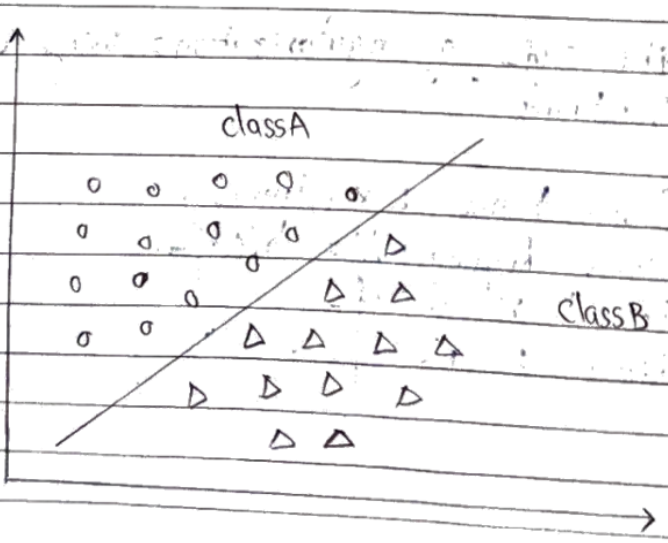
5. Classification in ML

Date
Page 108

1. Classification Algorithm

- The classification algorithm is used to identify the category of new observations on the basis of training data.
- In classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups.
- Such as, Yes or No, 0 or 1, Spam or Not spam, cat or dog, etc. Classes can be called as targets/ labels or categories.



Types of Classifications

1. **Binary Classifier :** If the classification problem has only two possible outcomes, then it is called as Binary Classifier.
   Example: SPAM or NOT SPAM, CAT or DOG, etc.

2. Multi-class Classifier: If a classification problem has more than two outcomes, then it is called as Multi-class Classifier.
Example: Classification of types of crops, Classification of types of music, etc.

Types of ML Classification Algorithms

Non-linear Models
- K-Nearest Neighbours
- SVM (Support Vector Machines)
- Naive Bayes
- Decision Tree Classification
- Random forest Classification

Linear Models
- Logistic Regression
- Support Vector Machines

2. Evaluating a Classification Model

1. Log Loss or Cross-Entropy Loss
2. Confusion Matrix
3. AUC - ROC curve.

Real World Examples of Classification Problems:

- Disguishing the spam mails and non-spam mails.
- Classification of documents.
- Distinguish between animals.

3. Logistic Regression

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique.
- It is used for predicting the categorical dependent variable using a given set of independent variables.
- Therefore, the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, True or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.

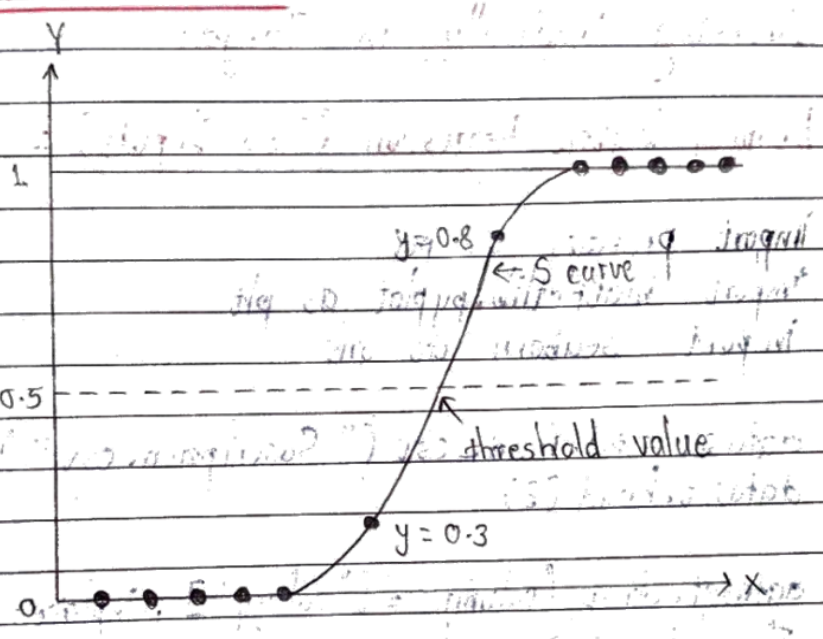
Types of Logistic Regression

On the basis of categories, Logistic Regression can be classified into three types:

1. **Binomial**: In binomial logistic regression, there can be only two possible types of dependent variables, such as 0 or 1, Pass or Fail, etc.
2. **Multinomial**: In multinomial Logistic regression, there can three or more possible unordered types of the dependent variable, such as "cat", "dog", or "sheep"
3. **Ordinal**: In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent

variables, such as "low", "Medium", "High".

Sigmoid Function



The logistic regression equation can be obtained from the linear Regression equation.
Mathematically, Logistic regression equation is:

$$y = \frac{1}{1+e^{-x}}$$

where,

y = dependent variable
x = independent variable
e = euler's constant

Uses of Logistic Regression

i. Spam email filtering

ii. Object classification
iii. Credit card fraud detection.

Working Practically in Jupyter

Binomial Logistic Regression (Two Inputs) :

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

dataset.drop(columns=["Salary"], inplace=True)
# Delete salary column.
dataset.head(3)

# Visualize the data
plt.figure(figsize=(4,3))
sns.scatterplot(x="Age", y="Purchase", data=dataset)
plt.show()

# Select dependent and independent variables.
x = dataset[["Age"]]
y = dataset["Purchase"]

# Train test split
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x,
        y, test_size=0.2, random_state=42)

# Implement model
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(x_train, y_train)  # Train model

lr.score(x_test, y_test)  # Test model

lr.predict([[40]])  # New prediction

# Visualize prediction line.
plt.figure(figsize=(9,3))
sns.scatterplot(x="Age", y="Purchase", data=dataset)
sns.lineplot(x="Age", y=lr.predict(x), data=dataset,
             color="red")
plt.show()
```

Binomial Logistic Regression (Multiple Inputs):

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

# Graphical visualization
plt.figure(figsize=(5,4))
```

```
sns.scatterplot(x="Age", y="Salary", data=dataset,
                hue="Purchase")
plt.legend(loc=2)
plt.show()

# Dependent and Independent data.
x = dataset.iloc[:, :-1]
y = dataset["Purchase"]

# Train test split.
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(
        x, y, test_size=0.2, random_state=42)

# Train your model.
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(x_train, y_train)

lr.score(x_test, y_test)

lr.predict([[36, 35000]])

# Visualization of prediction line
from mlxtend.plotting import plot_decision_regions

plot_decision_regions(x.to_numpy(), y.to_numpy(),
                      clf=lr)
plt.show()
```

Coefficients of logistic regression equation
lr.coef_

Constant of logistic regression equation
lr.intercept_

Binomial Logistic Regression (Using PolynomialFeatures):

- This trains model on polynomial equation instead of linear.
- This is used for better accuracy of model.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

sns.scatterplot(x="Age", y="Salary", data=dataset,
                hue="Purchase")
plt.show()

# Seperate dependent and independent datas.
x = dataset.iloc[:, :-1]
y = dataset["Purchase"]

# Polynomialize the independent data
from sklearn.preprocessing import PolynomialFeatures
```

```
pf = PolynomialFeatures (degree = 2)
pf. fit (x)
x = pd. DataFrame (pf. transform (x))
x. head (3)

# Train test split
from sklearn. model_selection import train_test_split

x_train , x_test , y_train , y_test = train_test_split (x,
        y, test_size = 0.2 , random_state = 42 )

# Build the polynomial model.
from sklearn. linear_model import LogisticRegression

lr = LogisticRegression ()
lr. fit (x_train , y_train)

lr. score (x_test , y_test)
```

Multinomial Logistic Regression :

```
import pandas as pd
import matplotlib. pyplot as plt
import seaborn as sns

dataset = pd. read_csv ("Irisflower. csv")
dataset. head (3)

dataset ["Species"]. unique() # Check for unique values
```

```
# Visualizing dataset
sns.pairplot(data=dataset, hue="Species")
plt.show()

# Seperate dependent and independent data
x = dataset.iloc[:, :-1]
y = dataset["Species"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,
        y, test_size=0.2, random_state=42)

# Build model
from sklearn.linear_model import LogisticRegression

""" OVR Method """

lr = LogisticRegression(multi_class="ovr")
lr.fit(x_train, y_train)

lr.score(x_test, y_test)

""" Using Multinomial """

lr = LogisticRegression(multi_class="multinomial")
lr.fit(x_train, y_train)

lr.score(x_test, y_test)
```

4. Confusion Matrix

- A confusion matrix is a simple and useful tool for understanding the performance of a classification model, like one used in machine learning or statistics.
- It helps you evaluate how well your model is doing in categorizing things correctly.
- It is also known as the error matrix.
- The matrix consists of prediction result in a summarized form, which has a total number of correct predictions and incorrect predictions.

Sr. No.	Actual	Predicted
1	1	1
2	0	0
3	1	0
4	0	1
5	0	0
6	0	1
7	1	1

Now, the confusion matrix for the data in the table can be created as follows:

Date
Page 11.9

Confusion Matrix

Actual Labels	0	1
0	TN = 86	FP = 2
1	FN = 1	TP = 2

Predicted Labels

TN → true negative
FP → false positive
FN → false negative
TP → true positive

- Mathematically,
  - Accuracy = $\frac{TN + TP}{TN + TP + FP + FN}$

  i.e. Accuracy = $\frac{TN + TP}{N}$; N = total number of data.

  - Error = $\frac{FN + FP}{N}$

- **False Negative:** The model has predicted No, but the actual value was Yes.
  It is also called as Type-II error.

- **False Positive:** The model has predicted Yes, but the actual value was No.
  It is also called Type-I error.

Precision

- It helps us to measure the ability to classify positive samples in model.
- Mathematically,

$$\text{Precision} = \frac{TP}{TP + FP}$$

- Example:

A

	Detected Cancer	Not Detected Cancer
Has Cancer	1000	200
No cancer	800	8000

B

	Detected Cancer	Not Detected Cancer
Has Cancer	1000	500
No Cancer	500	8000

Here, we find that has cancer but not detected is a major issue which is type I error (FP error). To minimize FP error we try to keep the value of precision high by minimizing value of FP.

In this example model B will be selected though having same accuracy as value of FP is less in model B.

Remember that to minimize Type I error, we should always try to increase value of precision. The max value of precision is 1 when FP = 0.

Date Page 121

Recall :

- It helps us to measure how many positive samples were correctly classified by the ML model.
- Mathematically,

$$\text{Recall} = \frac{TP}{TP + FN}$$

- Example:

A

	Sent to spam	Not sent to spam
Spam	100	170
Not spam	30	700

B

	Sent to Spam	Not sent to spam
Spam	100	190
Not spam	10	700

Here, we find that not spam mails sent to spam is a major issue which is type II error (FN error). To minimize FN error we try to keep the value of recall high by minimizing value of FN.

In this example model B will be selected though having same accuracy as value of FN is less in model B.

Remember that to minimize type II error, we should always try to increase value of recall. The max value of recall is 1 when FN = 0.

F1 - Score

- It is harmonic mean of precision and recall. It takes both false positive and false negative into account. Therefore, it performs well on an imbalanced dataset.

- Mathematically,

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Working Practically

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

# Seperate dependent and independent variables
x = dataset.iloc[:, :-1]
y = dataset["Purchase"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,
    y, test_size = 0.2, random_state = 42)
```

Date
Page 123

```
# Build your model
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(x_train, y_train)

lr.score(x_test, y_test)

# Create a confusion matrix
from sklearn.metrics import confusion_matrix, precision_score,
        recall_score, f1_score

# You need training data and predicted data to make a
# confusion matrix
cf_mat = confusion_matrix(y_test, lr.predict(x_test))
cf_mat

# For graphical visualization
sns.heatmap(cf_mat, annot=True)
plt.show()

# Precision
precision_score(y_test, lr.predict(x_test))

# Recall
recall_score(y_test, lr.predict(x_test))

# F1 score
f1_score(y_test, lr.predict(x_test))
```

5. Imbalanced Dataset Handling

1. Random Under Sampling

- We will reduce the majority of the class so that it will have same no of as the minority.
- Random samples from majority class is taken.

2. Random Over Sampling

- We will increase the size of minority i.e. inactive class to size of majority class i.e. active class.
- Copies of minority class is formed.

Working Practically

```
import pandas as pd

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

# Check if the data is imbalanced.
dataset["Purchase"].value_counts()
# The dataset is unbalanced.

''' Imbalanced dataset leads to biasing of the machine learning module. So, we need to balance the dataset for avoiding this problem. '''
```

```
# Seperate dependent and independent variables
x = dataset.iloc[:, :-1]
y = dataset["Purchase"]
```

Random Under Sampling:

```
# Balance the dataset using random under sampling.
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler()
rus_x, rus_y = rus.fit_resample(x, y)

rus_y.value_counts() # Data is now balanced.

# Train Test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(rus_x,
    rus_y, test_size = 0.2, random_state = 42)

# Build the model
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(x_train, y_train)

lr.score(x_test, y_test)
```

Random Over Sampling:

```
# Balance the dataset using random over sampling.
```

```
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler()
ros_x, ros_y = ros.fit_resample(x, y)

ros_y.value_counts()  # Data is now balanced

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(ros_x,
        ros_y, test_size = 0.2, random_state = 42)

# Build the model
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(x_train, y_train)

lr.score(x_test, y_test)
```

6. Naive Bayes Algorithm

- Naive Bayes is a classification algorithm based on Baye's theorem.

- **Bayes' Theorem :** is a probability theory that describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

Date
Page 127

- **Naive :** It is called Naive because it assumes that the occurrence of a certain feature is independent of the occurrence of other features.

- **Bayes :** It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem

- Bayes' theorem is also known as Bayes' rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge.

- It depends on the conditional probability.

$$P(A/B) = \frac{P(B/A)\ P(A)}{P(B)}$$

Where,

P(A/B) is Posterior probability : Probability of hypothesis A on the observed event B.

P(B/A) is Likelihood probability : Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior probability : Probability of hypothesis before observing the evidence.

P(B) is Marginal probability : Probability of evidence

Types of Naive Bayes Model

There are three types of Navie Bayes Model:

1. Gaussian
2. Multinomial
3. Bernoulli

1. Gausslan Naive Bayes :

- Assume that continous features follow a Gaussian (normal) distribution.
- Suitable for features that are continous and have a normal distribution.

2. Multinomial Naive Bayes :

- Assumes that features follow a multinomial distribution.
- Typically used for discrete data, such as text data, where each feature represents the frequency of a term.

3. Bernoulli Naive Bayes :

- Assumes that features are binary (boolean) variables.
- Suitable for data that can be represented as binary features, such as document classification problems where each term is either present or absent.

Working Practically:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from mlxtend.plotting import plot_decision_regions

dataset = pd.read_csv("Suscription.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum()  # No null values

# Visualize data.
plt.figure(figsize = (5,5))
sns.scatterplot(x = "Age", y = "Salary", data = dataset,
                hue = "Purchase")
plt.show()

# Seperate dependent and independent variables
x = dataset.iloc[:, :-1]
y = dataset["Purchase"]

# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
        test_size = 0.2, random_state = 42)
```

Gaussian Naive Bayes:

```
# Build model
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
gnb.fit(x_train, y_train)

# Visualize the decision region
plt.figure(figsize=(5,5))
plot_decision_regions(x.to_numpy(), y.to_numpy(),
                      clf=gnb)
plt.show()

gnb.score(x_test, y_test)
```

Multinomial Naive Bayes:

```
# Build model
from sklearn.naive_bayes import MultinomialNB

mnb = MultinomialNB()
mnb.fit(x_train, y_train)

mnb.score(x_test, y_test)

# Visualize the decision region
plt.figure(figsize=(5,5))
plot_decision_regions(x.to_numpy(), y.to_numpy(),
                      clf=mnb)
plt.show()
```

Bernoulli Naive Bayes:

```
# Build model
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB()
bnb.fit(x_train, y_train)

bnb.score(x_test, y_test)

# Visualize the decision region
plt.figure(figsize=(5,5))
plot_decision_regions(x.to_numpy(), y.to_numpy(),
                      clf=bnb)
plt.show()
```