**WEEK 13: Hands on tutorial for Machine Learning Tasks**

**Objective:**

In this lab work we will create machine learning models for three different tasks (linear regression, binary classification, and multiclass classification) discussed in last lecture and evaluate their performance.

**Following Tools and Libraries will be required:**

Google Colab, Python, Scikit-learn

**Datasets and Use Cases:**

1. Wine Quality Dataset

https://raw.githubusercontent.com/fenago/datasets/main/winequalityN.csv

1. Pima Indians Diabetes Dataset

https://raw.githubusercontent.com/fenago/datasets/main/pima-indians-diabetes2.data.csv"

1. Iris Dataset

https://raw.githubusercontent.com/fenago/datasets/main/iris.csv

All datasets will be loaded directly from GitHub URLs.

In this activity, you will work with three different datasets to explore three common machine learning use cases:

1. Regression,
2. Binary classification, and
3. Multiclass classification.

This activity will provide a solid foundation for understanding the fundamental concepts of machine learning and applying them in your project.

**A. Linear Regression using Wine Quality dataset**, a popular dataset containing various physicochemical properties of red wines and their corresponding quality scores. The goal is to create a linear regression model that predicts wine **quality** based on these properties. The understanding gained from this model can help winemakers enhance their production process and optimize the quality of their wines.

**Step 1: Setting up Google Colab environment**

1. Open Google Colab with your MAJU account: Go to <https://colab.research.google.com/> and create a new Python 3 notebook.

2. Install required libraries: In the first cell, type and run:

!pip install -U scikit-learn

# Step 2: Importing required libraries

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression, LogisticRegression  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import mean\_squared\_error, r2\_score, accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix, classification\_report

# Step 3: Load the Wine Quality dataset

# 3.1. Load the dataset from a GitHub URL:

url = "https://raw.githubusercontent.com/fenago/datasets/main/winequalityN.csv"  
wine\_quality\_df = pd.read\_csv(url, sep=",")  
wine\_quality\_df.head()

You will get top 5 rows of your dataset.

wine\_quality\_df.describe()

This statement will show you the statistical description of dataset

wine\_quality\_df.info()

It gives you information about this dataset

# Step 4: Create and evaluate a Linear Regression model

4.1. Split the dataset into features (X) and target (y):

As discussed in the last class that you have featureset containing 1 or more columns (tuple) on the basis of which you have to predict the value (target). So here we are going to predict Quality feature from the feature vector.

# Encode all columns so all data is numeric  
# Also, remove all empty values and replace them with 0  
wine\_quality\_df = pd.get\_dummies(wine\_quality\_df)  
wine\_quality\_df = wine\_quality\_df.fillna(0)  
wine\_quality\_df.head()

The above statements are necessary to convert categorical data (qualitative) to numerical data (quantitative). Forexample type feature is white or red. With introducing dummy variable it will be enumerated as type\_red (0 (if white) and 1 (if red)) and type\_white (1 (if white) and 0 (if red)). Moreover all the null values will be filled with 0.

The feature that we are going to predict will be separated from the dataset.

X = wine\_quality\_df.drop("quality", axis=1)  
y = wine\_quality\_df["quality"]  
print(X.head())  
y.head()

Now X contains whole dataset without target (‘quality’) and y contains array of ‘quality’.

Now we are splitting our data set into test and training.

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

X\_train will contain all dataset use for training, while y\_train contains target dataset for training. X\_test contains those examples that will be tested and y\_test contains dataset for testing the target in model.

Test Size is set at 0.2 means Training dataset is 80% and Testing dataset is 20%. Random state is mostly set at 0 or 42 as seed number for generator. This is necessary for reproducibility. With this statement examples are picked randomly to avoid biasness.

**4.3. Create a Linear Regression model, as discussed in last class. Fit it to the training data (X and y), and Make predictions on the test data (X):**

**4.4. Evaluate the model’s performance by printing Mean Squared Error (MSE) and R-squared (R2) score:**

mse = mean\_squared\_error(y\_test, y\_pred)  
r2 = r2\_score(y\_test, y\_pred)  
print(f"Mean Squared Error: {mse:.2f}")  
print(f"R-squared Score: {r2:.2f}")

lr\_model = LinearRegression()  
lr\_model.fit(X\_train, y\_train)  
y\_pred = lr\_model.predict(X\_test)

You should get MSE as 0.47 and R2 score as 0.34. MSE was discussed in the last class. The r2 score varies between 0 and 100%. So if it is 100%, the two variables are perfectly correlated, i.e., with no variance at all. A low value would show a low level of correlation, meaning a regression model that is not valid, but not in all cases.

**4.5. Submit new data for predictions:**

# Replace the input data with the actual data you want to predict  
input\_data = [[7.4, 0.7, 0.0, 1.9, 0.076, 11, 34, 0.9978, 3.51, 0.56, 9.4,1,1]]  
prediction = lr\_model.predict(input\_data)  
print(f"Predicted Wine Quality: {prediction[0]:.2f}")

Place different values and evaluate the predictions. You may not be able to enjoy it much because of lack of domain knowledge ….p. The Next scenario is understandable better. This also shows that domain understanding is also one of the necessary component.

# B. Binary Classification using Pima Indians Diabetes dataset

As discussed in the last class that Binary Classification is used to classify in one of the two groups. In this task you will explore the Pima Indians Diabetes dataset, which consists of several medical predictor variables and a binary target variable indicating the presence or absence of diabetes. By creating a binary classification model using logistic regression, you will be able to predict whether a person is diabetic based on the given medical attributes. This model can be useful for medical professionals and researchers studying diabetes and its risk factors.

# Step 1: Load the Pima Indians Diabetes dataset

1. Load the dataset from a GitHub URL as under:

url = "https://raw.githubusercontent.com/fenago/datasets/main/pima-indians-diabetes2.data.csv"  
  
column\_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']  
  
pima\_df = pd.read\_csv(url, header=None, names=column\_names)

2. Print the first few rows to confirm successful loading as done in previous dataset:

pima\_df.head()

pima\_df.describe()

pima\_df.info()

pima\_df.shape

Implement each row separately to see first 5 rows, statistical description, dimensions etc. The statements were discussed in ETA part.

# Step 2: Create and evaluate a Binary Classifier model

1. Split the dataset into features (X) and target (y):

# Encode all columns so all data is numeric  
# Also, remove all empty values and replace them with 0  
pima\_df = pd.get\_dummies(pima\_df)  
pima\_df = pima\_df.fillna(0)  
pima\_df.head()

X = pima\_df.drop("Outcome", axis=1)  
y = pima\_df["Outcome"]

In this dataset since all values are numeric you will not see any change due to dummy variable. Binary variable Outcome is considered as target which shows whether a person has these features (X) is diabetic or not. As done in last task Outcome variable is separated from main dataset.

**2. Split the data into training and testing sets:**

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

This step was done in last task so description can be seen from there.

**3. Create a Logistic Regression model, fit it to the training data, and make predictions on the test data:**

log\_model = LogisticRegression(max\_iter=1000)  
log\_model.fit(X\_train, y\_train)  
y\_pred = log\_model.predict(X\_test)  
y\_pred

Each iteration will make the model more accurate till the point where you will see lesser difference. You will see a predictions of 1s and 0s against each example.

**4. Evaluate the model’s performance by printing accuracy, precision, recall, F1-score, and confusion matrix:**

accuracy = accuracy\_score(y\_test, y\_pred)  
precision = precision\_score(y\_test, y\_pred)  
recall = recall\_score(y\_test, y\_pred)  
f1 = f1\_score(y\_test, y\_pred)  
conf\_matrix = confusion\_matrix(y\_test, y\_pred)  
  
print(f"Accuracy: {accuracy:.2f}")  
print(f"Precision: {precision:.2f}")  
print(f"Recall: {recall:.2f}")  
print(f"F1-score: {f1:.2f}")  
print("Confusion Matrix:")  
print(conf\_matrix)

Unlike Regression here performance are evaluated using Accuracy, Precision, Recall, F1 and Confusion Matrix. You will find following scores:

Accuracy: 0.75

Precision: 0.64

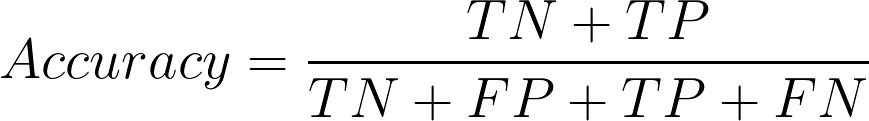
Recall: 0,67

F1 Score: 0.65

Confusion Matrix: [[78 21]

[18 37]]

Accuracy represents the number of correctly classified data instances over the total number of data instances.



Accuracy is not proper metric when dataset is not balanced. For example there are 90 people who are healthy (positive) and 10 people who have some disease (negative). Now let’s say our machine learning model perfectly classified the 90 people as healthy but it also classified the unhealthy people as healthy. What will happen in this scenario? TP = 90, FP = 0, FN = 10 and TN = 0. Accuracy in this case is 90%, but this model is very poor because all the 10 people who are unhealthy are classified as healthy. Using accuracy in such scenarios can result in misleading interpretation of results. In such scenarios Precision is the better metric.

A black background with a black square

Description automatically generated with medium confidence

For unhealthy predictions it is giving 10% Precision if we consider unhealthy as True Positive. **Precision** should ideally be 1 (high) for a good classifier. **Precision** becomes 1 only when the numerator and denominator are equal i.e **TP = TP +FP**, this also means **FP** is zero. As **FP** increases the value of denominator becomes greater than the numerator and **precision** value decreases.

Now we will introduce another important metric called ***recall***.

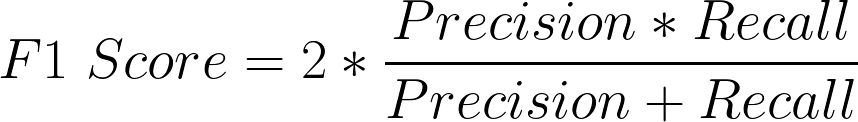
***Recall*** is also known as  ***true positive rate*** and is defined as follows:

A black background with a black square

Description automatically generated with medium confidence

***Recall*** should ideally be 1 (high) for a good classifier. ***Recall*** becomes 1 only when the numerator and denominator are equal i.e ***TP = TP +FN***, this also means ***FN*** is zero. As ***FN*** increases the value of denominator becomes greater than the numerator and ***recall*** value decreases.

F1-score is a metric which takes into account both precision and recall and is defined as follows:



**5.Submit new data for predictions:**

# Replace the input data with the actual data you want to predict  
input\_data = [[1, 85, 66, 29, 0, 26.6, 0.351, 31]]  
prediction = log\_model.predict(input\_data)  
print(f"Predicted Diabetes Outcome: {prediction[0]}")

For better understanding you can place different values here because of relatively better understanding to know if the person is diabetic or not. For example change the level of BP, or Glucose or Age and see the effect on Prediction.

**C. Multiclass Classification for Iris dataset**

This type of classification was discussed in last class in which we can have more than two classes. This famous Iris dataset, contains the measurements of sepal length, sepal width, petal length, and petal width of three different species of iris flowers. The goal is to build a multiclass classification model using k-Nearest Neighbors to predict the species of a flower based on its measurements. This use case demonstrates the application of machine learning in the field of botany and taxonomy.

# Step 1: Load the Iris dataset

url = "https://raw.githubusercontent.com/fenago/datasets/main/iris.csv"  
  
column\_names = ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width', 'species']  
  
iris\_df = pd.read\_csv(url, header=None, names=column\_names, skiprows=1)

Print the first few rows to confirm successful loading:

iris\_df.head()

iris\_df.describe()

iris\_df.shape

# Step 2: Create and evaluate a Multiclass Classifier model

1. Split the dataset into features (X) and target (y):

# Also, remove all empty values and replace them with 0  
iris\_df = iris\_df.fillna(0)  
iris\_df.head()

X = iris\_df.drop("species", axis=1)  
y = iris\_df["species"]

2. Split the data into training and testing sets:

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

3. Create a k-Nearest Neighbors (k-NN) model, fit it to the training data, and make predictions on the test data:

knn\_model = KNeighborsClassifier(n\_neighbors=3)  
knn\_model.fit(X\_train, y\_train)  
y\_pred = knn\_model.predict(X\_test)

4. Evaluate the model’s performance by printing accuracy, precision, recall, F1-score, and confusion matrix:

accuracy = accuracy\_score(y\_test, y\_pred)  
conf\_matrix = confusion\_matrix(y\_test, y\_pred)  
classification\_rep = classification\_report(y\_test, y\_pred)  
  
print(f"Accuracy: {accuracy:.2f}")  
print("Confusion Matrix:")  
print(conf\_matrix)  
print("Classification Report:")  
print(classification\_rep)

5. Submit new data for predictions:

# Replace the input data with the actual data you want to predict  
input\_data = [[5.1, 3.5, 1.4, 0.2]]  
prediction = knn\_model.predict(input\_data)  
print(f"Predicted Iris Species: {prediction[0]}")

**D. Conclusion**

Throughout the activity, you learned to create and evaluate machine learning models using Google Colab and Python. This hands-on experience will solidify your understanding of the underlying concepts and give you the confidence to apply these techniques to your datasets of the project whose EDA you already completed.