\_pred,

'Probability': y\_pred\_proba

}).head(10)

print("\nSample Predictions:")

print(results\_df)

def plot\_confusion\_matrix(y\_true, y\_pred, title='Confusion Matrix'):

cm = confusion\_matrix(y\_true, y\_pred)

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

xticklabels=['Price Down (0)', 'Price Up (1)'],

yticklabels=['Price Down (0)', 'Price Up (1)'],

cbar\_kws={'label': 'Count'})

plt.title(title, fontsize=16, pad=20)

plt.xlabel('Predicted', fontsize=12)

plt.ylabel('Actual', fontsize=12)

plt.tight\_layout()

plt.show()

tn, fp, fn, tp = cm.ravel()

print(f"\nConfusion Matrix Breakdown:")

print(f"True Negatives (TN): {tn}")

print(f"False Positives (FP): {fp}")

print(f"False Negatives (FN): {fn}")

print(f"True Positives (TP): {tp}")

precision = tp / (tp + fp) if (tp + fp) > 0 else 0

recall = tp / (tp + fn) if (tp + fn) > 0 else 0

specificity = tn / (tn + fp) if (tn + fp) > 0 else 0

print(f"\nAdditional Metrics:")

print(f"Precision: {precision:.4f}")

print(f"Recall (Sensitivity): {recall:.4f}")

print(f"Specificity: {specificity:.4f}")

plot\_confusion\_matrix(y\_test, y\_pred, 'Stock Price Direction Prediction - Confusion Matrix')

def plot\_feature\_correlation(X, feature\_names):

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

correlation\_matrix = np.corrcoef(X.T)

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', center=0,

xticklabels=feature\_names, yticklabels=feature\_names)

plt.title('Feature Correlation Matrix')

plt.tight\_layout()

plt.show()

plot\_feature\_correlation(X\_train\_scaled, feature\_columns)

**INPUT**

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2518 entries, 0 to 2517

Data columns (total 13 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Date 2518 non-null datetime64[ns]

1 Close/Last 2518 non-null float64

2 Volume 2518 non-null int64

3 Open 2518 non-null float64

4 High 2518 non-null float64

5 Low 2518 non-null float64

6 Price\_Up 2518 non-null int32

7 Price\_Range 2518 non-null float64

8 Price\_Change 2518 non-null float64

9 Price\_Change\_Pct 2518 non-null float64

10 Volume\_MA\_5 2514 non-null float64

11 Close\_MA\_5 2514 non-null float64

12 Volume\_Above\_Avg 2518 non-null int32

dtypes: datetime64[ns](1), float64(9), int32(2), int64(1)

memory usage: 236.2 KB

------------------------------------------------------------------------------

Dataset shape: (2514, 6)

Target distribution:

Price\_Up

0 1332

1 1182

Name: count, dtype: int64

**OUTPUT**

Accuracy: 0.8270

Classification Report:

precision recall f1-score support

0 0.82 0.86 0.84 267

1 0.83 0.79 0.81 236

accuracy 0.83 503

macro avg 0.83 0.83 0.83 503

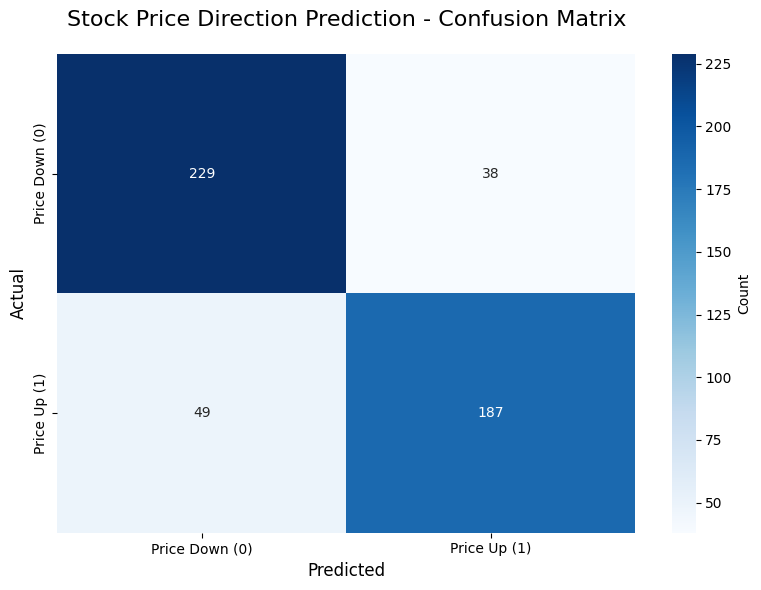
weighted avg 0.83 0.83 0.83 503

Confusion Matrix:

[[229 38]

[ 49 187]]

----------------------------------------------------------------------------------



Confusion Matrix Breakdown:

True Negatives (TN): 229

False Positives (FP): 38

False Negatives (FN): 49

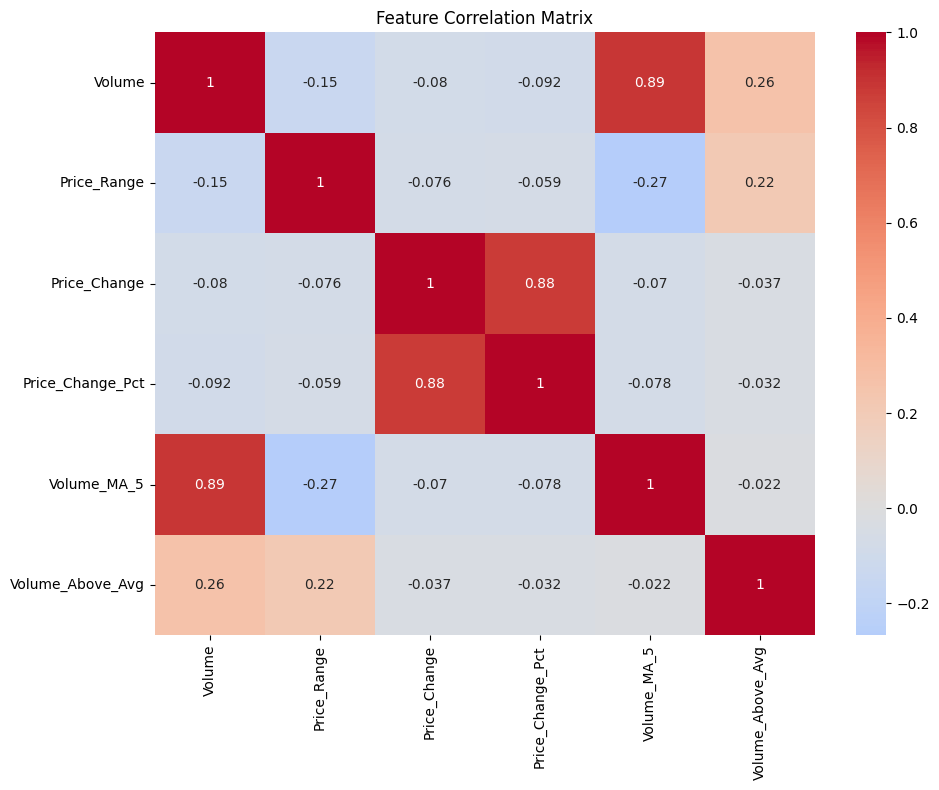
True Positives (TP): 187

Additional Metrics:

Precision: 0.8311

Recall (Sensitivity): 0.7924

Specificity: 0.8577



**RESULT**

Feature Importance:

feature coefficient abs\_coefficient

3 Price\_Change\_Pct -2.001839 2.001839

2 Price\_Change -0.936801 0.936801

1 Price\_Range 0.377739 0.377739

0 Volume 0.054065 0.054065

5 Volume\_Above\_Avg -0.051708 0.051708

4 Volume\_MA\_5 -0.021862 0.021862

Sample Predictions:

Actual Predicted Probability

0 1 1 0.962132

1 1 1 0.761882

2 1 1 0.913239

3 1 1 0.623813

4 1 1 0.602739

5 0 0 0.308557

6 1 0 0.380055

7 1 1 0.729258

8 0 0 0.028868

9 1 1 0.979649