DATA MINING

Data Mining Assignments

اسم الطالب /

عبدالرحمن عبدالجليل عبدالرحمن

اشراف الدكتورة /

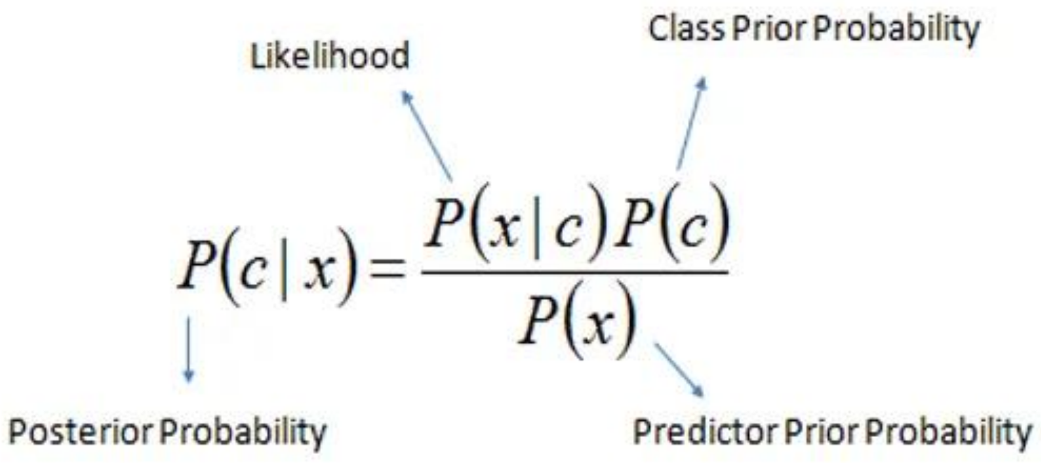
هبة المروعي

Naïve Bayes Classification in Python

Introduction

Naive Bayes is a classification algorithm that is based on Bayes' theorem. Bayes' theorem states that the probability of an event is equal to the prior probability of the event multiplied by the likelihood of the event given some evidence. In the context of classification, this means that we are trying to find the class that is most likely given a set of features or attributes.

Naive Bayes assumes that the features are independent of each other, meaning that the presence or absence of one feature does not affect the presence or absence of another feature. This simplifies the calculation of the likelihood of the features, as we can calculate the likelihood of each feature separately and then multiply them together.



$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Implement Naïve Bayes Classification in Python

In this example, we will use the social network ads data concerning the *Gender, Age, and Estimated Salary* of several users and based on these data we would classify each user whether they would purchase the insurance or not.

Step 1: Import libraries

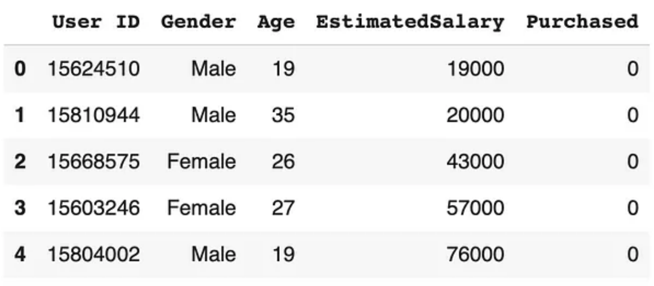
We need *Pandas* for data manipulation, *NumPy* for mathematical calculations, *MatplotLib, and Seaborn* for visualizations. *Sklearn* libraries are used for machine learning operations

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
```

Step 2: Import data

Download the dataset from here and upload it to your notebook and read it into the pandas dataframe.

```
# Read dataset
df_net = pd.read_csv('/content/Social_Network_Ads.csv')
df_net.head()
```



	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

Step 3: Data Analysis / Preprocessing

Exploratory Data Analysis (EDA) is a process of analyzing and summarizing the main characteristics of a dataset, with the goal of gaining insight into the underlying structure, relationships, and patterns within the data. EDA helps to identify important features, anomalies, and trends in the data that can inform further analysis and modeling.

EDA typically involves several key steps, including:

- **Data cleaning and preparation** involve removing missing or incorrect values, transforming variables, and handling outliers.
- **Data visualization** is the process of creating graphs, charts, and other visual representations of the data to help identify patterns, relationships, and anomalies.
- **Statistical analysis** involves applying mathematical and statistical methods to the data to identify important features and relationships.

Preprocessing aims to prepare the data in a way that will enable effective analysis and modeling and remove any biases or errors that may affect the results.

Get required data

We don’t need the *User ID* column so we can drop it.

```
# Get required data
df_net.drop(columns = ['User ID'], inplace=True)
df_net.head()
```

	Gender	Age	EstimatedSalary	Purchased
0	Male	19	19000	0
1	Male	35	20000	0
2	Female	26	43000	0
3	Female	27	57000	0
4	Male	19	76000	0

Describe data

Get statistical description of data using *Pandas* ***describe()*** function. It shows us the count, mean, standard deviation, and range of data.

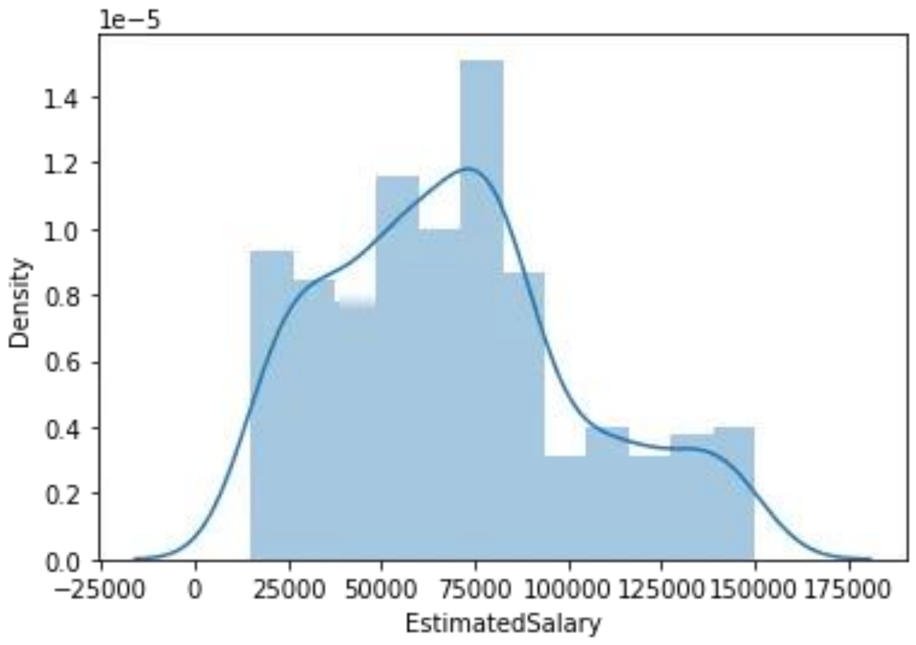
```
# Describe data
df_net.describe()
```

	Age	EstimatedSalary	Purchased
count	400.000000	400.000000	400.000000
mean	37.655000	69742.500000	0.357500
std	10.482877	34096.960282	0.479864
min	18.000000	15000.000000	0.000000
25%	29.750000	43000.000000	0.000000
50%	37.000000	70000.000000	0.000000
75%	46.000000	88000.000000	1.000000
max	60.000000	150000.000000	1.000000

Distribution of data

Check data distribution.

```
# Salary distribution
sns.distplot(df_net['EstimatedSalary'])
```



Label encoding

Label encoding is a preprocessing technique in machine learning and data analysis where categorical data is converted into numerical values, to make it compatible with mathematical operations and models.

The categorical data is assigned an integer value, typically starting from 0, and each unique category in the data is given a unique integer value so that the categorical data can be treated as numerical data.

```
# Label encoding
le = LabelEncoder()
df_net['Gender']= le.fit_transform(df_net['Gender'])
```

Correlation matrix

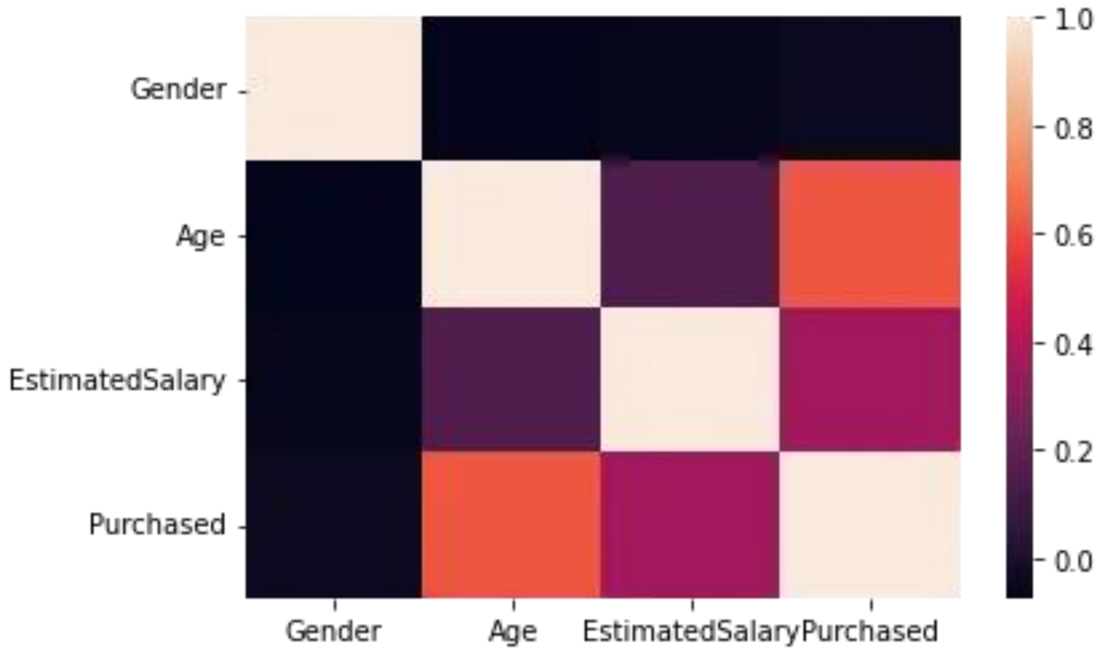
A ***correlation matrix*** is a table that summarizes the relationship between multiple variables in a dataset. It shows the correlation coefficients between each pair of variables, which indicate the strength and direction of the relationship between the variables. It is useful for identifying highly correlated variables and selecting a subset of variables for further analysis.

The correlation coefficient can range from -1 to 1, where:

- A ***correlation coefficient of -1*** indicates a strong negative relationship between two variables
- A ***correlation coefficient of 0*** indicates no relationship between two variables
- A ***correlation coefficient of 1*** indicates a strong positive relationship between two variables

```
# Correlation matrix
df_net.corr()
sns.heatmap(df_net.corr())
```

	Age	EstimatedSalary	Purchased
Age	1.000000	0.155238	0.622454
EstimatedSalary	0.155238	1.000000	0.362083
Purchased	0.622454	0.362083	1.000000



Drop insignificant data

From the correlation matrix, we see that *Gender* is not correlated to other attributes so we can drop that too.

```
# Drop Gender column
df_net.drop(columns=['Gender'], inplace=True)
```

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0

Step 4: Split data

Splitting data into independent and dependent variables involves separating the *input features (**independent variables**)* from the *target variable (**dependent variable**)*. The independent variables are used to predict the value of the dependent variable.

The data is then split into a training set and a test set, with the training set used to fit the model and the test set used to evaluate its performance.

Independent / Dependent variables

In our data *Age, EstimatedSalary* is the independent variable assigned as $\boldsymbol{X}$, and *Purchased* is the dependent variable $\boldsymbol{y}$.

```
# Split data into dependent/independent variables
X = df_net.iloc[:, :-1].values
y = df_net.iloc[:, -1].values
```

Train / Test split

The data is usually divided into two parts, with the majority of the data used for training the model and a smaller portion used for testing.

The training set is used to train the model and find the optimal parameters. The model is then tested on the test set to evaluate its performance and determine its accuracy. This is important because if the model is trained and tested on the same data, it may over-fit the data and perform poorly on new, unseen data.

We have split the data into 75% for training and 25% for testing.

```
# Split data into test/train set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.25, random_state = True)
```

Step 5: Feature scaling

Feature scaling is a method of transforming the values of numeric variables so that they have a common scale as machine learning algorithms are sensitive to the scale of the input features.

There are two common methods of feature scaling: *normalization* and *standardization*.

- **Normalization** scales the values of the variables so that they fall between 0 and 1. This is done by subtracting the minimum value of the feature and dividing it by the range (max-min).
- **Standardization** transforms the values of the variables so that they have a mean of 0 and a standard deviation of 1. This is done by subtracting the mean and dividing it by the standard deviation.

Feature scaling is usually performed before training a model, as it can improve the performance of the model and reduce the time required to train it, and helps to ensure that the algorithm is not biased towards variables with larger values.

```
# Scale dataset
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Step 6: Train model

Training a machine learning model involves using a training dataset to estimate the parameters of the model. The training process uses a learning algorithm that iteratively updates the model parameters, minimizes a loss function, which measures the difference between the predicted values and the actual values in the training data, and updates the model parameters to improve the accuracy of the model.

It's important to note that the SVM algorithm requires feature scaling and proper choice of kernel functions and regularization parameters to produce accurate predictions.

Pass the *X_train* and *y_train* data into the *Naïve Bayes* classifier model by ***classifier.fit*** to train the model with our training data.

```
# Classifier
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

Step 7: Predict result / Score model

Once the likelihood of the features for each class is calculated, the algorithm multiplies the likelihood by the prior probability of each class, which is estimated from the training data. The class with the highest probability is then selected as the predicted class.

The accuracy of the model can be evaluated on a test set, which was previously held out from the training process.

```
# Prediction
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred), 1),
y_test.reshape(len(y_test), 1)), 1))
```

Step 8: Evaluate model

Accuracy is a useful metric for assessing the performance of a model, but it can be misleading in some cases. For example, in a highly imbalanced dataset, a model that always predicts the majority class will have high accuracy, even though it may not be performing well. Therefore, it is important to consider other metrics, such as confusion matrix, precision, recall, F1-score, and ROC-AUC, along with accuracy, to get a more complete picture of the performance of a model.

Accuracy

Accuracy is a commonly used metric for evaluating the performance of a machine learning model. It measures the proportion of correct predictions made by the model on a given dataset.

In a binary classification problem, accuracy is defined as the number of correct predictions divided by the total number of predictions. In a multi-class classification problem, accuracy is the average of the individual class accuracy scores.

```
# Accuracy
accuracy_score(y_test, y_pred)
```

```
0.86
```

Classification report

A classification report is a summary of the performance of a classification model. It provides several metrics for evaluating the performance of the model on a classification task, including precision, recall, f1-score, and support.

The classification report also provides a weighted average of the individual class scores, which takes into account the imbalance in the distribution of classes in the dataset.

```
# Classification report
print(f'Classification Report: \n{classification_report(y_test, y_pred)}')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.88      0.88        58
           1       0.83      0.83      0.83        42

    accuracy                           0.86       100
   macro avg       0.86      0.86      0.86       100
weighted avg       0.86      0.86      0.86       100
```

F1 score

F1-score is the harmonic mean of precision and recall. It provides a single score that balances precision and recall. Support is the number of instances of each class in the evaluation dataset.

```
# F1 score
print(f"F1 Score : {f1_score(y_test, y_pred)}")
```

```
F1 Score : 0.8333333333333334
```

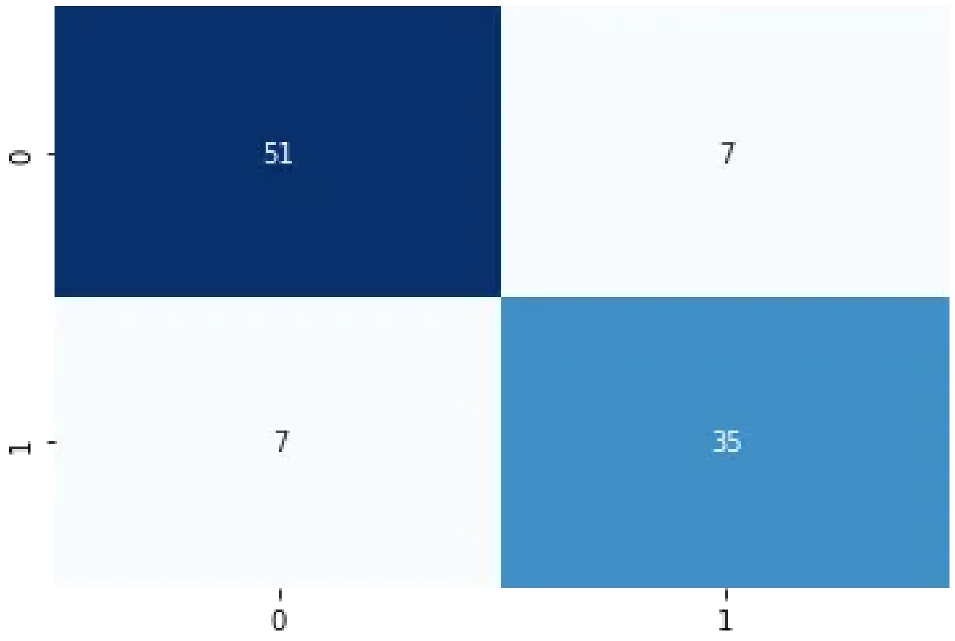
Confusion matrix

A confusion matrix is used to evaluate the performance of a classification model. It summarizes the model's performance by comparing the actual class labels of the data to the predicted class labels generated by the model.

True Positives (TP): Correctly predicted positive instances.
False Positives (FP): Incorrectly predicted positive instances.
True Negatives (TN): Correctly predicted negative instances.
False Negatives (FN): Incorrectly predicted negative instances.

It provides a clear and detailed understanding of how well the model is performing and helps to identify areas of improvement.

```
# Confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
```



Precision-Recall curve

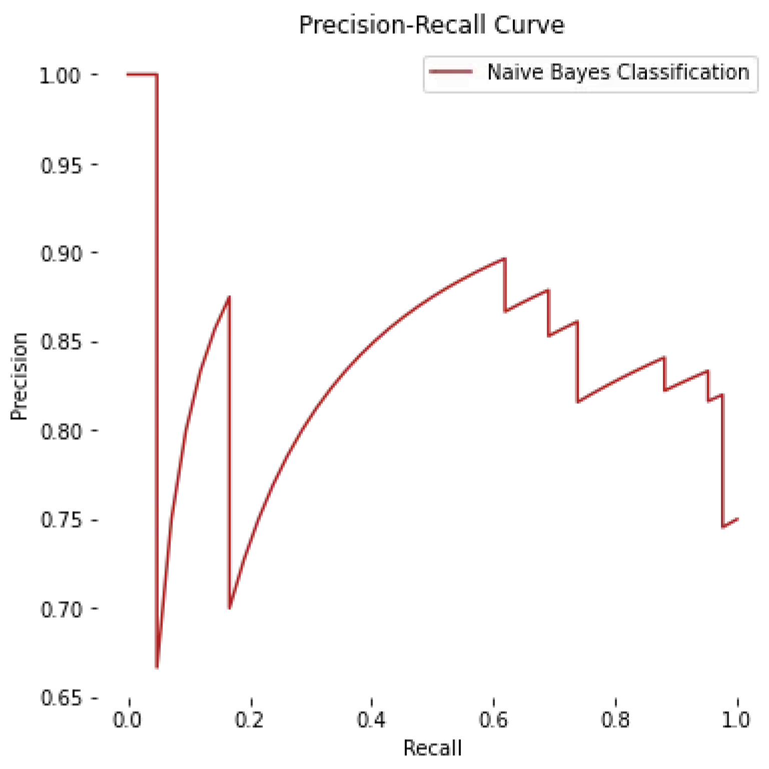
A precision-recall curve is a plot that summarizes the performance of a binary classification model as a trade-off between precision and recall and is useful for evaluating the model’s ability to make accurate positive predictions while finding as many positive instances as possible. *Precision* and *Recall* are two common metrics for evaluating the performance of a classification model.

Precision is the number of *true positive* predictions divided by the sum of *true positive* and *false positive* predictions. It measures the accuracy of the positive predictions made by the model.

Recall is the number of *true positive* predictions divided by the sum of true positive and *false negative* predictions. It measures the ability of the model to find all positive instances.

```
# Plot Precision-Recall Curve
y_pred_proba = classifier.predict_proba(X_test)[:,1]
precision, recall, thresholds = precision_recall_curve(y_test,
y_pred_proba)

fig, ax = plt.subplots(figsize=(6,6))
ax.plot(recall, precision, label='Naive Bayes Classification', color =
'firebrick')
ax.set_title('Precision-Recall Curve')
ax.set_xlabel('Recall')
ax.set_ylabel('Precision')
plt.box(False)
ax.legend();
```


Precision-Recall Curve
Naive Bayes Classification
Precision
Recall