|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Kingdom of Saudi Arabia**  **Ministry of Education**  **University of Jeddah**  **College of Computer Science and Engineering**  **Department of Computer Science and Artificial Intelligence** | Logo, company name  Description automatically generated | **المملكة العربية السعودية**  **وزارة التعليم**  **جامعة جدّة**  **كلية علوم وهندسة الحاسب**  **قسم علوم الحاسب والذكاء الاصطناعي** | |  |  |

|  |
| --- |
| **Lab 6** |
| **CCAI 312 Pattern Recognition** |
| **Third Trimester 2023**   |  |  | | --- | --- | | **Lab Date/Time:**  **Lab assignment submission Date/Time:** |  | | **Student Name: \_\_\_\_\_\_\_bushra dajam\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Student ID: \_\_\_\_\_\_2110054\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | | |

|  |  |
| --- | --- |
| **Instructor Name** | **Section** |
|  |  |

**Instructions**:

The lab assignments must be submitted before the allocated Date/Time.

The lab assignments must by uploaded on LMS / sent by email to teacher@uj.edu.sa.

Plagiarism will be punished according to university rules.

|  |  |
| --- | --- |
| **PLO/CLO** | **SO** |
| **PLO S2 (CLO 2):** **Implement** a suitable pattern recognition technique for a given problem using Python | **SO 2:** Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program’s discipline |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Max Score** | **Student Score** |
| **PLO S2 / CLO 2 / SO 2** | **Task 1** | **2** |  |
| **PLO S2 / CLO 2 / SO 2** | **Task 2** | **2** |  |
| **Total** | |  |  |

# Lab Description

# In this lab, we will learn about naive bayes classifiers and walk through an example of how to create naive bayes model using Scikit-Learn. We'll see how the algorithm works with both numeric and non-numeric data. We will also learn how to evaluate the algorithm's training and testing accuracy and improve it using grid search and cross-validation.

# Objectives

* Utilize features correlation and a correlation **heatmap** to analyze and pinpoint the most crucial features.
* Creating and evaluating income prediction models using **naive bayes**.
* Acquire knowledge on how to use **Gridsearch** for hyper parameter tuning.
* Get to know the pandas **getdummies** method for converting categorical features to numeric features.
* Acquire knowledge on how to use CountVectorizer to Convert the text data into numerical features

# Lab Tool(s)

<https://www.kaggle.com/>

or

<https://colab.research.google.com/>

# Lab Deliverables

Submit A notebook to Blackboard containing your solution to the lab assessment at the end of this document.

# References:

<https://www.kaggle.com/datasets/wenruliu/adult-income-dataset>

Other Resources:

<https://www.kaggle.com/code/akshaysharma001/naive-bayes-with-hyperpameter-tuning>

https://www.kaggle.com/code/geoffreygeo/income-prediction-with-census-data/notebook

Naive Bayes

* 1. **Introducing Naive Bayes classifiers**

scikit-learn provides multiple implementations of Naive Bayes that differ on how conditional probabilities are calculated. So the different implementations are suitable for different types of data.

* **CategorialNB**

will work with categorical data once it is processed using an OrdinalEncoder

* **GaussianNB**

assumes the numerica features have a Gaussian distribution

* **BernoulliNB**

binary data

* **MultinomialNB**

count data, e.g. word counts

Part1

## **Case Study:** Income Prediction on Census Data

**Business Objective**:

Often, financial institutions need to categorize their customers based on their income, however, generic features like age, work-sector, education, are not enough to predict their income. The Adult Census dataset available in UCI ML repository, uses 14 features to predict an individual's income, thus helping in binomial classification of individuals based on their income. The result is useful in accurate service recommendation. Individuals with >= 50k income fall in medium wage income and thus can be segmented to respective promotional services, while those with < 50k income, can be provided with low cost services. Those above >=50k income are eligible for expensive offers and promotional memberships for clubs, credit cards, etc. while those with income <50k, shall be targeted with more budget constrained offers. The categorization can also help in predicting the life-style of the individual.

**Dataset**: This data was extracted from the [1994 Census bureau database](http://www.census.gov/en.html)

Understand each of the steps and practice the solution by replicating it:

**Step 1**: Import Libraries

import pandas as pd  
import numpy as np  
from sklearn import preprocessing  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split# used to split the dataset into train and test datasets  
from sklearn.naive\_bayes import GaussianNB # To model the Gaussian Naive Bayes classifier  
from sklearn.metrics import accuracy\_score # To calculate the accuracy score of the model  
from sklearn.metrics import confusion\_matrix

**Step 2**: Reading and understanding Dataset (EDA)

data = pd.read\_csv('adult.csv')

data.head()

Graphical user interface, application

Description automatically generated

data.shape



data.info()

Table

Description automatically generated

len(data)



data.describe()

Table

Description automatically generated

data.describe(include= 'all')

data.income.unique()



data.income.value\_counts()

Text

Description automatically generated

data.income=data.income.replace(['<=50K', '>50K'],[0,1])

# count plot on single categorical variable  
sns.countplot(x ='income', data = data)  
   
# Show the plot  
plt.show()

Chart, bar chart

Description automatically generated

**Correlations Heatmap**

Visualise how well features correlate with each other with a simple heatmap.

We first need to create a correlation matrix. This is very easy to do by calling upon the .corr() method on our dataframe.

The correlation matrix provides us with an indication of how well (or not so well) each feature is correlated with each other. The returned value will be between -1 and +1, with higher correlations tending toward these endpoints, and poorer correlations tending towards 0.

We can then call upon the seaborn heatmap using sns.heatmap() and passing in the correlation matric ( corr).

# plotting correlation heatmap  
plt.figure(figsize=(7,6))  
sns.heatmap(data.corr(),annot=True,cmap='Blues')

Graphical user interface, application

Description automatically generated

# Let us see the significant correlation either negative or positive among independent attributes..  
c = data.corr().abs() # Since there may be positive as well as -ve correlation  
s = c.unstack() #   
so = s.sort\_values(ascending=False) # Sorting according to the correlation  
so=so[(so<1) & (so>0.3)].drop\_duplicates().to\_frame() # Due to symmetry.. dropping duplicate entries.  
so.columns = ['correlation']  
so

Graphical user interface, application

Description automatically generated

**Step 3**: Data Pre\_Processing

**Step 3.1**: We will now proceed to data preprocessing.

data\_rev = data.copy()

**Step 3.2**: Check the presence of null values in our dataset, as follows:

data.isnull().sum()

Table

Description automatically generated with medium confidence

There can be some categorical variables having missing values. We will check that, sometimes they have “?” in place of missing values.

for value in data\_rev.columns:  
    print(value,":", sum(data\_rev[value] == '?'))

Text

Description automatically generated with medium confidence

The output of the preceding code snippet shows that there are 2799 missing values in the workclass attribute, 2809 missing values in the occupation attribute, and 857 values in the native\_country attribute.

**Step 3.3**: We will now impute the missing categorical values:

for value in ['workclass','education','marital-status','occupation','relationship','race','gender','native-country','income']:  
    replaceValue = data\_rev.describe(include='all')[value][2]  
    data\_rev[value][data\_rev[value]=='?'] = replaceValue

Check again for missing values:

for value in data\_rev.columns:  
    print(value,":", sum(data\_rev[value] == '?'))

A picture containing scatter chart

Description automatically generated

**Step 3.4**: normalize numeric features.

numericalcols=list(data.select\_dtypes(exclude='object').columns)  
numericalcols.pop()  
numericalcols

from sklearn.preprocessing import StandardScaler  
M=StandardScaler()  
data\_rev[numericalcols]=M.fit\_transform(data\_rev[numericalcols])

**Step 4**: Prepare the dataset for model training and evaluation.

**Step 4.1**: We will now get our data arranged into dependent variables and target variable:

x=data\_rev.drop(['income'],axis=1)  
y=data\_rev.income

x.head()

**Graphical user interface

Description automatically generated with low confidence**

**Step 4.2**: One-hot encoding to convert all the categorical variables to numeric

x=pd.get\_dummies(x)

x.head()

**Graphical user interface, text, application

Description automatically generated**

**Step 4.3**: Now split the data into train and test in the ratio of 75:25.

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 5)

**Step 5**: Train and test the model

**Step 5.1**: We will fit the naïve Bayes model now.

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

Graphical user interface, text

Description automatically generated

**Step 5.2:** Check the training and testing accuracy of the model.

The model classifier is now trained using training data and is ready to make predictions. We can use the predict() method with train set features as its parameter to

get training accuarcy or with test set features to get test accuarcy

Y\_pred\_train = clf.predict(X\_train)

train\_Accuracy=accuracy\_score(Y\_train, Y\_pred\_train)  
print("Train Accuracy: "+str(train\_Accuracy))



Y\_pred\_test = clf.predict(X\_test)

test\_Accuracy=accuracy\_score(Y\_test, Y\_pred\_test)  
print("Test Accuracy: "+str(test\_Accuracy))



**Step 5.4**: Draw confusion matrix for both training set and testing set

print("Train confusion\_matrix")  
confusion\_matrix(Y\_train, Y\_pred\_train)

Text

Description automatically generated

print("Test confusion\_matrix")  
confusion\_matrix(Y\_test, Y\_pred\_test)

Text

Description automatically generated

## **Step 6:** Improve the model by features selection or Hyperparameters tuning.

### **Step 6.1**: Tune the naive bayes model hyperparameters (var\_smoothing)

np.logspace(0,-9, num=10)

from sklearn.model\_selection import RepeatedStratifiedKFold  
  
cv\_method = RepeatedStratifiedKFold(n\_splits=5,   
                                    n\_repeats=3,   
                                    random\_state=999)

from sklearn.preprocessing import PowerTransformer  
from sklearn.model\_selection import train\_test\_split,GridSearchCV  
params\_NB = {'var\_smoothing': np.logspace(0,-9, num=100)}  
clf = GaussianNB()  
gs\_NB = GridSearchCV(estimator=clf,   
                     param\_grid=params\_NB,   
                     cv=cv\_method,  
                     verbose=1,   
                     scoring='accuracy')  
  
Data\_transformed = PowerTransformer().fit\_transform(X\_test)  
  
gs\_NB.fit(Data\_transformed, Y\_test);



gs\_NB.best\_params\_



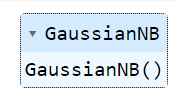
gs\_NB.best\_score\_

A picture containing text

Description automatically generated

**Step 6.2**: Features Selection(train the model using only numeric features)

# select only numeric features  
clf = GaussianNB()  
clf.fit(X\_train[numericalcols], Y\_train)



Y\_pred\_train = clf.predict(X\_train[numericalcols])

train\_Accuracy=accuracy\_score(Y\_train, Y\_pred\_train)  
print("Train Accuracy: "+str(train\_Accuracy))



print("Train confusion\_matrix")  
confusion\_matrix(Y\_train, Y\_pred\_train)

Text

Description automatically generated

Y\_pred\_test = clf.predict(X\_test[numericalcols])

test\_Accuracy=accuracy\_score(Y\_test, Y\_pred\_test)  
print("Test Accuracy: "+str(test\_Accuracy))

Text

Description automatically generated with medium confidence

print("Test confusion\_matrix")  
confusion\_matrix(Y\_test, Y\_pred\_test)

Text

Description automatically generated

**Step 6.3:** train naive bayes model using best features and best hyperparameters

clf = GaussianNB(var\_smoothing=0.06579)  
clf.fit(X\_train[numericalcols], Y\_train)

Y\_pred\_train = clf.predict(X\_train[numericalcols])

train\_Accuracy=accuracy\_score(Y\_train, Y\_pred\_train)  
print("Train Accuracy: "+str(train\_Accuracy))



print("Train confusion\_matrix")  
confusion\_matrix(Y\_train, Y\_pred\_train)

Text

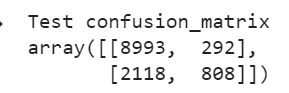
Description automatically generated

Y\_pred\_test = clf.predict(X\_test[numericalcols])

test\_Accuracy=accuracy\_score(Y\_test, Y\_pred\_test)  
print("Test Accuracy: "+str(test\_Accuracy))



print("Test confusion\_matrix")  
confusion\_matrix(Y\_test, Y\_pred\_test)



Part2

**News Articles Classification**

## **Dataset**

“20 Newsgroups" dataset, which is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The newsgroups cover a variety of topics such as religion, politics, computer hardware and software, sports, and space, among others.

The dataset is commonly used in natural language processing (NLP) tasks, including text classification and topic modeling. The dataset has been preprocessed and split into training and testing subsets by the Scikit-Learn library for easy access and usage. The training subset contains 11,314 documents, while the testing subset contains 7,532 documents.

Task **Task 1: [PLO S2 / CLO 2 / SO 2] [2 marks]**

1. Import the necessary libraries: sklearn.datasets, sklearn.feature\_extraction.text, and sklearn.naive\_bayes.

# Step 1: Import necessary libraries  
from sklearn.datasets import fetch\_20newsgroups  
from sklearn.feature\_extraction.text import CountVectorizer  
from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score

1. Load the "20 Newsgroups" dataset using the fetch\_20newsgroups function and specify the categories to be used for training and testing.

# Step 2: Load the "20 Newsgroups" dataset  
categories = ['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc']

newsgroups\_train = fetch\_20newsgroups(subset='train', categories=categories)  
newsgroups\_test = fetch\_20newsgroups(subset='test', categories=categories)

1. Convert the text data into numerical features using the CountVectorizer class from Scikit-Learn.

# Convert text data into numerical features  
vectorizer = CountVectorizer()  
X\_train = vectorizer.fit\_transform(newsgroups\_train.data)  
X\_test = vectorizer.transform(newsgroups\_test.data)  
y\_train = newsgroups\_train.target  
y\_test = newsgroups\_test.target

1. Train the Naive Bayes classifier on the training data using the MultinomialNB class from Scikit-Learn.

# Step 4: Train the Naive Bayes classifier  
clf = MultinomialNB()  
clf.fit(X\_train, y\_train)

1. Calculate the training accuracy

# Step 5: Calculate the training accuracy  
train\_predictions = clf.predict(X\_train)  
train\_accuracy = accuracy\_score(y\_train, train\_predictions)

1. Evaluate the performance of the classifier on the test data

# Step 6: Evaluate the performance of the classifier on test data  
predictions = clf.predict(X\_test)  
accuracy = accuracy\_score(y\_test, predictions)

1. Print the training and test accuracies of the classifier.

# Step 7:Print the training and test accuracies of the classifier  
print("Training accuracy:", train\_accuracy)  
print("Test accuracy:", test\_accuracy)

**Task? 2: [PLO C4 / CLO 3 / SO 7] [2 mark]**

1. use GridSearchCV to perform hyperparameter tuning for the Naive Bayes model. Define a range of values for the alpha hyperparameter, which controls the smoothing of the probability estimates.

# Hyperparameter tuning using GridSearchCV  
param\_grid = {'alpha': [0.1, 1.0, 10.0]}  
grid\_search = GridSearchCV(clf, param\_grid=param\_grid, cv=5)  
grid\_search.fit(X\_train, y\_train)  
print("Best hyperparameters:", grid\_search.best\_params\_)  
print("Best cross-validation score: {:.2f}%".format(grid\_search.best\_score\_ \* 100))

1. Evaluate the model with the best hyperparameters on the testing set and report the testing accuracy.

# Evaluate the model with the best hyperparameters on the testing set  
best\_clf = grid\_search.best\_estimator\_  
y\_test\_pred = best\_clf.predict(X\_test)  
test\_acc = accuracy\_score(y\_test, y\_test\_pred)  
print("Testing accuracy with best hyperparameters: {:.2f}%".format(test\_acc \* 100))

1. Does the model improve with hyperparameters tuning?

Yes, the model’s performance improved.