# Ensemble Techniques

**Instructions:**

Please share your answers filled in-line in the word document. Submit code separately wherever applicable.

Please ensure you update all the details:

**Name: Vishvash C Batch ID:** 23012024

**Topic: Ensemble Techniques**

**Guidelines:**

**1. An assignment submission is considered complete only when the correct and executable code(s) and documentation explaining the method and results are submitted. Failing to submit either of those will be considered an invalid submission and not a correct submission.**

**2. Ensure that you submit your assignments correctly and in full. Resubmission is not allowed.**

**3. Post the submission you can evaluate your work by referring to the keys provided. (will be available only post the submission).**

**Hints:**

1. **Business Problem**
   1. **What is the business objective?**
   2. **Are there any constraints?**
2. **Work on each feature of the dataset to create a data dictionary as displayed in the below image:**

**Make a table as shown above and provide information about the features such as its data type and its relevance to the model building. And if not relevant, provide reasons and a description of the feature.**

**Using Python codes perform:**

1. **Data Pre-processing**

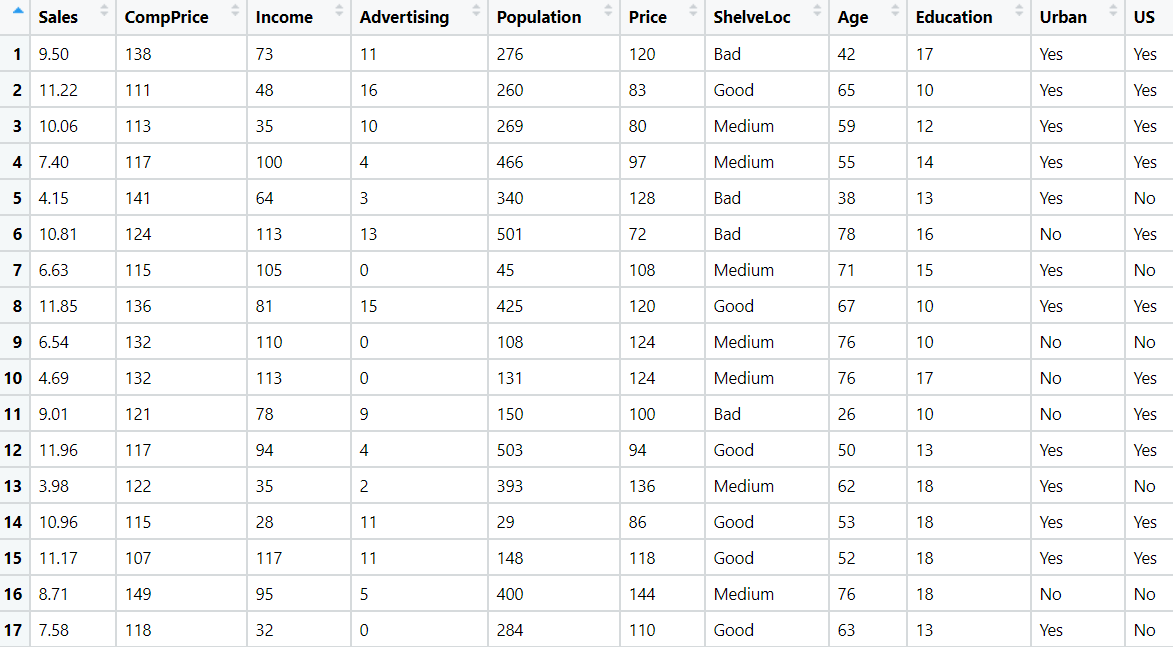
**3.1 Data Cleaning, Feature Engineering, etc.**

**3.2 Outlier Treatment.**

1. **Exploratory Data Analysis (EDA):**
   1. **Summary.**
   2. **Univariate analysis.**
   3. **Bivariate analysis.**
2. **Model Building**
   1. **Build the model on the cleaned data.**
   2. **Experiment with Ensemble models to improve the reliability of Machine Learning model output**
   3. **Perform Bagging Boosting (adaboost, gradient boost, Xgboost), Stacking, and Voting on the given datasets in Hands-on Material.**
   4. **Train and Test the model and compare performances by capturing various metrics for experiments conducted with different hyperparameters. Also, use Hyperparameter optimization techniques (ex: GridSearchCV) to capture the best set of parameters for an improved model.**
   5. **Briefly explain the model output in the documentation.**
3. **Write about the benefits/impact of the solution - in what way does the business (client) benefit from the solution provided.**
4. **Deploy the final best solution for your client with UI.**

**Problem Statements:**

1. A cloth manufacturing company is interested to know about the different attributes contributing to high sales. Build a decision tree & random forest model with Sales as the target variable (first convert it into categorical variable).



|  |  |  |  |
| --- | --- | --- | --- |
| Name of Feature | Description | Type | Relevance |
| Sales | Total sales of the cloth manufacturing company | Quantitative | Relevant |
| CompPrice | Price charged by competitor for similar product | Quantitative | Relevant |
| Income | Average income of the consumers in the area | Quantitative | Relevant |
| Advertising | Advertising budget for promoting the product | Quantitative | Relevant |
| Population | Population size of the area | Quantitative | Relevant |
| Price | Price of the product | Quantitative | Relevant |
| ShelveLoc | Shelf location of the product in stores (e.g., Bad, Good, Medium) | Nominal | Relevant |
| Age | Average age of consumers in the area | Quantitative | Relevant |
| Education | Average education level in the area | Quantitative | Relevant |
| Urban | Whether the store is located in an urban area or not | Nominal | Relevant |
| US | Whether the store is in the US or not | Nominal | Relevant |

**Code:**

'''

1.a. Business problem: A cloth manufacturing company is interested to know about the different attributes contributing to high sales.

i. Business Objectives: Maximize the sales

ii. Business Constraints: Minimize the advertising

Success Criteria:

i. Business success criteria: Increase the efficient of marketing by 10%

ii. ML success criteria: Achieve an accuracy of at least 70%

iii. Economic success criteria: Increase the sales by at least by 20%

1.b. Data Collection: Bank -> 400 sales data, 11 variables (10 Inputs and 1 Ouput)

2. Data Preprocessing - Cleansing & EDA / Descriptive Analytics

3. Model Building - Experiment with different models alongside Hyperparameter tuning

4. Evaluation - Not just model evaluation based on accuracy but we also need

to evaluate business & economic success criteria

5. Model Deployment (Flask)

6. Monitoring & Maintenance (Prediction results to the database - MySQL / MS SQL)

'''

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

#import seaborn as sns

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import MinMaxScaler

from sklearn.preprocessing import OneHotEncoder

from feature\_engine.outliers import Winsorizer

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

import joblib

import pickle

from sklearn.svm import LinearSVC

from sklearn.linear\_model import LogisticRegression

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier as DT

from sklearn import datasets, linear\_model, neighbors, ensemble #, svm, naive\_bayes

from sklearn.ensemble import VotingClassifier

from sklearn.ensemble import StackingClassifier

from sklearn.ensemble import BaggingClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import GradientBoostingClassifier

# pip install xgboost

import xgboost as xgb

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import cross\_validate

# Hyperparameter optimization

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import GridSearchCV

import sklearn.metrics as skmet

data = pd.read\_csv(r"C:/Users/Lenovo/Downloads/Study material/Data Science/Ensemble\_Techniques/Assignment/Ensemble Models/ClothCompany\_Data (1).csv")

from sqlalchemy import create\_engine, text

from urllib.parse import quote

# creating engine to connect MySQL database

user = 'root' # user name

pw = quote('1234') # password

db = 'sales\_db' # database

engine = create\_engine(f"mysql+pymysql://{user}:{pw}@localhost/{db}")

data.to\_sql('sales', con = engine, if\_exists = 'replace', chunksize = 1000, index = False )

sql = "SELECT \* FROM sales;"

# for sqlalchmey 1.4.x version

# df = pd.read\_sql\_query(sql, engine)

# for sqlalchmey 2.x version

df = pd.read\_sql\_query(text(sql), engine.connect())

# df = pd.read\_csv(r"movies\_classification.csv")

df.head()

df.info()

# Define the numerical ranges and corresponding labels for the categories

sales\_bins = [0, 5, 10, float('inf')]

sales\_labels = [ 'Low', 'Medium', 'High']

# Convert numerical "Sales" df to categorical using cut function

df['Sales\_Category'] = pd.cut(df['Sales'], bins=sales\_bins, labels=sales\_labels, right=False)

# Drop the original "Sales" column if needed

df.drop(columns=['Sales'], inplace=True)

# AutoEDA

# D-Tale

#########

# pip install dtale

import dtale

d = dtale.show(df)

d.open\_browser()

# or

# AutoEDA

# Sweetviz

import sweetviz

my\_report = sweetviz.analyze(df)

my\_report.show\_html('Report1.html')

#########

df['Sales\_Category'].unique()

df['Sales\_Category'].value\_counts()

# Data split into Input and Output

X = df.iloc[:, :10] # Predictors

y = df['Sales\_Category'] # Target

df.info()

# #### Separating Numeric and Non-Numeric columns

numeric\_features = X.select\_dtypes(exclude = ['object']).columns

numeric\_features

categorical\_features = X.select\_dtypes(include=['object']).columns

categorical\_features

# ### Data Preprocessing

# Numeric\_features

# ### Imputation to handle missing values

# ### MinMaxScaler to convert the magnitude of the columns to a range of 0 to 1

num\_pipeline = Pipeline(steps = [('impute', SimpleImputer(strategy = 'mean')), ('scale', MinMaxScaler())])

# ### Encoding - One Hot Encoder to convert Categorical data to Numeric values

# Categorical features

encoding\_pipeline = Pipeline([('onehot', OneHotEncoder(drop = 'first'))])

# Creating a transformation of variable with ColumnTransformer()

preprocessor = ColumnTransformer(transformers = [('num', num\_pipeline, numeric\_features), ('categorical', encoding\_pipeline, categorical\_features)])

imp\_enc\_scale = preprocessor.fit(X)

# #### Save the pipeline model using joblib

joblib.dump(imp\_enc\_scale, 'imp\_enc\_scale')

import os

os.getcwd()

cleandata = pd.DataFrame(imp\_enc\_scale.transform(X), columns = imp\_enc\_scale.get\_feature\_names\_out())

cleandata

# Note: If you get any error then update the scikit-learn library version & restart the kernel to fix it

# ### Outlier Analysis

# Multiple boxplots in a single visualization.

# Columns with larger scales affect other columns.

# Below code ensures each column gets its own y-axis.

# pandas plot() function with parameters kind = 'box' and subplots = True

cleandata.iloc[:, 0:7].plot(kind = 'box', subplots = True, sharey = False, figsize = (10, 6))

'''sharey True or 'all': x- or y-axis will be shared among all subplots.

False or 'none': each subplot x- or y-axis will be independent.'''

# Increase spacing between subplots

plt.subplots\_adjust(wspace = 0.75) # ws is the width of the padding between subplots, as a fraction of the average Axes width.

plt.show()

cleandata.iloc[:, 0:7].columns

# #### Outlier analysis: Columns 'months\_loan\_duration', 'amount', and 'age' are continuous, hence outliers are treated

winsor = Winsorizer(capping\_method = 'iqr', # choose IQR rule boundaries or gaussian for mean and std

tail = 'both', # cap left, right or both tails

fold = 1.5,

variables = list(cleandata.iloc[:, 0:7].columns))

outlier = winsor.fit(cleandata.iloc[:, 0:7])

# Save the winsorizer model

joblib.dump(outlier, 'winsor')

cleandata.iloc[:, 0:7] = outlier.transform(cleandata.iloc[:, 0:7])

# Clean data

cleandata

# Verify for outliers

cleandata.iloc[:, 0:7].plot(kind = 'box', subplots = True, sharey = False, figsize = (10, 6))

# increase spacing between subplots

plt.subplots\_adjust(wspace = 0.75) # ws is the width of the padding between subplots, as a fraction of the average Axes width.

plt.show()

clean\_data = cleandata

target = y

# Splitting data into training and testing data set

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(clean\_data, target, test\_size = 0.2,

stratify = target, random\_state = 0)

### Decision Tree Model

model = DT(criterion = 'entropy')

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

# Prediction on Test Data

preds = model.predict(X\_test)

preds

# Accuracy

print(accuracy\_score(Y\_test, preds))

pd.crosstab(Y\_test, preds, rownames = ['Actual'], colnames = ['Predictions'])

### Hyperparameter Optimization

# create a dictionary of all hyperparameters to be experimented

param\_grid = { 'criterion':['gini', 'entropy'], 'max\_depth': np.arange(3, 15)}

# Decision tree model

dtree\_model = DT()

# GridsearchCV with cross-validation to perform experiments with parameters set

dtree\_gscv = GridSearchCV(dtree\_model, param\_grid, cv = 5, scoring = 'accuracy',

return\_train\_score = False, verbose = 1)

# Train the model with Grid search optimization technique

dtree\_gscv.fit(X\_train, Y\_train)

# The best set of parameter values

dtree\_gscv.best\_params\_

# Model with best parameter values

DT\_best = dtree\_gscv.best\_estimator\_

DT\_best

# Prediction on Test Data

preds1 = DT\_best.predict(X\_test)

preds1

# Model evaluation

# Cross Table (Confusion Matrix)

pd.crosstab(Y\_test, preds, rownames = ['Actual'], colnames= ['Predictions'])

# Accuracy

print(accuracy\_score(Y\_test, preds))

# ### Save the Best Model with pickel library

pickle.dump(DT\_best, open('DT.pkl', 'wb'))

# Base Model 1

### k-Nearest Neighbors (k-NN) with GridSearchCV

knn = neighbors.KNeighborsClassifier()

params\_knn = {'n\_neighbors': np.arange(1, 25)}

knn\_gs = GridSearchCV(knn, params\_knn, cv = 5)

knn\_gs.fit(X\_train, Y\_train)

knn\_gs.best\_params\_

knn\_best = knn\_gs.best\_estimator\_

# Base Model 2

### Random Forest Classifier with GridSearchCV

rf = ensemble.RandomForestClassifier(random\_state = 0)

params\_rf = {'n\_estimators': [50, 100, 200]}

rf\_gs = GridSearchCV(rf, params\_rf, cv = 5)

rf\_gs.fit(X\_train, Y\_train)

rf\_gs.best\_params\_

rf\_best = rf\_gs.best\_estimator\_

# Base Model 3

### Logistic Regression with GridSearchCV

log\_reg = linear\_model.LogisticRegression()

C = np.logspace(1, 4, 10)

params\_lr = dict(C = C)

lr\_gs = GridSearchCV(log\_reg, params\_lr, cv = 5)

lr\_gs.fit(X\_train, Y\_train)

lr\_gs.best\_estimator\_

lr\_best = lr\_gs.best\_estimator\_

# Combine all three Based models

estimators = [('knn', knn\_best), ('rf', rf\_best), ('log\_reg', lr\_best)]

# Hard/Majority Voting

# # VotingClassifier with voting = "hard" parameter

ensemble\_H = VotingClassifier(estimators, voting = "hard")

# Fit classifier with the training data

hard\_voting = ensemble\_H.fit(X\_train, Y\_train)

# Save the voting classifier

pickle.dump(hard\_voting, open('hard\_voting.pkl', 'wb'))

# Loading a saved model

model = pickle.load(open('hard\_voting.pkl', 'rb'))

print("knn\_gs.score: ", knn\_best.score(X\_test, Y\_test))

# Output: knn\_gs.score:

print("rf\_gs.score: ", rf\_best.score(X\_test, Y\_test))

# Output: rf\_gs.score:

print("log\_reg.score: ", lr\_best.score(X\_test, Y\_test))

# Output: log\_reg.score:

# Hard Voting Ensembler

print("Hard Voting Ensemble Score: ", ensemble\_H.score(X\_test, Y\_test))

# Output: ensemble.score:

#############################################################

# Soft Voting

# VotingClassifier with voting = "soft" parameter

ensemble\_S = VotingClassifier(estimators, voting = "soft")

soft\_voting = ensemble\_S.fit(X\_train, Y\_train)

# Save model

pickle.dump(soft\_voting, open('soft\_voting.pkl', 'wb'))

# Load the saved model

model = pickle.load(open('soft\_voting.pkl', 'rb'))

print("knn\_gs.score: ", knn\_gs.score(X\_test, Y\_test))

# Output: knn\_gs.score:

print("rf\_gs.score: ", rf\_gs.score(X\_test, Y\_test))

# Output: rf\_gs.score:

print("log\_reg.score: ", lr\_gs.score(X\_test, Y\_test))

# Output: log\_reg.score:

print("Soft Voting Ensemble Score: ", ensemble\_S.score(X\_test, Y\_test))

# Output: ensemble.score:

# Base estimators

estimators = [('rf', RandomForestClassifier(n\_estimators = 10, random\_state = 42)),

('svc', LinearSVC(random\_state = 42))]

# Meta Model stacked on top of base estimators

clf = StackingClassifier(estimators = estimators, final\_estimator = LogisticRegression())

# Fit the model on traing data

stacking = clf.fit(X\_train, Y\_train)

# Accuracy

stacking.score(X\_test, Y\_test)

# Save the Stacking model

pickle.dump(stacking, open('stacking\_cloth.pkl', 'wb'))

# Load the saved model

model = pickle.load(open('stacking\_cloth.pkl', 'rb'))

model

# # Bagging Classifier Model

# from sklearn.ensemble import BaggingClassifier

# decision tree defined first

# from sklearn import tree

# from sklearn.metrics import confusion\_matrix

# from sklearn.metrics import accuracy\_score

clftree = tree.DecisionTreeClassifier()

bag\_clf = BaggingClassifier(estimator = clftree, n\_estimators = 500,

bootstrap = True, n\_jobs = -1, random\_state = 42)

# Fit the model

bagging = bag\_clf.fit(X\_train, Y\_train)

print(confusion\_matrix(Y\_train, bagging.predict(X\_train)))

print(accuracy\_score(Y\_train, bagging.predict(X\_train)))

print('\n')

print(confusion\_matrix(Y\_test, bagging.predict(X\_test)))

print(accuracy\_score(Y\_test, bagging.predict(X\_test)))

# Saving the best model

pickle.dump(bagging, open('baggingmodel.pkl', 'wb'))

# ## Cross Validation implementation

# from sklearn.model\_selection import cross\_validate

def cross\_validation(model, \_X, \_y, \_cv=5):

\_scoring = ['accuracy', 'precision', 'recall', 'f1']

results = cross\_validate(estimator=model,

X=\_X,

y=\_y,

cv=\_cv,

scoring=\_scoring,

return\_train\_score=True)

return pd.DataFrame({"Training Accuracy scores": results['train\_accuracy'],

"Mean Training Accuracy": results['train\_accuracy'].mean()\*100,

"Training Precision scores": results['train\_precision'],

"Mean Training Precision": results['train\_precision'].mean(),

"Training Recall scores": results['train\_recall'],

"Mean Training Recall": results['train\_recall'].mean(),

"Training F1 scores": results['train\_f1'],

"Mean Training F1 Score": results['train\_f1'].mean(),

"Validation Accuracy scores": results['test\_accuracy'],

"Mean Validation Accuracy": results['test\_accuracy'].mean()\*100,

"Validation Precision scores": results['test\_precision'],

"Mean Validation Precision": results['test\_precision'].mean(),

"Validation Recall scores": results['test\_recall'],

"Mean Validation Recall": results['test\_recall'].mean(),

"Validation F1 scores": results['test\_f1'],

"Mean Validation F1 Score": results['test\_f1'].mean()

})

# Call the above custom function

Bagging\_cv\_scores = cross\_validation(bag\_clf, X\_train, Y\_train, 5)

Bagging\_cv\_scores

# Visualization custom function

def plot\_result(x\_label, y\_label, plot\_title, train\_data, val\_data):

# Set size of plot

plt.figure(figsize=(12,6))

labels = ["1st Fold", "2nd Fold", "3rd Fold", "4th Fold", "5th Fold"]

X\_axis = np.arange(len(labels))

# ax = plt.gca()

plt.ylim(0.40000, 1)

plt.bar(X\_axis-0.2, train\_data, 0.4, color='blue', label='Training')

plt.bar(X\_axis+0.2, val\_data, 0.4, color='red', label='Validation')

plt.title(plot\_title, fontsize=30)

plt.xticks(X\_axis, labels)

plt.xlabel(x\_label, fontsize=14)

plt.ylabel(y\_label, fontsize=14)

plt.legend()

plt.grid(True)

plt.show()

# get\_ipython().run\_line\_magic('matplotlib', 'inline')

model\_name = "Bagging Classifier"

plot\_result(model\_name,

"Accuracy",

"Accuracy scores in 5 Folds",

Bagging\_cv\_scores["Training Accuracy scores"],

Bagging\_cv\_scores["Validation Accuracy scores"])

# ## Random Forest Model

# from sklearn.ensemble import RandomForestClassifier

rf\_Model = RandomForestClassifier()

# #### Hyperparameters

# Number of trees in random forest

n\_estimators = [int(x) for x in np.linspace(start = 10, stop = 80, num = 10)]

# Number of features to consider at every split

max\_features = ['log2', 'sqrt']

# Maximum number of levels in tree

max\_depth = [2, 4]

# Minimum number of samples required to split a node

min\_samples\_split = [2, 5]

# Minimum number of samples required at each leaf node

min\_samples\_leaf = [1, 2]

# Method of selecting samples for training each tree

bootstrap = [True, False]

# Create the param grid

param\_grid = {'n\_estimators': n\_estimators,

'max\_features': max\_features,

'max\_depth': max\_depth,

'min\_samples\_split': min\_samples\_split,

'min\_samples\_leaf': min\_samples\_leaf,

'bootstrap': bootstrap}

print(param\_grid)

# ### Hyperparameter optimization with GridSearchCV

rf\_Grid = GridSearchCV(estimator = rf\_Model, param\_grid = param\_grid, cv = 10, verbose = 1, n\_jobs = -1)

rf\_Grid.fit(X\_train, Y\_train)

rf\_Grid.best\_params\_

cv\_rf\_grid = rf\_Grid.best\_estimator\_

# ## Check Accuracy

# Evaluation on Test Data

test\_pred = cv\_rf\_grid.predict(X\_test)

accuracY\_test = np.mean(test\_pred == Y\_test)

accuracY\_test

cm = skmet.confusion\_matrix(Y\_test, test\_pred)

cmplot = skmet.ConfusionMatrixDisplay(confusion\_matrix = cm, display\_labels = ['Low', 'Medium', 'High'])

cmplot.plot()

cmplot.ax\_.set(title = 'Sales Detection Confusion Matrix',

xlabel = 'Predicted Value', ylabel = 'Actual Value')

print (f'Train Accuracy - : {rf\_Grid.score(X\_train, Y\_train):.3f}')

print (f'Test Accuracy - : {rf\_Grid.score(X\_test, Y\_test):.3f}')

# RandomizedSearchCV

# ### Hyperparameter optimization with RandomizedSearchCV

rf\_Random = RandomizedSearchCV(estimator = rf\_Model, param\_distributions = param\_grid, cv = 10, verbose = 2, n\_jobs = -1)

rf\_Random.fit(X\_train, Y\_train)

rf\_Random.best\_params\_

cv\_rf\_random = rf\_Random.best\_estimator\_

# Evaluation on Test Data

test\_pred\_random = cv\_rf\_random.predict(X\_test)

accuracY\_test\_random = np.mean(test\_pred\_random == Y\_test)

accuracY\_test\_random

print (f'Train Accuracy - : {rf\_Random.score(X\_train, Y\_train):.3f}')

print (f'Test Accuracy - : {rf\_Random.score(X\_test, Y\_test):.3f}')

# ## Save the best model from Randomsearch CV approach

pickle.dump(cv\_rf\_random, open('rfc.pkl', 'wb'))

# ## Cross Validation implementation

# from sklearn.model\_selection import cross\_validate

def cross\_validation(model, \_X, \_y, \_cv = 5):

\_scoring = ['accuracy', 'precision', 'recall', 'f1']

results = cross\_validate(estimator=model,

X = \_X,

y = \_y,

cv = \_cv,

scoring = \_scoring,

return\_train\_score = True)

return pd.DataFrame({"Training Accuracy scores": results['train\_accuracy'],

"Mean Training Accuracy": results['train\_accuracy'].mean()\*100,

"Training Precision scores": results['train\_precision'],

"Mean Training Precision": results['train\_precision'].mean(),

"Training Recall scores": results['train\_recall'],

"Mean Training Recall": results['train\_recall'].mean(),

"Training F1 scores": results['train\_f1'],

"Mean Training F1 Score": results['train\_f1'].mean(),

"Validation Accuracy scores": results['test\_accuracy'],

"Mean Validation Accuracy": results['test\_accuracy'].mean()\*100,

"Validation Precision scores": results['test\_precision'],

"Mean Validation Precision": results['test\_precision'].mean(),

"Validation Recall scores": results['test\_recall'],

"Mean Validation Recall": results['test\_recall'].mean(),

"Validation F1 scores": results['test\_f1'],

"Mean Validation F1 Score": results['test\_f1'].mean()

})

Random\_forest\_result = cross\_validation(cv\_rf\_random, X\_train, Y\_train, 5)

Random\_forest\_result

def plot\_result(x\_label, y\_label, plot\_title, train\_data, val\_data):

# Set size of plot

plt.figure(figsize=(12, 6))

labels = ["1st Fold", "2nd Fold", "3rd Fold", "4th Fold", "5th Fold"]

X\_axis = np.arange(len(labels))

plt.ylim(0.40000, 1)

plt.bar(X\_axis - 0.2, train\_data, 0.1, color = 'blue', label = 'Training')

plt.bar(X\_axis + 0.2, val\_data, 0.1, color = 'red', label = 'Validation')

plt.title(plot\_title, fontsize=30)

plt.xticks(X\_axis, labels)

plt.xlabel(x\_label, fontsize=14)

plt.ylabel(y\_label, fontsize=14)

plt.legend()

plt.grid(True)

plt.show()

model\_name = "RandomForestClassifier"

plot\_result(model\_name,

"Accuracy",

"Accuracy scores in 5 Folds",

Random\_forest\_result["Training Accuracy scores"],

Random\_forest\_result["Validation Accuracy scores"])

# # AdaBoosting

# from sklearn.ensemble import AdaBoostClassifier

ada\_clf = AdaBoostClassifier(learning\_rate = 0.02, n\_estimators = 5000)

ada\_clf1 = ada\_clf.fit(X\_train, Y\_train)

predictions = ada\_clf1.predict(X\_test)

# Evaluation on Testing Data

confusion\_matrix(Y\_test, predictions)

accuracy\_score(Y\_test, predictions)

# Evaluation on Training Data

accuracy\_score(Y\_train, ada\_clf1.predict(X\_train))

# Saving the best model

pickle.dump(ada\_clf1, open('adaboost.pkl','wb'))

## GradientBoosting

# from sklearn.ensemble import GradientBoostingClassifier

boost\_clf = GradientBoostingClassifier()

boost\_clf1 = boost\_clf.fit(X\_train, Y\_train)

grad\_pred = boost\_clf1.predict(X\_test)

print(confusion\_matrix(Y\_test, grad\_pred))

print(accuracy\_score(Y\_test, grad\_pred))

print(confusion\_matrix(Y\_train, boost\_clf1.predict(X\_train)))

print(accuracy\_score(Y\_train, boost\_clf1.predict(X\_train)))

# https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html#sklearn.ensemble.GradientBoostingClassifier

# Hyperparameters

boost\_clf2 = GradientBoostingClassifier(learning\_rate = 0.02, n\_estimators = 1000, max\_depth = 1)

boost\_clf\_p = boost\_clf2.fit(X\_train, Y\_train)

grad\_pred\_p = boost\_clf\_p.predict(X\_test)

# Evaluation on Testing Data

print(confusion\_matrix(Y\_test, grad\_pred\_p))

print('\n')

print(accuracy\_score(Y\_test,grad\_pred\_p))

# Evaluation on Training Data

print(confusion\_matrix(Y\_train, boost\_clf\_p.predict(X\_train)))

accuracy\_score(Y\_train, boost\_clf\_p.predict(X\_train))

# Save the ML model

pickle.dump(boost\_clf\_p, open('gradiantboostparam.pkl', 'wb'))

grad\_model\_p = pickle.load(open('gradiantboostparam.pkl', 'rb'))

## XGBoosting

# pip install xgboost

# import xgboost as xgb

xgb\_clf = xgb.XGBClassifier(max\_depth = 5, n\_estimators = 10000,

learning\_rate = 0.3, n\_jobs = -1)

# n\_jobs – Number of parallel threads used to run xgboost.

# learning\_rate (float) – Boosting learning rate (xgb’s “eta”)

from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder

label\_encoder = LabelEncoder()

# Encode the string classes into numerical labels

Y\_train\_encoded = label\_encoder.fit\_transform(Y\_train)

Y\_test\_encoded = label\_encoder.fit\_transform(Y\_test)

xgb\_clf1 = xgb\_clf.fit(X\_train, Y\_train\_encoded)

xgb\_pred = xgb\_clf1.predict(X\_test)

# Evaluation on Testing Data

print(confusion\_matrix(Y\_test\_encoded, xgb\_pred))

accuracy\_score(Y\_test\_encoded, xgb\_pred)

# Feature Importance Plot

# "F score", also known as the "feature importance score".

# F-score of a feature is calculated based on how often the feature is used to

# split the data across all trees in the model.

# In simple terms, it reflects how useful or relevant a feature is for the

# model's decision-making process.

xgb.plot\_importance(xgb\_clf)

fi = pd.DataFrame(xgb\_clf1.feature\_importances\_.reshape(1, -1), columns = X\_train.columns)

fi

# Save the ML model

pickle.dump(xgb\_clf1, open('xgb.pkl', 'wb'))

xgb\_model = pickle.load(open('xgb.pkl', 'rb'))

## RandomizedSearchCV for XGB

xgb\_clf = xgb.XGBClassifier(n\_estimators = 500, learning\_rate = 0.1, random\_state = 42)

# Grid Search

param\_test1 = {'max\_depth': range(3,10,2), 'gamma': [0.1, 0.2, 0.3],

'subsample': [0.8, 0.9], 'colsample\_bytree': [0.8, 0.9],}

xgb\_RandomGrid = RandomizedSearchCV(estimator = xgb\_clf,

param\_distributions = param\_test1,

cv = 5, verbose = 2, n\_jobs = -1)

Randomized\_search1 = xgb\_RandomGrid.fit(X\_train, Y\_train\_encoded)

cv\_xg\_clf = Randomized\_search1.best\_estimator\_

cv\_xg\_clf

randomized\_pred = Randomized\_search1.predict(X\_test)

# Evaluation on Testing Data with model with hyperparameter

accuracy\_score(Y\_test\_encoded, randomized\_pred)

Randomized\_search1.best\_params\_

randomized\_pred\_1 = Randomized\_search1.predict(X\_train)

# Evaluation on Training Data with model with hyperparameters

accuracy\_score(Y\_train\_encoded, randomized\_pred\_1)

pickle.dump(cv\_xg\_clf, open('Randomizedsearch\_xgb.pkl', 'wb'))

**Output:**

data['Sales\_Category'].value\_counts()

Out[30]:

Sales\_Category

Medium 244

High 79

Low 77

Name: count, dtype: int64

data.info()

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

RangeIndex: 400 entries, 0 to 399

Data columns (total 11 columns):

# Column Non-Null Count Dtype

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

0 CompPrice 400 non-null int64

1 Income 400 non-null int64

2 Advertising 400 non-null int64

3 Population 400 non-null int64

4 Price 400 non-null int64

5 ShelveLoc 400 non-null object

6 Age 400 non-null int64

7 Education 400 non-null int64

8 Urban 400 non-null object

9 US 400 non-null object

10 Sales\_Category 400 non-null category

dtypes: category(1), int64(7), object(3)

memory usage: 31.9+ KB

cleandata = pd.DataFrame(imp\_enc\_scale.transform(X),

columns = imp\_enc\_scale.get\_feature\_names\_out())

cleandata

Out[61]:

num\_\_CompPrice num\_\_Income ... categorical\_\_Urban\_Yes categorical\_\_US\_Yes

0 0.622449 0.525253 ... 1.0 1.0

1 0.346939 0.272727 ... 1.0 1.0

2 0.367347 0.141414 ... 1.0 1.0

3 0.408163 0.797980 ... 1.0 1.0

4 0.653061 0.434343 ... 1.0 0.0

.. ... ... ... ... ...

395 0.622449 0.878788 ... 1.0 1.0

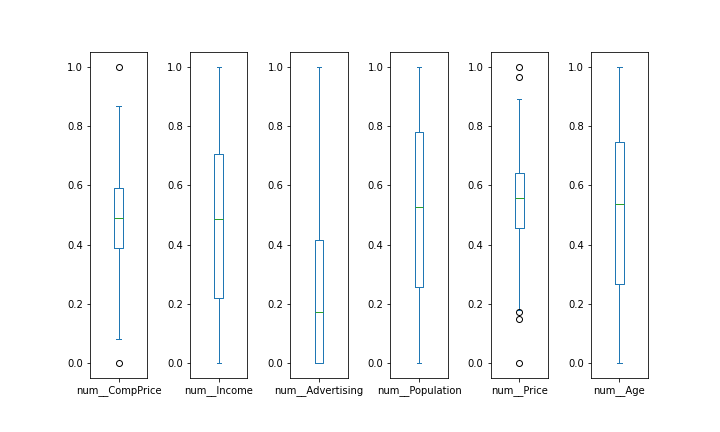
396 0.632653 0.020202 ... 0.0 1.0

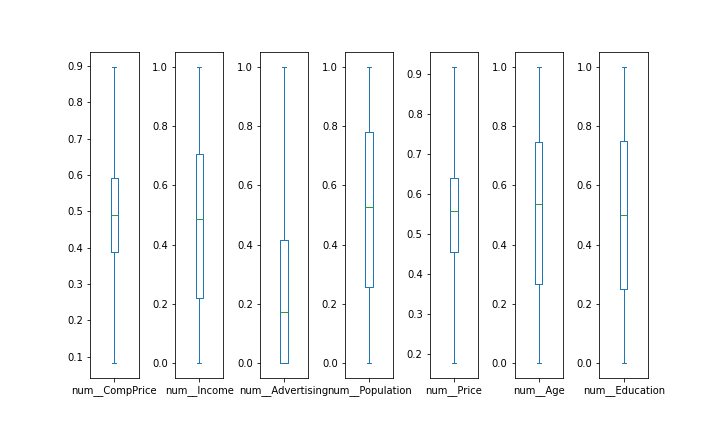
397 0.867347 0.050505 ... 1.0 1.0

398 0.234694 0.585859 ... 1.0 1.0

399 0.581633 0.161616 ... 1.0 1.0

[400 rows x 11 columns]





pd.crosstab(Y\_test, preds, rownames = ['Actual'], colnames= ['Predictions'])

Out[332]:

Predictions High Low Medium

Actual

Low 0 11 4

Medium 6 4 39

High 12 0 4

# Accuracy

print(accuracy\_score(Y\_test, preds))

0.775

knn\_gs.score: 0.5875

rf\_gs.score: 0.6125

log\_reg.score: 0.8125

Hard Voting Ensemble Score: 0.675

print("Soft Voting Ensemble Score: ", ensemble\_S.score(X\_test, Y\_test))

knn\_gs.score: 0.5875

rf\_gs.score: 0.6125

log\_reg.score: 0.8125

Soft Voting Ensemble Score: 0.675

#Bagging

Train Accuracy - : 0.744

Test Accuracy - : 0.625

#Random Forest

Train Accuracy - : 0.769

Test Accuracy - : 0.613

#Adaboosting

[[ 6 0 10]

[ 0 5 10]

[ 8 4 37]]

0.6

[[ 63 0 0]

[ 0 61 1]

[ 0 0 195]]

0.996875

#GradientBoosting

0.55

[[ 46 0 17]

[ 0 39 23]

[ 5 2 188]]

Out[358]: 0.853125

#XG Boosting

[[ 7 0 9]

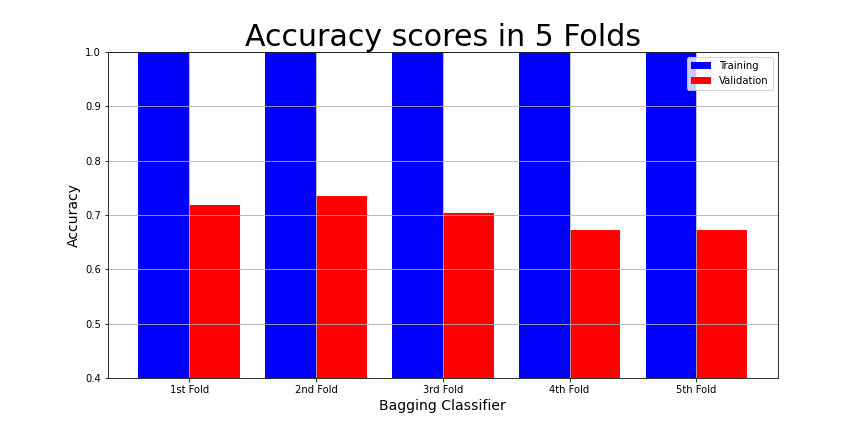
[ 0 5 10]

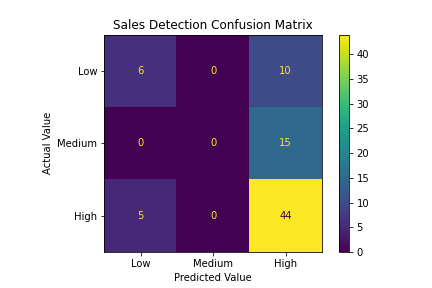
[ 9 4 36]]

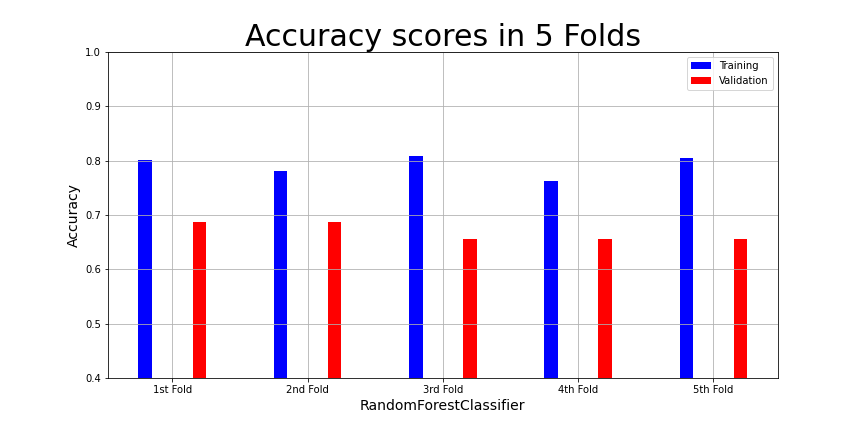
Out[359]: 0.6

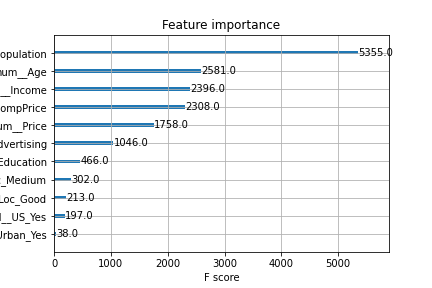
accuracy\_score(Y\_train\_encoded, randomized\_pred\_1)

Out[360]: 1.0

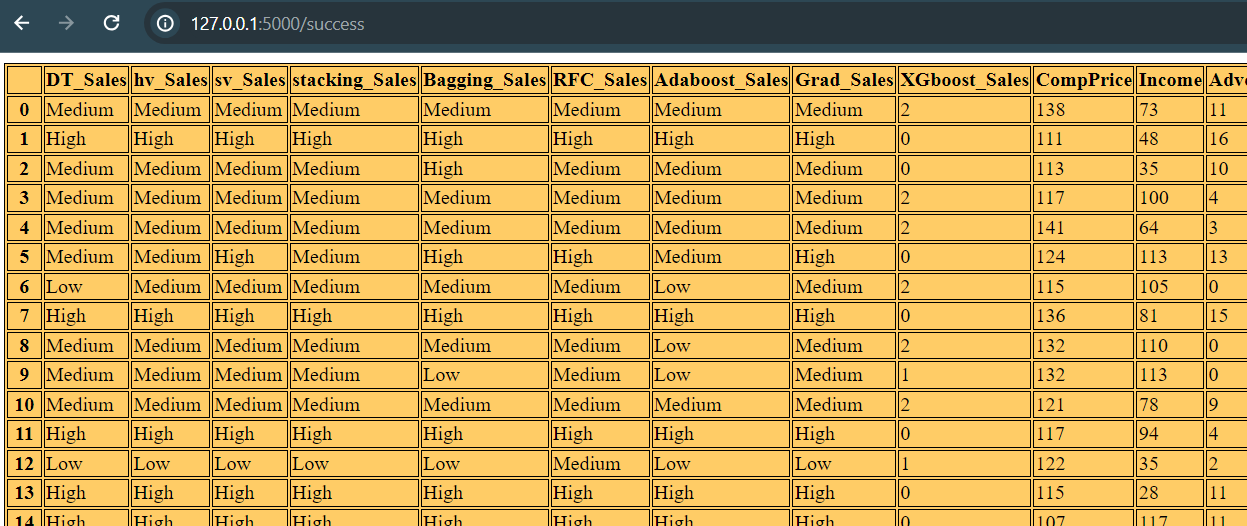




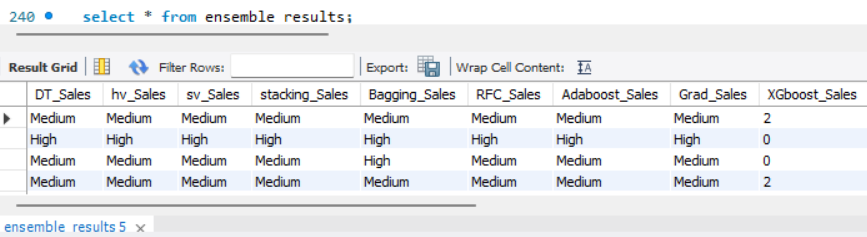




**Deployment of Sales prediction using Ensemble techniques through Flask**

****

**Saving the predicted sales values in MySQL for monitoring**

****