

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: 01** | **IMPLEMENT DECISION TREE LEARNING** |
| **DATE:** |

**AIM**

Implement Decision Tree learning

**PROCEDURE**

The following steps outline the procedure applied in the provided code to use a Decision Tree Classifier for predicting stock price movement (Up, Down, Sideways):

**Data Loading and Initial Inspection:**

* Load the apple\_stock.csv and HistoricalQuotes.csv datasets into pandas DataFrames.
* Perform initial data inspection using df.tail(), df.info(), and df.head(10) to understand the data structure and content.

**Dataset Loading (Augmented Data):**

* Load an augmented\_financial\_data.csv file, which is presumably a pre-processed or enhanced version of the stock data, as the primary DataFrame (df) for the analysis.

**Date Handling and Sorting:**

* Convert the 'Date' column to datetime objects using pd.to\_datetime().
* Sort the DataFrame by 'Date' in ascending order and reset the index to ensure proper time-series analysis.

**Target Variable Engineering (Price Direction):**

* Calculate Next\_Day\_Close by shifting the 'Close' price by one day.
* Compute Price\_Change\_Pct (percentage change from current day's close to next day's close).
* Define a threshold (e.g., 0.01 for 1%).

1. Create the multi-class target variable Price\_Direction:
2. (Up): If Price\_Change\_Pct is greater than threshold.
3. (Down): If Price\_Change\_Pct is less than -threshold.
4. (Sideways): Otherwise (price change within the [-threshold, threshold] range).

* Map numerical Price\_Direction values to descriptive labels ('Down', 'Sideways', 'Up') in a new column Direction\_Label for better interpretation.
* Print the distribution of the Price\_Direction to understand class balance.

**Feature Engineering (Technical Indicators):**

* Calculate Price\_Change and Volume\_Change (daily percentage changes).
* Compute Simple Moving Averages (SMA) for 5-day (SMA\_5) and 10-day (SMA\_10) windows.
* Create binary indicators Price\_Above\_SMA5 and Price\_Above\_SMA10 (1 if closing price is above the respective SMA, 0 otherwise).
* Calculate Volatility (5-day rolling standard deviation of Price\_Change).
* Derive High\_Low\_Ratio and Open\_Close\_Ratio.
* Compute Price\_Position within the daily range (Close - Low) / (High - Low).
* Create Prev\_Day\_Up (1 if previous day's close was higher than the day before, 0 otherwise) and Prev\_Day\_Volume (previous day's volume).
* Calculate momentum indicators: Price\_Momentum\_3 and Volume\_Momentum\_3 (ratio of current to 3-day shifted values).

**Feature Selection and Data Cleaning:**

* Define a list features containing the selected technical indicators and raw price/volume data.
* Remove rows with any NaN values from the DataFrame using df.dropna() to create df\_clean. This is crucial because many technical indicators involve rolling windows or shifts, which introduce NaN values at the beginning of the series.
* Separate the features (X) and the target variable (y) from the df\_clean DataFrame.

**Time-Based Data Splitting:**

* Split the data into training and testing sets based on time, which is essential for time-series forecasting to avoid data leakage.
* A split\_index is calculated (80% for training, 20% for testing).
* X\_train, X\_test, y\_train, y\_test are created by slicing the data up to the split\_index for training and from split\_index onwards for testing.

**Decision Tree Model Initialization and Training:**

* Initialize a DecisionTreeClassifier with specific hyperparameters:
* max\_depth=8: Limits the tree's depth to prevent overfitting.
* min\_samples\_split=10: Minimum number of samples required to split an internal node.
* min\_samples\_leaf=5: Minimum number of samples required to be at a leaf node.
* max\_features='sqrt': Considers sqrt(n\_features) when looking for the best split.
* class\_weight='balanced': Automatically adjusts weights inversely proportional to class frequencies to handle potential class imbalance.
* random\_state=42: For reproducibility of results.

**Train the model using the model.fit(X\_train, y\_train) method.**

**Prediction:**

* Make predictions on the test set using y\_pred = model.predict(X\_test).
* Obtain probability estimates for each class using y\_pred\_proba = model.predict\_proba(X\_test).

**Time Series Cross-Validation**:

* Implement TimeSeriesSplit with n\_splits=5 for robust evaluation.
* Iterate through the splits, train the model on each training fold, and evaluate accuracy on the corresponding test fold.
* Calculate and print the mean and standard deviation of cross-validation accuracies.

**Model Evaluation:**

* Calculate and print the Overall Accuracy of the model on the test set.
* Generate and print a Classification Report (precision, recall, f1-score for each class) and a Confusion Matrix to provide a detailed breakdown of model performance for each of the 'Down', 'Sideways', and 'Up' classes.
* Calculate and print Class-specific Performance (precision and recall for each class).

**Trading Strategy Analysis (Interpretation):**

Analyze the model's performance in specific trading scenarios:

* Accuracy when predicting 'Up'.
* Accuracy when predicting 'Down'.
* Accuracy of high-confidence predictions (where the predicted probability for the chosen class is above a certain threshold, e.g., 60%).

**Feature Importance Analysis:**

* Extract and display the feature\_importances\_ from the trained model.
* Sort features by importance in descending order and print the top 10 most important features, which helps understand which factors contribute most to the predictions.

**Recent and Future Predictions:**

Recent Predictions: Predict the direction for the last few days (e.g., 5 days) of the test set and display the actual close, predicted direction, and class probabilities.

Last Day Prediction: Predict the direction for the very last day of the test set for a quick glance at the most recent prediction.

Future Predictions (Example):

* Determine the last\_date from the cleaned dataset.
* Generate a list of future\_dates (e.g., next 5 days).
* Use the features from the last available day in the test set as a proxy for future features (assuming similar market conditions).
* Predict the Predicted\_Direction and class Probabilities for these future dates based on the last known features. This is a simplistic approach and a more sophisticated model would typically use projected future features.

**PROGRAM**

import pandas as pd

import numpy as np

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from datetime import datetime, timedelta

dataset = pd.read\_csv("D:\\ML PROJECTS\\AML lab\\dataset\\apple\_stock.csv")

dataset.tail()

df = pd.read\_csv('dataset\HistoricalQuotes.csv')

df.info()

df.head(10)

df = pd.read\_csv('augmented\_financial\_data.csv')

print(f"Original dataset size: {len(df\_clean)}")

split\_index = int(len(df\_clean) \* 0.75)

model = DecisionTreeClassifier(

max\_depth=8,

min\_samples\_split=10,

min\_samples\_leaf=5,

max\_features='sqrt',

class\_weight='balanced',

random\_state=42

)

print(f"Optimized for {len(df\_clean)} samples")

print(f"Training: {split\_index} samples ({split\_index/len(df\_clean):.1%})")

print(f"Testing: {len(df\_clean)-split\_index} samples ({(len(df\_clean)-split\_index)/len(df\_clean):.1%})")

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

df = df.sort\_values('Date').reset\_index(drop=True)

print("Dataset shape:", df.shape)

print(df.head())

df['Next\_Day\_Close'] = df['Close'].shift(-1)

df['Price\_Change\_Pct'] = (df['Next\_Day\_Close'] - df['Close']) / df['Close']

threshold = 0.01

df['Price\_Direction'] = np.where(df['Price\_Change\_Pct'] > threshold, 2,

np.where(df['Price\_Change\_Pct'] < -threshold, 0, 1))

direction\_labels = {0: 'Down', 1: 'Sideways', 2: 'Up'}

df['Direction\_Label'] = df['Price\_Direction'].map(direction\_labels)

print("Multi-Class Price Direction Distribution:")

print(df['Price\_Direction'].value\_counts().sort\_index())

print("\nPercentage Distribution:")

print(df['Price\_Direction'].value\_counts(normalize=True).sort\_index())

df['Price\_Change'] = df['Close'].pct\_change()

df['Volume\_Change'] = df['Volume'].pct\_change()

df['SMA\_5'] = df['Close'].rolling(window=5).mean()

df['SMA\_10'] = df['Close'].rolling(window=10).mean()

df['Price\_Above\_SMA5'] = (df['Close'] > df['SMA\_5']).astype(int)

df['Price\_Above\_SMA10'] = (df['Close'] > df['SMA\_10']).astype(int)

df['Volatility'] = df['Price\_Change'].rolling(window=5).std()

df['High\_Low\_Ratio'] = df['High'] / df['Low']

df['Open\_Close\_Ratio'] = df['Open'] / df['Close']

df['Price\_Position'] = (df['Close'] - df['Low']) / (df['High'] - df['Low'])

df['Prev\_Day\_Up'] = (df['Close'] > df['Close'].shift(1)).astype(int)

df['Prev\_Day\_Volume'] = df['Volume'].shift(1)

df['Price\_Momentum\_3'] = df['Close'] / df['Close'].shift(3)

df['Volume\_Momentum\_3'] = df['Volume'] / df['Volume'].shift(3)

print("Features created successfully!")

features = [

'Open', 'High', 'Low', 'Volume',

'Price\_Change', 'Volume\_Change',

'Price\_Above\_SMA5', 'Price\_Above\_SMA10',

'Volatility', 'High\_Low\_Ratio', 'Open\_Close\_Ratio',

'Price\_Position', 'Prev\_Day\_Up',

'Price\_Momentum\_3', 'Volume\_Momentum\_3'

]

df\_clean = df.dropna()

print(f"Dataset shape after cleaning: {df\_clean.shape}")

X = df\_clean[features]

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

print(f"Final features shape: {X.shape}")

print(f"Multi-class target distribution after cleaning:")

print(y.value\_counts(normalize=True).sort\_index())

split\_index = int(len(df\_clean) \* 0.80)

X\_train = X[:split\_index]

X\_test = X[split\_index:]

y\_train = y[:split\_index]

y\_test = y[split\_index:]

print(f"Training set: {X\_train.shape[0]} samples (75%)")

print(f"Test set: {X\_test.shape[0]} samples (25%)")

model = DecisionTreeClassifier(

max\_depth=8,

min\_samples\_split=10,

min\_samples\_leaf=5,

max\_features='sqrt',

class\_weight='balanced',

random\_state=42

)

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

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

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

print("Multi-class model trained successfully!")

tscv = TimeSeriesSplit(n\_splits=5)

cv\_scores = []

for train\_idx, test\_idx in tscv.split(X):

X\_train\_cv = X.iloc[train\_idx]

X\_test\_cv = X.iloc[test\_idx]

y\_train\_cv = y.iloc[train\_idx]

y\_test\_cv = y.iloc[test\_idx]

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

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

cv\_scores.append(accuracy\_score(y\_test\_cv, y\_pred\_cv))

print(f"Cross-validation accuracy: {np.mean(cv\_scores):.4f} (±{np.std(cv\_scores):.4f})")

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

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

target\_names = ['Down', 'Sideways', 'Up']

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred, target\_names=target\_names))

print("\nConfusion Matrix:")

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

print(" Down Sideways Up")

for i, row in enumerate(cm):

print(f"{target\_names[i]:>6}: {row}")

print("\nClass-specific Performance:")

for i, class\_name in enumerate(target\_names):

class\_precision = cm[i, i] / cm[:, i].sum() if cm[:, i].sum() > 0 else 0

class\_recall = cm[i, i] / cm[i, :].sum() if cm[i, :].sum() > 0 else 0

print(f"{class\_name}: Precision={class\_precision:.4f}, Recall={class\_recall:.4f}")

print("\nTrading Strategy Analysis:")

up\_predictions = (y\_pred == 2)

if up\_predictions.sum() > 0:

up\_accuracy = (y\_test[up\_predictions] == 2).mean()

print(f"When predicting UP: Accuracy = {up\_accuracy:.4f}")

down\_predictions = (y\_pred == 0)

if down\_predictions.sum() > 0:

down\_accuracy = (y\_test[down\_predictions] == 0).mean()

print(f"When predicting DOWN: Accuracy = {down\_accuracy:.4f}")

max\_proba = y\_pred\_proba.max(axis=1)

high\_confidence = max\_proba > 0.6

if high\_confidence.sum() > 0:

high\_conf\_accuracy = (y\_test[high\_confidence] == y\_pred[high\_confidence]).mean()

print(f"High confidence predictions (>60%): Accuracy = {high\_conf\_accuracy:.4f}")

print(f"High confidence predictions: {high\_confidence.sum()}/{len(y\_test)} = {high\_confidence.mean():.2%}")

feature\_importance = pd.DataFrame({

'feature': features,

'importance': model.feature\_importances\_

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

print("\nTop 10 Most Important Features:")

print(feature\_importance.head(10))

last\_5\_days = X\_test.tail(5)

predictions = model.predict(last\_5\_days)

probabilities = model.predict\_proba(last\_5\_days)

results = pd.DataFrame({

'Date': df\_clean['Date'].iloc[-5:].values,

'Actual\_Close': df\_clean['Close'].iloc[-5:].values,

'Predicted\_Direction': [target\_names[p] for p in predictions],

'Prob\_Down': probabilities[:, 0],

'Prob\_Sideways': probabilities[:, 1],

'Prob\_Up': probabilities[:, 2],

'Actual\_Direction': [target\_names[a] for a in y\_test.tail(5)]

})

print("\nRecent Multi-Class Predictions:")

print(results.round(3))

last\_1\_day = X\_test.tail(1)

predictions = model.predict(last\_1\_day)

probabilities = model.predict\_proba(last\_1\_day)

results = pd.DataFrame({

'Date': df\_clean['Date'].iloc[-1:].values,

'Actual\_Close': df\_clean['Close'].iloc[-1:].values,

'Predicted\_Direction': [target\_names[p] for p in predictions],

'Prob\_Down': probabilities[:, 0],

'Prob\_Sideways': probabilities[:, 1],

'Prob\_Up': probabilities[:, 2],

'Actual\_Direction': [target\_names[a] for a in y\_test.tail(1)]

})

print("\nRecent Multi-Class Prediction:")

print(results.round(3))

last\_date = df\_clean['Date'].iloc[-1]

print(f"Last date in dataset: {last\_date}")

future\_dates = []

for i in range(1, 6):

future\_date = last\_date + timedelta(days=i)

future\_dates.append(future\_date)

last\_features = X\_test.iloc[-1:].values

predictions = model.predict([last\_features[0]] \* 5)

probabilities = model.predict\_proba([last\_features[0]] \* 5)

results = pd.DataFrame({

'Date': future\_dates,

'Predicted\_Direction': [target\_names[p] for p in predictions],

'Prob\_Down': probabilities[:, 0],

'Prob\_Sideways': probabilities[:, 1],

'Prob\_Up': probabilities[:, 2]

})

print("\nFuture Predictions:")

print(results.round(3))

**INPUT**

Original dataset size: 349

Optimized for 349 samples

Training: 261 samples (74.8%)

Testing: 88 samples (25.2%)

Dataset shape: (2000, 10)

Date Adj Close Close High Low \

0 2023-11-02 00:00:00+00:00 169.532969 169.284010 173.358886 168.848136

1 2023-11-02 00:00:00+00:00 193.327099 194.611325 196.129159 194.381548

2 2023-11-02 00:00:00+00:00 223.062965 222.666488 224.237492 219.466940

3 2023-11-02 00:00:00+00:00 214.021669 214.284820 214.523636 211.181214

4 2023-11-02 00:00:00+00:00 227.002819 226.579384 227.730680 223.185310

Open Volume returns ma\_5 ma\_20

0 169.234626 51300713 -0.006047 NaN NaN

1 195.375432 53422579 -0.003484 NaN NaN

2 222.758328 37501765 0.000494 221.454004 224.735001

3 211.889645 58734942 -0.012317 NaN NaN

4 225.545858 33102053 0.002520 NaN NaN

**OUTPUT**

Overall Accuracy: 0.6143

Classification Report:

precision recall f1-score support

Down 0.74 0.66 0.70 35

Sideways 0.00 0.00 0.00 2

Up 0.74 0.61 0.67 33

accuracy 0.61 70

macro avg 0.49 0.42 0.45 70

weighted avg 0.72 0.61 0.66 70

Confusion Matrix:

Down Sideways Up

Down: [23 6 6]

Sideways: [1 0 1]

Up: [ 7 6 20]

Class-specific Performance:

Down: Precision=0.7419, Recall=0.6571

Sideways: Precision=0.0000, Recall=0.0000

Up: Precision=0.7407, Recall=0.6061

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

Trading Strategy Analysis:

When predicting UP: Accuracy = 0.7407

When predicting DOWN: Accuracy = 0.7419

High confidence predictions (>60%): Accuracy = 0.7069

High confidence predictions: 58/70 = 82.86%

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

Recent Multi-Class Predictions:

Date Actual\_Close Predicted\_Direction Prob\_Down Prob\_Sideways \

0 2024-10-27 227.595 Sideways 0.145 0.855

1 2024-10-28 214.331 Down 0.618 0.000

2 2024-10-29 192.856 Down 0.853 0.000

3 2024-10-31 183.769 Up 0.000 0.000

4 2024-10-31 221.044 Down 0.765 0.174

Prob\_Up Actual\_Direction

0 0.000 Down

1 0.382 Down

2 0.147 Up

3 1.000 Up

4 0.061 Down

**RESULT**

Last date in dataset: 2024-10-31 00:00:00+00:00

Future Predictions:

Date Predicted\_Direction Prob\_Down Prob\_Sideways \

0 2024-11-01 00:00:00+00:00 Down 0.765 0.174

1 2024-11-02 00:00:00+00:00 Down 0.765 0.174

2 2024-11-03 00:00:00+00:00 Down 0.765 0.174

3 2024-11-04 00:00:00+00:00 Down 0.765 0.174

4 2024-11-05 00:00:00+00:00 Down 0.765 0.174

Prob\_Up

0 0.061

1 0.061

2 0.061

3 0.061

4 0.061

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

**AIM**