# K-Nearest Neighbors

**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: K-Nearest Neighbors**

**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.**

1. **Data Pre-processing**

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

1. **Exploratory Data Analysis (EDA):**
   1. **Summary.**
   2. **Univariate analysis.**
   3. **Bivariate analysis.**
2. **Model Building**
   1. **Build the model on the scaled data (try multiple options).**
   2. **Perform KNN and use cross-validation techniques to get the optimum K value.**
   3. **Train and test the model and perform cross-validation techniques. Compare accuracies, precision, and recall and explain them in the documentation.**
   4. **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?**

**Problem Statement:**

1. A glass manufacturing plant uses different earth elements to design new glass materials based on customer requirements. For that, they would like to automate the process of classification as it’s a tedious job to manually classify them. Help the company achieve its objective by correctly classifying the glass type based on the other features using KNN algorithm.

A picture containing table

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| Name of Feature | Description | Type | Relevance |
| RI | Refractive Index | Quantitative | Relevant |
| Na | Sodium Content | Quantitative | Relevant |
| Mg | Magnesium Content | Quantitative | Relevant |
| Al | Aluminum Content | Quantitative | Relevant |
| Si | Silicon Content | Quantitative | Relevant |
| K | Potassium Content | Quantitative | Relevant |
| Ca | Calcium Content | Quantitative | Relevant |
| Ba | Barium Content | Quantitative | Relevant |
| Fe | Iron Content | Quantitative | Relevant |
| Type | Type of Glass | Nominal | Relevant |

**CODE:**

­ '''CRISP-ML(Q):

Business Problem:

A glass manufacturing plant uses different earth elements to design new glass materials based on customer requirements. For that, they would like to automate the process of classification as it’s a tedious job to manually classify them. Help the company achieve its objective by correctly classifying the glass type based on the other features.

Business Objective: Maximize Glasstype Detection

Business Constraints: Minimize labour Cost & Maximize productivity

Success Criteria:

Business success criteria: Reduce the time consumption for glass type detection atleast by 50 percent

Machine Learning success criteria: Achieve an accuracy of atleast 98%

Economic success criteria: Reducing labour cost atleast by 20%

Data Collection:

The features of the collected data are

RI Refractive Index

Na Sodium Content

Mg Magnesium Content

Al Aluminum Content

Si Silicon Content

K Potassium Content

Ca Calcium Content

Ba Barium Content

Fe Iron Content

Type Type of Glass

'''

# CODE MODULARITY IS EXTREMELY IMPORTANT

# Import the libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder

from sklearn.preprocessing import MinMaxScaler

# pip install sklearn\_pandas

from sklearn.pipeline import Pipeline

from sklearn\_pandas import DataFrameMapper

from sklearn.compose import ColumnTransformer

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

from sklearn.neighbors import KNeighborsClassifier

import sklearn.metrics as skmet

import pickle

# MySQL Database connection

from sqlalchemy import create\_engine, text

glassdata = pd.read\_csv(r"C:/Users/Lenovo/Downloads/Study material/Data Science/KNN\_Classifier/Assignments/KNN/glass.csv")

# Creating engine which connect to MySQL

user = 'root' # user name

pw = '1234' # password

db = 'glass\_db' # database

# creating engine to connect database

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

# dumping data into database

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

# loading data from database

sql = 'select \* from glass'

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

print(glassdf)

# Data Preprocessing & EDA

glassdf.info() # No missing values observed

a = glassdf.describe()

# Seperating input and output variables

glassdf\_X = pd.DataFrame(glassdf.iloc[:, :9])

glassdf\_y = pd.DataFrame(glassdf.iloc[:, 9])

# glassdf\_y = glassdf\_y.astype(str)

# glassdf\_y.info()

# EDA and Data Preparation

glassdf\_X.info()

import joblib

# Define scaling pipeline

scale\_pipeline = Pipeline([('scale', MinMaxScaler())])

preprocess\_pipeline2 = ColumnTransformer([('scale', scale\_pipeline, glassdf\_X.columns)])

processed2 = preprocess\_pipeline2.fit(glassdf\_X)

processed2

# Save the Scaling pipeline

joblib.dump(processed2, 'processed2')

import os

os.getcwd()

# Normalized data frame (considering the numerical part of data)

glassclean\_n = pd.DataFrame(processed2.transform(glassdf\_X), columns = glassdf\_X.columns)

res = glassclean\_n.describe()

res

# Separating the input and output from the dataset

# X = np.array(glassclean\_n.iloc[:, :]) # Predictors

Y = np.array(glassdf\_y['Type']) # Target

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(glassclean\_n, Y,

test\_size = 0.2, random\_state = 0)

X\_train.shape

X\_test.shape

x\_train = np.array(X\_train)

x\_test = np.array(X\_test)

# Model building

knn = KNeighborsClassifier(n\_neighbors = 11)

KNN = knn.fit(X\_train, Y\_train) # Train the kNN model

# Evaluate the model with train data

pred\_train = knn.predict(x\_train) # Predict on train data

pred\_train

# Cross table

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

print(skmet.accuracy\_score(Y\_train, pred\_train)) # Accuracy measure

import numpy as np

print(np.mean(Y\_train == pred\_train))

# Predict the class on test data

pred = knn.predict(x\_test)

pred

# Evaluate the model with test data

print(skmet.accuracy\_score(Y\_test, pred))

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

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

cmplot = skmet.ConfusionMatrixDisplay(confusion\_matrix = cm)

cmplot.plot()

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

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

# creating empty list variable

acc = []

# running KNN algorithm for 3 to 50 nearest neighbours(odd numbers) and

# storing the accuracy values

for i in range(1, 15, 2):

# print(i)

neigh = KNeighborsClassifier(n\_neighbors = i)

neigh.fit(x\_train, Y\_train)

train\_acc = np.mean(neigh.predict(x\_train) == Y\_train)

test\_acc = np.mean(neigh.predict(x\_test) == Y\_test)

diff = train\_acc - test\_acc

acc.append([diff, train\_acc, test\_acc])

acc

# Plotting the data accuracies

plt.plot(np.arange(1, 15, 2), [i[1] for i in acc], "ro-")

plt.plot(np.arange(1, 15, 2), [i[2] for i in acc], "bo-")

# Hyperparameter optimization

from sklearn.model\_selection import GridSearchCV

k\_range = list(range(1, 21, 2))

param\_grid = dict(n\_neighbors = k\_range)

# Defining parameter range

grid = GridSearchCV(knn, param\_grid, cv = 5,

scoring = 'accuracy',

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

help(GridSearchCV)

KNN\_new = grid.fit(x\_train, Y\_train)

print(KNN\_new.best\_params\_)

accuracy = KNN\_new.best\_score\_ \*100

print("Accuracy for our training dataset with tuning is : {:.2f}%".format(accuracy) )

# Predict the class on test data

pred = KNN\_new.predict(x\_test)

pred

print(skmet.accuracy\_score(Y\_test, pred))

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

cmplot = skmet.ConfusionMatrixDisplay(confusion\_matrix = cm)

cmplot.plot()

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

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

# Save the model

knn\_best = KNN\_new.best\_estimator\_

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

import os

os.getcwd()

**Output:**

pred\_train

Out[222]:

array([1, 1, 2, 1, 2, 1, 2, 1, 2, 7, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1,

2, 2, 1, 1, 1, 2, 6, 1, 2, 1, 1, 1, 7, 1, 2, 1, 1, 2, 2, 1, 1, 2,

1, 1, 2, 7, 1, 1, 1, 7, 1, 2, 1, 2, 1, 6, 7, 1, 1, 1, 2, 7, 7, 6,

1, 2, 1, 2, 1, 5, 7, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 7, 1,

1, 1, 2, 2, 7, 1, 1, 2, 1, 7, 5, 7, 6, 2, 2, 2, 1, 2, 1, 1, 1, 1,

1, 2, 7, 1, 2, 2, 2, 1, 1, 2, 1, 7, 2, 1, 1, 1, 7, 1, 1, 7, 1, 2,

1, 5, 2, 1, 2, 1, 1, 1, 1, 2, 2, 2, 1, 7, 2, 2, 1, 2, 2, 5, 2, 7,

1, 7, 1, 2, 2, 2, 2, 1, 1, 1, 1, 7, 1, 7, 2, 1, 7], dtype=int64)

Predictions 1 2 5 6 7

Actual

1 58 3 0 0 0

2 17 38 0 2 0

3 9 3 0 0 0

5 0 5 3 0 3

6 3 1 0 1 2

7 1 3 1 1 17

print(skmet.accuracy\_score(Y\_train, pred\_train)) # Accuracy measure

0.6842105263157895

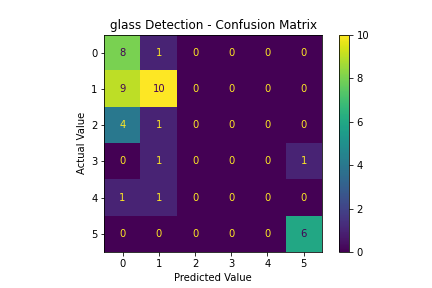
array([7, 1, 2, 1, 2, 2, 1, 2, 2, 2, 1, 1, 1, 2, 2, 7, 1, 1, 1, 2, 7, 1,

7, 7, 1, 1, 7, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 7, 2, 2, 2],

dtype=int64)

print(skmet.accuracy\_score(Y\_test, pred))

0.5581395348837209



[[0.39534883720930236, 1.0, 0.6046511627906976],

[0.17339861281109747, 0.8245614035087719, 0.6511627906976745],

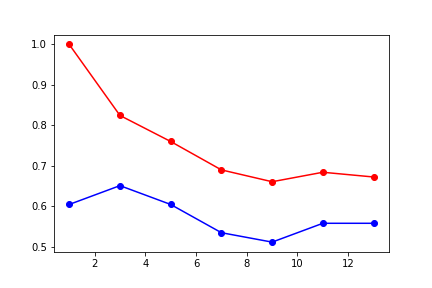
[0.1555827553379573, 0.7602339181286549, 0.6046511627906976],

[0.1551747586019312, 0.6900584795321637, 0.5348837209302325],

[0.1491908064735482, 0.6608187134502924, 0.5116279069767442],

[0.12607099143206857, 0.6842105263157895, 0.5581395348837209],

[0.11437508499932003, 0.672514619883041, 0.5581395348837209]]



KNN\_new = grid.fit(x\_train, Y\_train)

Fitting 5 folds for each of 10 candidates, totalling 50 fits

print(KNN\_new.best\_params\_)

{'n\_neighbors': 1}

Accuracy for our training dataset with tuning is : 70.10%

pred

Out[248]:

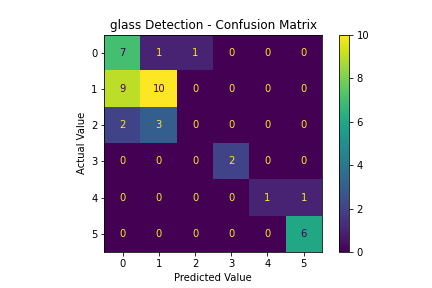
array([7, 1, 1, 6, 5, 2, 1, 2, 1, 2, 2, 1, 2, 2, 2, 7, 1, 1, 1, 2, 5, 1,

7, 7, 1, 1, 7, 1, 2, 2, 1, 3, 2, 1, 1, 2, 1, 1, 1, 7, 2, 7, 2],

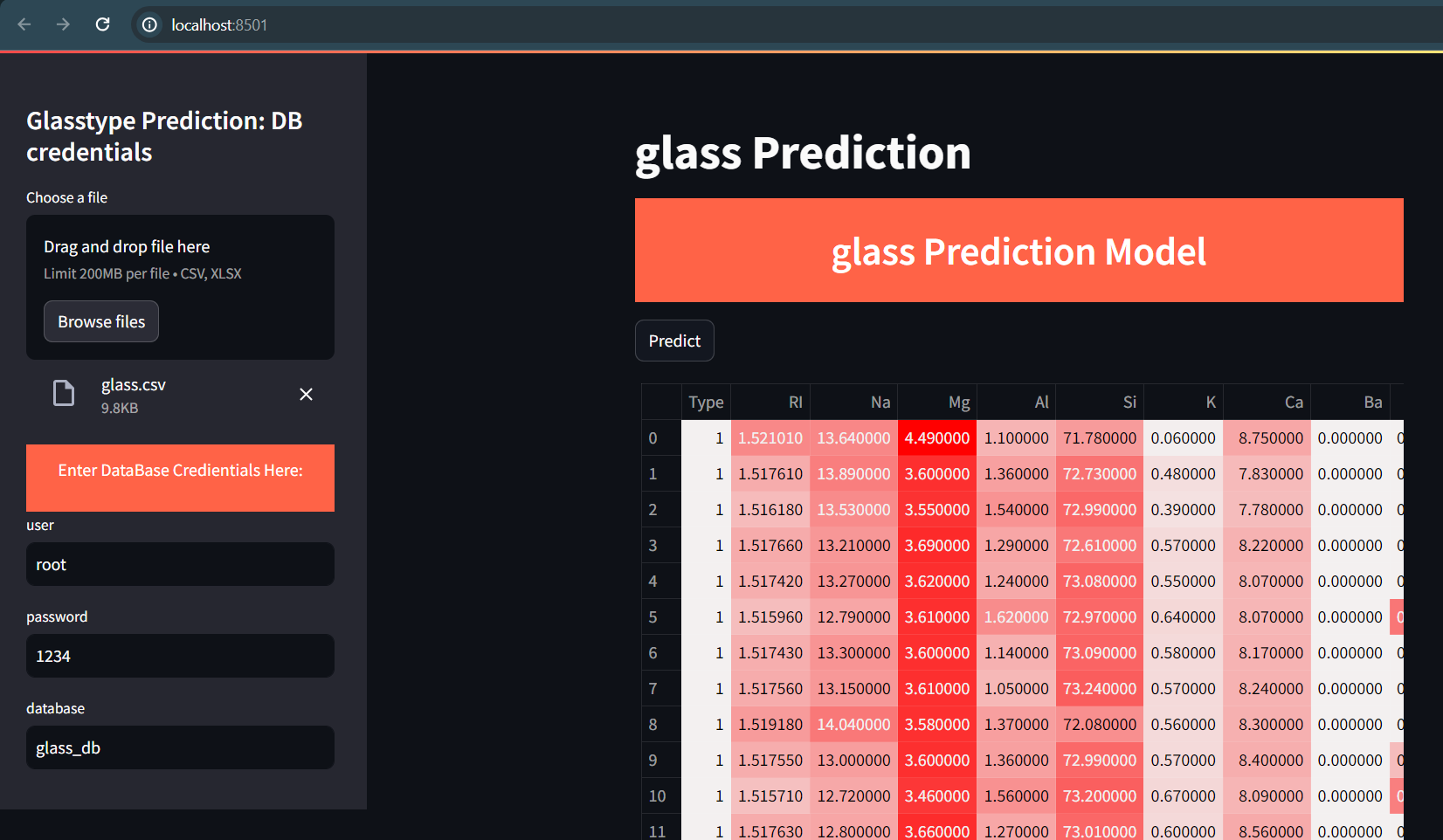
dtype=int64)

print(skmet.accuracy\_score(Y\_test, pred))

0.6046511627906976



**Deployment of the KNN model using Flask**

****