This lab is to deal with classification task using **Random Forests** and **Naïve Bayes** algorithms with/without **Feature Selection**.

```
from google.colab import drive
drive.mount('/content/gdrive')
%cd '/content/gdrive/MyDrive/Lab6/data'
```

```
Mounted at /content/gdrive
/content/gdrive/MyDrive/Lab6/data
```

Import libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression,LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from prettytable import PrettyTable
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest
from sklearn import metrics
from sklearn import svm
from sklearn.preprocessing import LabelEncoder
```

Task 1.

Task 1. Compare the performance of selected classification algorithms including **Random forest**, **NaiveBayes**, and **SVM** with **FASHION** dataset based on **accuracy, precision, recall, f1** measures according to **without using selection feature** and **using selection feature**.

```
algoDict = {
    0:"Random forest",
    1:"NaiveBayes GaussianNB:",
    2:"SVM linear",
    3:"Decision tree",
    4:"Logistic Regression",
    5: "kNN k=5"
}
def getScores(algo,X_train,X_test,y_train,y_test):

  if(algo==0):
    clf = RandomForestClassifier()
  elif(algo==1):
    clf = GaussianNB()
  elif(algo==2):
    clf = GaussianNB()
  elif(algo==3):
    clf = DecisionTreeClassifier()
  elif(algo==4):
    clf = LogisticRegression(max_iter=50000)
  elif(algo==5):
    clf = KNeighborsClassifier()
  clf.fit(X_train,y_train.values.ravel())
  y_pred =clf.predict(X_test)
  ac = metrics.accuracy_score(y_test,y_pred)
  pre = metrics.precision_score(y_test,y_pred,average="macro")
  recall = metrics.recall_score(y_test,y_pred,average="macro")
  f1 = metrics.f1_score(y_test,y_pred,average="macro")
  algoName = algoDict.get(algo)
  return [algoName,ac,pre,recall,f1]
```

```
fashtionData = pd.read_csv("fashion_train.csv")
fashtionData
X = fashtionData.drop(columns="y")
y = fashtionData[["y"]]
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
print("without using selection feature ")
table = PrettyTable(["Algorithms","Accuracy","Precision","Recall","F1"])
for i in range(0,3):
  table.add_row(getScores(i,X_train,X_test,y_train,y_test))
print(table)
```

```
print("using selection feature Univariate Selection")
table2 = PrettyTable(["Algorithms","Accuracy","Precision","Recall","F1"])
newX = SelectKBest(chi2,k=400).fit_transform(X,y)
X_train,X_test,y_train,y_test = train_test_split(newX,y,test_size=0.2)
for i in range(0,3):
  table2.add_row(getScores(i,X_train,X_test,y_train,y_test))
print(table2)
```

```
# print(table2)
```

```
without using selection feature
+------------------------+----------+--------------------+--------------------+--------------------+
|       Algorithms       | Accuracy |     Precision      |       Recall       |         F1         |
+------------------------+----------+--------------------+--------------------+--------------------+
|     Random forest      |  0.805   |  0.81211572551726  | 0.8052204236177516 | 0.8027100227437811 |
| NaiveBayes GaussianNB: |   0.51   | 0.5262079831932773 | 0.5214047164528387 | 0.4851300348800004 |
|       SVM linear       |   0.8    | 0.8098344428264337 | 0.8020803476372865 | 0.8027627006852136 |
+------------------------+----------+--------------------+--------------------+--------------------+
using selection feature Univariate Selection
+------------------------+----------+--------------------+--------------------+--------------------+
|       Algorithms       | Accuracy |     Precision      |       Recall       |         F1         |
+------------------------+----------+--------------------+--------------------+--------------------+
|     Random forest      |  0.805   | 0.8116692259147502 | 0.8046193956504901 | 0.8027270094825398 |
| NaiveBayes GaussianNB: |   0.55   | 0.5099444361944363 | 0.549953931741521  | 0.4887851519233418 |
|       SVM linear       |  0.775   | 0.7863304726462621 | 0.7701719239167079 | 0.773747590093268  |
```

```
+----------------------+----------+-------------------+-------------------+--------------------+
```

Task 2.

For given bank dataset (**bank.csv**) having the following attributes :

1. **age** (numeric)
2. **job** : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
3. **marital** : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
4. **education** (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')
5. **default**: has credit in default? (categorical: 'no','yes','unknown')
6. **housing**: has housing loan? (categorical: 'no','yes','unknown')
7. **loan**: has personal loan? (categorical: 'no','yes','unknown')
8. **contact**: contact communication type (categorical: 'cellular','telephone')
9. **month**: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
10. **day_of_week**: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
11. **duration**: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.
12. **campaign**: number of contacts performed during this campaign and for this client (numeric, includes last contact)
13. **pdays**: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
14. **previous**: number of contacts performed before this campaign and for this client (numeric)
15. **poutcome**: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success') Output variable (desired target):
16. **y**. has the client subscribed a term deposit? (binary: 'yes','no')

2.1. Apply StandardScaler() function to columns that contains numerical data ('age', 'balance', 'day', 'campaign', 'pdays', 'previous')

```
def myScaler(data,columns):
  scaler = StandardScaler()
  for i in columns:
    data[[i]] =scaler.fit_transform(data[[i]])
    data[[i]] += abs(data[[i]].min())

  return  data


bank_data= pd.read_csv("bank.csv")
columns = ['age', 'balance', 'day', 'campaign', 'pdays', 'previous']
new_bank_data= myScaler(bank_data,columns)
new_bank_data
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	pr
0	3.441666	admin.	married	secondary	no	2.849375	yes	no	unknown	0.475039	may	1042	0.000000	0.000000	0.
1	3.189837	admin.	married	secondary	no	2.136876	no	no	unknown	0.475039	may	1467	0.000000	0.000000	0.
2	1.930691	technician	married	secondary	no	2.516689	yes	no	unknown	0.475039	may	1389	0.000000	0.000000	0.
3	3.105894	services	married	secondary	no	2.890612	yes	no	unknown	0.475039	may	579	0.000000	0.000000	0.
4	3.021951	admin.	married	tertiary	no	2.179973	no	no	unknown	0.475039	may	673	0.367383	0.000000	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
11157	1.259146	blue-collar	single	primary	no	2.123234	yes	no	cellular	2.256435	apr	257	0.000000	0.000000	0.
11158	1.762804	services	married	secondary	no	2.350192	no	no	unknown	1.781396	jun	83	1.102149	0.000000	0.
11159	1.175203	technician	single	secondary	no	2.131915	no	no	cellular	2.137675	aug	156	0.367383	0.000000	0.
11160	2.098577	technician	married	secondary	no	2.122924	no	yes	cellular	0.831318	may	9	0.367383	1.590755	2.
11161	1.343089	technician	married	secondary	no	2.122924	no	no	cellular	0.950078	jul	628	0.000000	0.000000	0.

2.2. Apply Encode Categorical Value (OneHotEncoder) to transfrom categorical data to numerical data ('job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome')

```
def myLabelEncoder(data,columns):
  labelEncoder =  LabelEncoder()
  for i in columns:
    data[i]=labelEncoder.fit_transform(data[i])
  return data
```

```
columns = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome']
new_bank_data = myLabelEncoder(new_bank_data,columns)
new_bank_data
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previou
0	3.441666	0	1	1	0	2.849375	1	0	2	0.475039	8	1042	0.000000	0.000000	0.00000
1	3.189837	0	1	1	0	2.136876	0	0	2	0.475039	8	1467	0.000000	0.000000	0.00000
2	1.930691	9	1	1	0	2.516689	1	0	2	0.475039	8	1389	0.000000	0.000000	0.00000
3	3.105894	7	1	1	0	2.890612	1	0	2	0.475039	8	579	0.000000	0.000000	0.00000
4	3.021951	0	1	2	0	2.179973	0	0	2	0.475039	8	673	0.367383	0.000000	0.00000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
11157	1.259146	1	2	0	0	2.123234	1	0	0	2.256435	0	257	0.000000	0.000000	0.00000
11158	1.762804	7	1	1	0	2.350192	0	0	2	1.781396	6	83	1.102149	0.000000	0.00000
11159	1.175203	9	2	1	0	2.131915	0	0	0	2.137675	1	156	0.367383	0.000000	0.00000
11160	2.098577	9	1	1	0	2.122924	0	1	0	0.831318	8	9	0.367383	1.590755	2.18159
11161	1.343089	9	1	1	0	2.122924	0	0	0	0.950078	5	628	0.000000	0.000000	0.00000

11162 rows × 17 columns

```
Start coding or generate with AI.
```

2.3. Apply **Decision tree, Logistic Regression, Random forest, kNN, NaïveBayes, SVM** algorithms to the preproceed dataset in the previous steps. Then compare the obtained results using **accuracy, precision, recall, f1** measures.

```
X = new_bank_data.drop(columns="deposit")
y = new_bank_data[["deposit"]]
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
print("without using selection feature ")
table = PrettyTable(["Algorithms","Accuracy","Precision","Recall","F1"])
for i in range(0,6):
  table.add_row(getScores(i,X_train,X_test,y_train,y_test))
print(table)
```

```
without using selection feature
+------------------------+--------------------+--------------------+--------------------+--------------------+
|       Algorithms       |      Accuracy      |     Precision      |       Recall       |         F1         |
+------------------------+--------------------+--------------------+--------------------+--------------------+
|     Random forest      | 0.8549037169726825 | 0.8544720554702405 | 0.8564540047376541 |  0.85461795547382  |
| NaiveBayes GaussianNB: | 0.7505597850425436 | 0.7540884029529951 | 0.7544746303145311 | 0.7505525812025122 |
|       SVM linear       | 0.7505597850425436 | 0.7540884029529951 | 0.7544746303145311 | 0.7505525812025122 |
|     Decision tree      | 0.7908643081056874 | 0.7896587516796885 | 0.7904435257440505 | 0.7899664308345685 |
|  Logistic Regression   | 0.8007165248544559 | 0.799928695285786  | 0.7986480904156097 | 0.7991692591948096 |
|        kNN k=5         | 0.741603224361845  | 0.740282016213945  | 0.7389897146434818 | 0.739471255094293  |
+------------------------+--------------------+--------------------+--------------------+--------------------+
```

2.4. Using a selection feature technique to above dataset, then compare the classification results with those in Task 2.3.

```
X = new_bank_data.drop(columns="deposit")
y = new_bank_data[["deposit"]]
newX = SelectKBest(chi2,k=10).fit_transform(X,y)
X_train,X_test,y_train,y_test = train_test_split(newX,y,test_size=0.2)
print("using selection feature ")
table = PrettyTable(["Algorithms","Accuracy","Precision","Recall","F1"])
for i in range(0,6):
  table.add_row(getScores(i,X_train,X_test,y_train,y_test))
print(table)
```

```
using selection feature
+------------------------+--------------------+--------------------+--------------------+--------------------+
|       Algorithms       |      Accuracy      |     Precision      |       Recall       |         F1         |
+------------------------+--------------------+--------------------+--------------------+--------------------+
|     Random forest      | 0.7971339005821765 | 0.7966284264545884 | 0.7985326821610779 | 0.7966757339067544 |
| NaiveBayes GaussianNB: | 0.7299596954769368 | 0.7293808063447129 | 0.7308411297967832 | 0.7293265321506646 |
|       SVM linear       | 0.7299596954769368 | 0.7293808063447129 | 0.7308411297967832 | 0.7293265321506646 |
|     Decision tree      | 0.7429467084639498 | 0.7413284722188442 | 0.7403696920659377 | 0.7407618201617066 |
|  Logistic Regression   | 0.7926556202418271 | 0.7914280880850739 | 0.7907362187111893 | 0.7910457685554986 |
|        kNN k=5         | 0.7510076130765786 | 0.7502387694105486 | 0.7517724740831182 | 0.7503454610678648 |
+------------------------+--------------------+--------------------+--------------------+--------------------+
```

Task 3.

For a given dataset (**mobile price classification**), perform feature selection techniques and then compare the performance of selected classification algorithms (**Decision Tree, kNN, Logistic Regression, SVM, Random Forest, and NaiveBayes**) based on **accuracy, precision, recall, and f1** measures.

```
mobile_data = pd.read_csv("mobile.csv")
X = mobile_data.drop(columns="price_range")
y = mobile_data[["price_range"]]
newX = SelectKBest(chi2,k=10).fit_transform(X,y)
X_train,X_test,y_train,y_test = train_test_split(newX,y,test_size=0.2)
print("using selection feature ")
table = PrettyTable(["Algorithms","Accuracy","Precision","Recall","F1"])
for i in range(0,6):
  table.add_row(getScores(i,X_train,X_test,y_train,y_test))
print(table)
```

```
using selection feature
+------------------------+----------+--------------------+--------------------+--------------------+
|       Algorithms       | Accuracy |     Precision      |       Recall       |         F1         |
+------------------------+----------+--------------------+--------------------+--------------------+
|     Random forest      |  0.9025  | 0.9059479981592269 | 0.9064215334049734 | 0.9061113506097987 |
| NaiveBayes GaussianNB: |   0.78   | 0.788386855289761  | 0.7868991928318424 | 0.7872322037212488 |
|       SVM linear       |   0.78   | 0.788386855289761  | 0.7868991928318424 | 0.7872322037212488 |
|     Decision tree      |  0.8525  | 0.8547694753577106 | 0.855761458314326  | 0.8545941983348532 |
```

```
|  Logistic Regression   |  0.955   | 0.9566120479803631 | 0.9571131709298146 | 0.9566637790782159 |
|        kNN k=5         |  0.925   | 0.9258792765352952 | 0.9269778580893778 | 0.9261217472539143 |
+------------------------+----------+--------------------+--------------------+--------------------+
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of f AND g EVALUATIONS EXCEEDS LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Task 4.

For a given dataset (named **mushroom.csv**) with the following information:

- Attribute Information: (**classes**: edible=e, poisonous=p)
- **cap-shape**: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
- **cap-surface**: fibrous=f,grooves=g,scaly=y,smooth=s
- **cap-color**: brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y
- **bruises**: bruises=t,no=f
- **odor**: almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s
- **gill-attachment**: attached=a,descending=d,free=f,notched=n
- **gill-spacing**: close=c,crowded=w,distant=d
- **gill-size**: broad=b,narrow=n
- **gill-color**: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e,white=w,yellow=y
- **stalk-shape**: enlarging=e,tapering=t
- **stalk-root**: bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?
- **stalk-surface-above-ring**: fibrous=f,scaly=y,silky=k,smooth=s
- **stalk-surface-below-ring**: fibrous=f,scaly=y,silky=k,smooth=s
- **stalk-color-above-ring**: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
- **stalk-color-below-ring**: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
- **veil-type**: partial=p,universal=u
- **veil-color**: brown=n,orange=o,white=w,yellow=y
- **ring-number**: none=n,one=o,two=t
- **ring-type**: cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=z
- **spore-print-color**: black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y
- **population**: abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y
- **habitat**: grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d

Task 4.1. Apply approriate reprocessing techniques for the mushroom dataset

```
mushrom_data = pd.read_csv("mushrooms.csv")
columns = mushrom_data.drop(columns="class").columns
mushrom_data = myLabelEncoder(mushrom_data,columns)
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number
0	p	5	2	4	1	6	1	0	1	4	...	2	7	7	0	2	1
1	e	5	2	9	1	0	1	0	0	4	...	2	7	7	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Task 4.2. Apply classification algorithms (**Decision Tree, Logistic Regression, SVM, kNN, Random Forest, NaiveBayes**) to the reprocessed dataset

```
X = mushrom_data.drop(columns="class")
y = mushrom_data[["class"]]
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
print("without using selection feature ")
table = PrettyTable(["Algorithms","Accuracy","Precision","Recall","F1"])
for i in range(0,6):
  table.add_row(getScores(i,X_train,X_test,y_train,y_test))
print(table)
```

```
without using selection feature
+------------------------+--------------------+--------------------+--------------------+--------------------+
|       Algorithms       |      Accuracy      |     Precision      |       Recall       |         F1         |
+------------------------+--------------------+--------------------+--------------------+--------------------+
|     Random forest      |        1.0         |        1.0         |        1.0         |        1.0         |
| NaiveBayes GaussianNB: | 0.9163076923076923 | 0.9163798984309861 | 0.9161931430753425 | 0.9162642805396672 |
|       SVM linear       | 0.9163076923076923 | 0.9163798984309861 | 0.9161931430753425 | 0.9162642805396672 |
|     Decision tree      |        1.0         |        1.0         |        1.0         |        1.0         |
|  Logistic Regression   | 0.9452307692307692 | 0.9458507188018148 | 0.9449386768008292 | 0.9451702215824413 |
|        kNN k=5         | 0.9975384615384615 | 0.9975624503700651 | 0.9975157058304771 | 0.9975375653871099 |
+------------------------+--------------------+--------------------+--------------------+--------------------+
```

Start coding or generate with AI.

Task 4.3. Apply an feature selection method to dataset obtained in Task 4.1, then apply classification algorithm to the obtained dataset. Compare the performance of classification models before and after applying the feature selection technique in term of **accuracy, precision, recall, and F1** measures

```
newX = SelectKBest(chi2,k=13).fit_transform(X,y)
```