Unit 2 Platform for Machine Learning

OUTLINE:

- ❖Platform for Machine Learning

PLATFORM FOR MACHINE LEARNING

- **Jupyter Notebook:** Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It's widely used in the data science and machine learning community and is great for interactive coding and exploration.
- **Google Colab:** Google Colab is a cloud-based version of Jupyter Notebook, provided by Google. It allows you to run Python code directly on their servers, leveraging their GPUs and TPUs for free. It's an excellent choice if you need access to powerful hardware without setting up anything locally.
- **Notion:** Notion is an all-in-one workspace that allows you to take notes, organize information, and collaborate with others. It supports a wide range of media, including text, images, code snippets, and more. You can create different databases for organizing your machine learning notes efficiently.

PLATFORM FOR MACHINE LEARNING

- **Evernote:** Evernote is a popular note-taking application that supports multimedia content, such as text, images, and web clippings. It has excellent search and organization features, making it suitable for keeping track of machine learning resources and ideas.
- **OneNote:** OneNote is another comprehensive note-taking application developed by Microsoft. It allows you to create notebooks, sections, and pages, making it suitable for organizing machine learning notes and other content.
- **GitHub:** While GitHub is primarily a code repository platform, it can be utilized for taking machine learning notes as well. You can create a private repository and use Markdown files to document your learning progress, projects, and research.

PLATFORM FOR MACHINE LEARNING

Obsidian: Obsidian is a note-taking app that supports linking notes together, creating a network of interconnected ideas. This can be useful for organizing and understanding complex machine learning topics.

TiddlyWiki: TiddlyWiki is a unique note-taking platform that runs in a single HTML file. It allows you to create non-linear, interconnected notes. You can customize it to your needs and use it for machine learning documentation.

The best platform for you depends on your preferences and workflow. If you prefer interactive coding and experimentation, **Jupyter Notebook or Google Colab may be ideal**. If you need a more comprehensive workspace with multimedia support, **Notion or Evernote** might be better options. Experiment with a few platforms to find the one that suits your needs best.

PLATFORM FOR MACHINE LEARNING

Obsidian: Obsidian is a note-taking app that supports linking notes together, creating a network of interconnected ideas. This can be useful for organizing and understanding complex machine learning topics.

TiddlyWiki: TiddlyWiki is a unique note-taking platform that runs in a single HTML file. It allows you to create non-linear, interconnected notes. You can customize it to your needs and use it for machine learning documentation.

The best platform for you depends on your preferences and workflow. If you prefer interactive coding and experimentation, **Jupyter Notebook or Google Colab may be ideal**. If you need a more comprehensive workspace with multimedia support, **Notion or Evernote** might be better options. Experiment with a few platforms to find the one that suits your needs best.

PLATFORM FOR MACHINE LEARNING

1. Hosting and Environment:

- **Jupyter Notebook:** Jupyter Notebook is a local application that you install on your computer. It requires setting up a Python environment with the necessary libraries and dependencies on your machine.
- **Google Colab:** Colab is hosted on Google's servers, and it runs in the cloud. You don't need to install anything locally, and it comes with pre-installed libraries, including popular machine learning frameworks like TensorFlow and PyTorch.

2. Hardware Resources:

- **Jupyter Notebook:** The performance of Jupyter Notebook depends on the capabilities of your local machine, which may not always have access to powerful GPUs or TPUs.
- **Google Colab:** Colab offers free access to GPUs and TPUs provided by Google, making it an attractive option for computationally intensive tasks in machine learning.

PARAMTRIC & NON PARAMETRIC MODELS IN MACHINE LEARNING

3. Collaboration and Sharing:

- **Jupyter Notebook:** Collaboration and sharing can be more challenging with Jupyter Notebook, as it requires additional setup and configurations for sharing the notebooks with others.
- **Google Colab:** Colab makes collaboration and sharing much easier. You can share the Colab notebook with others just by sending them the link, and multiple users can work on the same notebook simultaneously.

4. Data and File Management:

- **Jupyter Notebook:** In a local Jupyter environment, you are responsible for managing data and files on your machine.
- **Google Colab:** Colab allows you to upload data from your local machine or directly from Google Drive. It simplifies data management as you can easily access and use data stored in your Google Drive.

PLATFORM FOR MACHINE LEARNING

5. Session Duration:

- **Jupyter Notebook:** In a local environment, your Jupyter session is active as long as your computer is running, and you can close and reopen it at will.
- **Google Colab:** Colab sessions have a limited duration of runtime (usually 12 hours). After this duration, the runtime restarts, and you may lose unsaved data.

6. **Integration with Google Services:**

Google Colab: Colab integrates well with other Google services like Google Drive, allowing seamless access to your files and datasets stored there.

7. Security and Privacy:

- **Jupyter Notebook:** In a local environment, you have more control over the security and privacy of your data.
- **Google Colab:** With Colab running in the cloud, you need to consider the privacy and security implications of running your code and data on Google's servers.

Unit 2 Machine Learning Framework

OUTLINE:

- Machine Learning Framework

MACHINE LEARNING LIBRARY

TensorFlow: Developed by Google, TensorFlow is one of the most widely used open-source machine learning frameworks. It provides extensive support for deep learning and neural networks, along with a flexible ecosystem for building and deploying machine learning models on various platforms.

PyTorch: Developed by Facebook's AI Research lab (FAIR), PyTorch is another popular open-source deep learning framework. It is known for its user-friendly and dynamic computation capabilities, making it easier to work with than some other frameworks.

scikit-learn: scikit-learn is a powerful and easy-to-use machine learning library in Python. It provides a wide range of classical machine learning algorithms, such as support vector machines, random forests, and k-nearest neighbors, along with tools for data preprocessing and evaluation.

Keras: Originally developed as a high-level API for TensorFlow, Keras has become a standalone deep learning framework that is now part of TensorFlow's official API. It offers a user-friendly interface for building and training neural networks.

MACHINE LEARNING LIBRARY

MXNet: Developed by Apache, MXNet is an open-source deep learning framework that supports both imperative and symbolic programming. It is known for its efficient execution on multi-GPU and distributed systems.

Caffe: Caffe is a deep learning framework developed by Berkeley Vision and Learning Center (BVLC). It is popular for its speed and efficiency in training convolutional neural networks (CNNs).

Microsoft Cognitive Toolkit (CNTK): CNTK is Microsoft's deep learning framework, designed for efficient training and evaluation of deep learning models. It is known for its performance and scalability.

MACHINE LEARNING LIBRARY

Scikit-learn (also known as sklearn) is a popular open-source machine learning library in Python. It is built on top of NumPy, SciPy, and matplotlib and provides a wide range of machine learning algorithms and tools for data preprocessing, feature engineering, model selection, and evaluation. Scikit-learn is widely used by data scientists and machine learning practitioners for building predictive models and performing data analysis tasks.

- **Wide Range of Algorithms:** Scikit-learn provides implementations of various supervised and unsupervised learning algorithms, including linear regression, logistic regression, decision trees, random forests, support vector machines, k-nearest neighbors, k-means clustering, and more.
- **Consistent API:** The library offers a consistent API for all algorithms, making it easy to switch between different models without having to change the code significantly.
- **Data Preprocessing:** Scikit-learn includes utilities for data preprocessing, such as feature scaling, one-hot encoding, handling missing values, and more.

MACHINE LEARNING LIBRARY

- **Model Evaluation:** The library provides functions for evaluating model performance using metrics like accuracy, precision, recall, F1-score, and more. It also supports cross-validation for robust evaluation.
- **Hyperparameter Tuning:** Scikit-learn offers tools for hyperparameter tuning, such as GridSearchCV and RandomizedSearchCV, to find the best combination of hyperparameters for a model.
- **Pipeline Creation:** You can create complex machine learning pipelines using scikit-learn's Pipeline class, allowing you to chain multiple data processing steps and models together.
- **Model Persistence:** Scikit-learn enables you to save and load trained models, so you can use them later for predictions without retraining.

MACHINE LEARNING LIBRARY

- **NumPy** (multi-dimensional arrays)
- **SciPy** (algorithms to use with numpy)
- **HDF5** (store & manipulate data)
- **Matplotlib** (data visualization)
- **Jupyter** (research collaboration)
- **PyTables** (managing HDF5 datasets)
- **HDFS** (C/C++ wrapper for Hadoop)
- **pymongo** (MongoDB driver)
- **SQLAlchemy** (Python SQL Toolkit)
- **redis** (Redis access libraries)
- **pyMySQL** (MySQL connector)
- **scikit-learn** (machine learning)
- **TensorFlow** (deep learning with neural networks)
- **scikit-learn** (machine learning algorithms)
- **keras** (high-level neural networks API)

Unit 1 Features in Machine Learning

OUTLINE:

- ❖Features in Machine Learning

Features in Machine Learning

Feature Extraction: Feature extraction is the process of **selecting or transforming the raw data into a set of relevant features** that can be used by machine learning algorithms. This process helps to reduce the dimensionality of the data and focus on the most important information.

Feature Engineering: Feature engineering involves creating new features from the existing ones or combining them in a meaningful way to enhance the performance of the machine learning model. This can involve mathematical transformations, grouping, scaling, and more.

Feature Selection: Feature selection **is the process of choosing the most relevant subset of features from the original set.** This helps to improve the model's efficiency and reduce overfitting by eliminating redundant or irrelevant features.

Features in Machine Learning

Categorical and Numerical Features: Features can be categorized as either categorical (qualitative) or numerical (quantitative). Categorical features represent discrete classes or labels, while numerical features represent measurable quantities.

One-Hot Encoding: Categorical features are often converted into numerical format through techniques like one-hot encoding, where each category is represented by a binary vector.

Feature Scaling: Numerical features are often scaled to a similar range to ensure that no single feature dominates the learning process. Common scaling techniques include normalization and standardization.

Feature Importance: Some machine learning algorithms, such as decision trees and ensemble methods, can provide information about the importance of each feature in making predictions. This helps in understanding which features contribute the most to the model's performance.

Features in Machine Learning

Text and Image Features: In natural language processing (NLP), features can be derived from text data using techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings. In computer vision, features can be extracted from images using methods like convolutional neural networks (CNNs).

Unit 2 Performance Metrices in Machine Learning

OUTLINE:

- ❖Performance Metrics in Machine Learning

Performance Metrices in Machine Learning

Classification Metrics:

- **Accuracy:** Accuracy score in machine learning is **an evaluation metric that measures the number of correct predictions** made by a model in relation to the total number of predictions made. We calculate it by dividing the number of correct.
- **Precision:** The ability of a classification model to identify only the relevant data points. Precision measures **the accuracy of positive predictions**.
- **Recall:** Also known as Sensitivity, it measures the proportion of true positive predictions among all actual positive instances. **Out of all the positive examples, how many are predicted as positive**
- **F1 Score:** The harmonic mean of precision and recall. It combines the precision and recall scores of a model. The accuracy metric computes how many times a model made a correct prediction across the entire dataset.

Performance Metrices in Machine Learning

- **ROC Curve (Receiver Operating Characteristic Curve):** A graphical representation of the trade-off between true positive rate and false positive rate at various thresholds. The area under the ROC curve (AUC-ROC) is a common metric for classification model comparison.
- **Confusion Matrix:** A table that summarizes the classification results showing true positives, true negatives, false positives, and false negatives.

Performance Metrices in Machine Learning

Regression Metrics:

- **Mean Squared Error (MSE):** The average of the squared differences between predicted and actual values. It penalizes larger errors.
- **Root Mean Squared Error (RMSE):** The square root of the MSE, providing a measure in the same units as the target variable.

Unit 2 Linear Regression with Multiple Variables

OUTLINE:

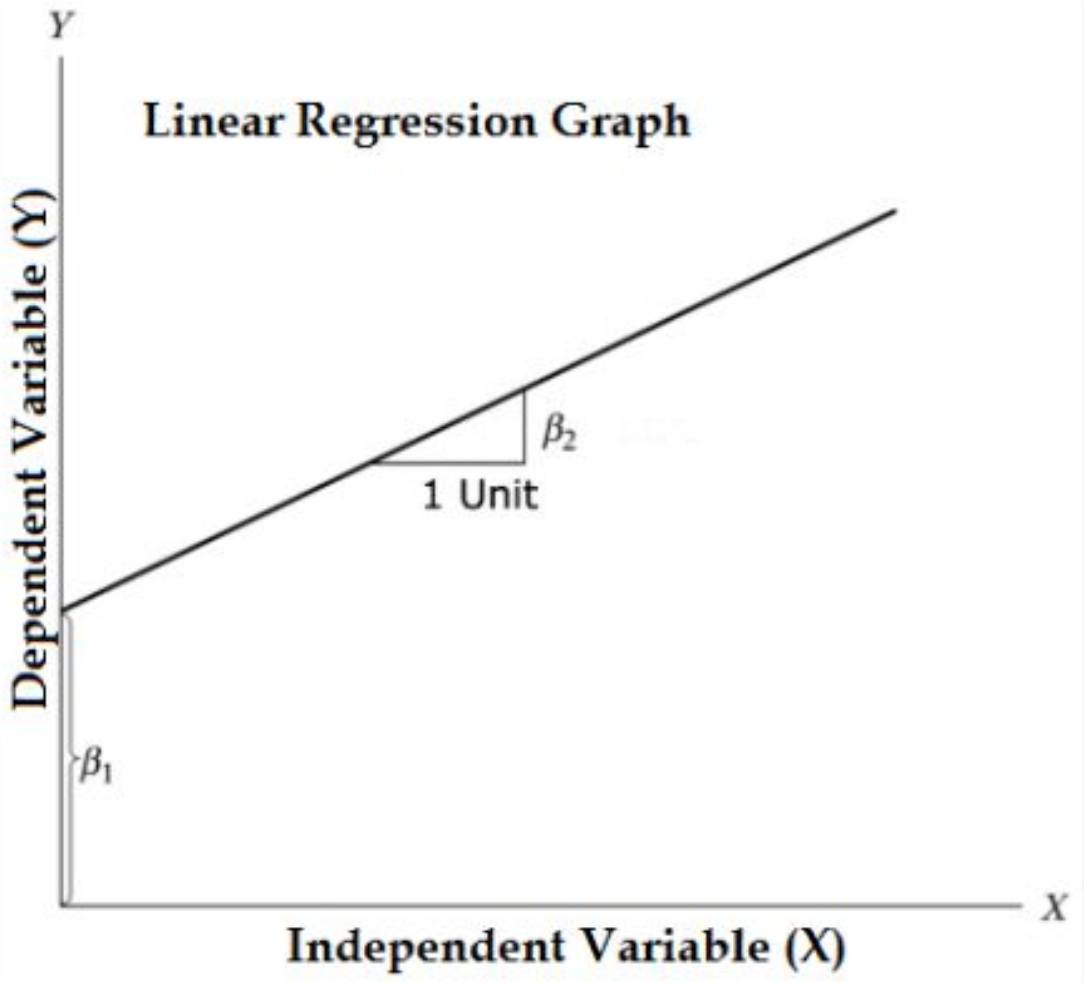
- Linear Regression with Multiple Variables

Linear Regression

- **Simple Linear regression** is a type of supervised machine learning algorithm that computes **the linear relationship between a dependent variable and one independent features**.
- When the number of the independent feature, is 1 then it is known as Univariate Linear regression, and in the case of more than one feature, it is known as multivariate linear regression.
- The goal of the algorithm is to find the best linear equation that can **predict the value of the dependent variable** based on the **independent variables**.
- The equation provides a straight line that represents the relationship between the dependent and independent variables.
- The slope of the line indicates how much the dependent variable changes for a unit change in the independent variable(s).

- **Multiple Linear Regression** is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.

Linear Regression



Linear Regression with Single Variables

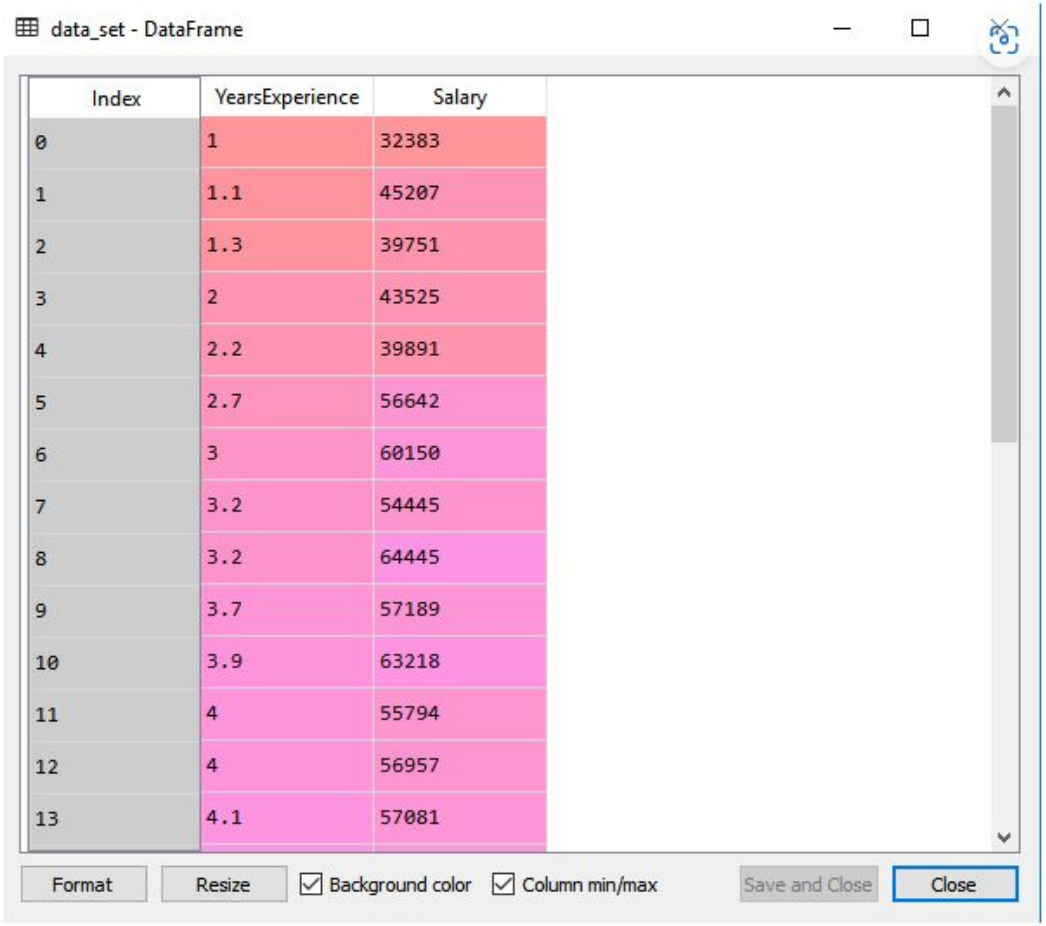
- The goals of this problem is:
- **We want to find out if there is any correlation between these two variables**
- **We will find the best fit line for the dataset.**
- **How the dependent variable is changing by changing the independent variable.**

- Simple Linear Regression model to find out the best fitting line for representing the relationship between these two variables.

Linear Regression with Single Variables

Step-1: Data Pre-processing

```
import numpy as nm
import matplotlib.pyplot as
mtp
import pandas as pd
data_set= pd.read_csv('Salar
_Data.csv')
```

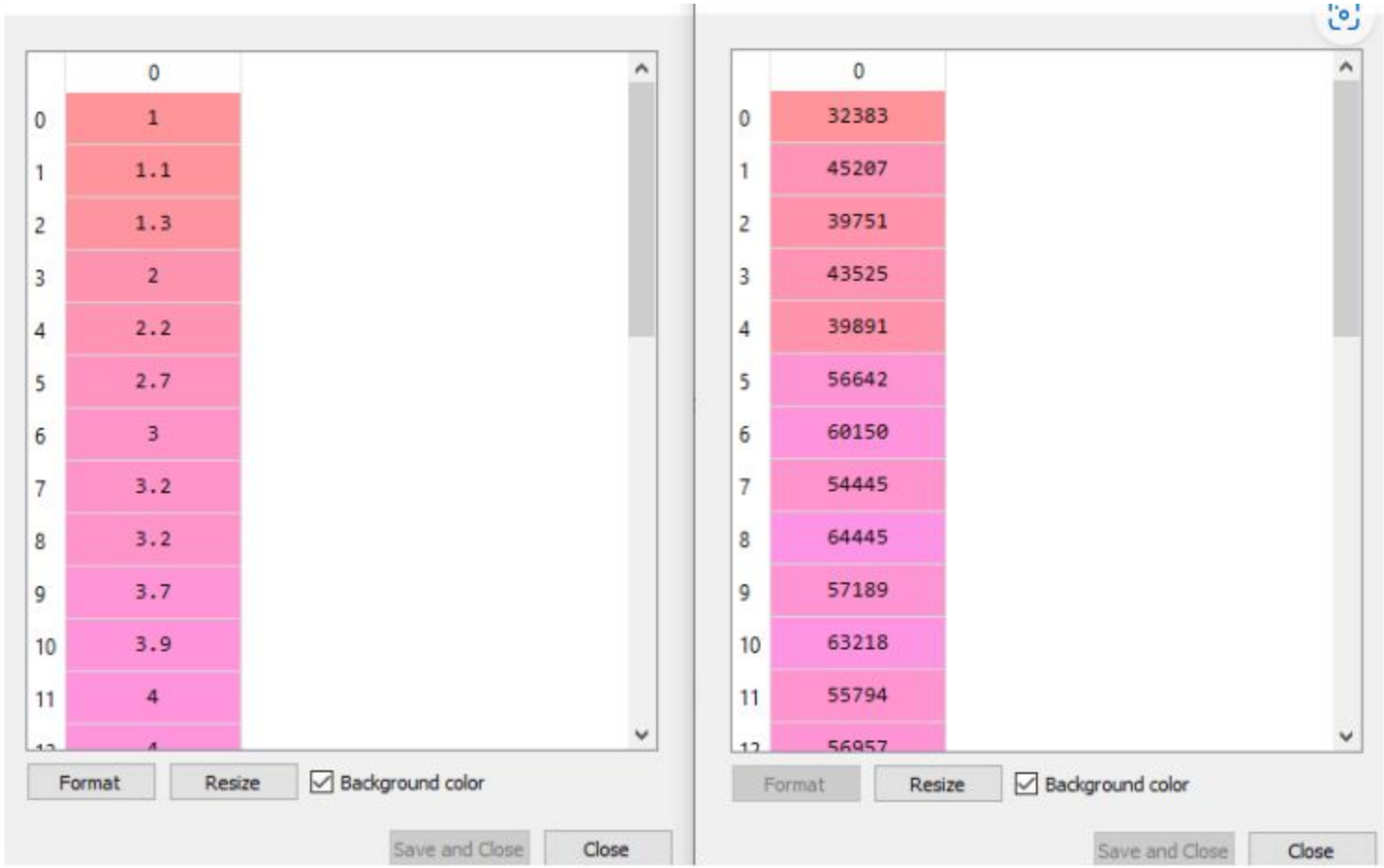


Index	YearsExperience	Salary
0	1	32383
1	1.1	45207
2	1.3	39751
3	2	43525
4	2.2	39891
5	2.7	56642
6	3	60150
7	3.2	54445
8	3.2	64445
9	3.7	57189
10	3.9	63218
11	4	55794
12	4	56957
13	4.1	57081

Linear Regression with Single Variables

x= data_set.iloc[:, :-1].values

y= data_set.iloc[:, 1].values



Linear Regression with Single Variables

Splitting the dataset into training and test set.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 1/3,
random_state=0)
```

With random_state=0 , we get the same train and test sets across different executions.

Step-2: Fitting the Simple Linear Regression to the Training Set:

```
#Fitting the Simple Linear Regression model to the training data set
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(x_train, y_train)
```

Linear Regression with Single Variables

Step: 4. visualizing the Training set results:

```
mtp.scatter(x_train, y_train, color="green")
mtp.plot(x_train, x_pred, color="red")
mtp.title("Salary vs Experience (Training Dataset)")
mtp.xlabel("Years of Experience")
mtp.ylabel("Salary(In Rupees)")
mtp.show()
```

Linear Regression with Single Variables

Step: 5. visualizing the Test set results:

```
#visualizing the Test set results
mtp.scatter(x_test, y_test, color="blue")
mtp.plot(x_train, x_pred, color="red")
mtp.title("Salary vs Experience (Test Dataset)")
mtp.xlabel("Years of Experience
mtp.ylabel("Salary(In Rupees)")
mtp.show()
```

Linear Regression with Single Variables

Step: 5. visualizing the Test set results:

```
#visualizing the Test set results
mtp.scatter(x_test, y_test, color="blue")
mtp.plot(x_train, x_pred, color="red")
mtp.title("Salary vs Experience (Test Dataset)")
mtp.xlabel("Years of Experience
mtp.ylabel("Salary(In Rupees)")
mtp.show()
```

Linear Regression with Variables

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
data = pd.read_csv('data.csv')
# Select features (X) and target variable (y)
features = data[['bedrooms', 'square_footage']]
target = data['price']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2,)
random_state=0
# Create a linear regression model
model = LinearRegression()
# Train the model
model.fit(X_train, y_train)
# Make predictions on the test set
predictions = model.predict(X_test)
```

Linear Regression with Multiple Variables

```
# Visualize the predictions
plt.scatter(y_test, predictions)
plt.xlabel('True Prices') plt.ylabel('Predicted Prices')
plt.title('True Prices vs. Predicted Prices')
 plt.show()

# Example prediction for a new house
new_house = np.array([[3, 2000]])
# New house with 3 bedrooms and 2000 square footage predicted_price =
model.predict(new_house)
print(f"Predicted Price for New House: {predicted_price[0]}")
```

Linear Regression with Multiple Variables

Problem Description:

We have a dataset of **50 start-up companies**. This dataset contains five main information: **R&D Spend, Administration Spend, Marketing Spend, State, and Profit for a financial year**. Our goal is to create a model that can easily determine which company has a maximum profit, and which is the most affecting factor for the profit of a company.

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
#importing datasets
data_set= pd.read_csv('50_CompList.csv')
```

Linear Regression with Multiple Variables

Index	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349	136898	471784	New York	192262
1	162598	151378	443899	California	191792
2	153442	101146	407935	Florida	191050
3	144372	118672	383200	New York	182902
4	142107	91391.8	366168	Florida	166188
5	131877	99814.7	362861	New York	156991
6	134615	147199	127717	California	156123
7	130298	145530	323877	Florida	155753
8	120543	148719	311613	New York	152212
9	123335	108679	304982	California	149760
10	101913	110594	229161	Florida	146122
11	100672	91790.6	249745	California	144259
12	93863.8	127320	249839	Florida	141586
13	91992.4	135495	252665	California	134307

Format Resize ☑ Background color ☑ Column min/max Save and Close Close

Linear Regression with Multiple Variables

Extracting dependent and independent Variables:

#Extracting Independent and dependent Variable

x= data_set.iloc[:, :-1].values

y= data_set.iloc[:, 4].values

```
array([[165349.2, 136897.8, 471784.1, 'New York'],
       [162597.7, 151377.59, 443898.53, 'California'],
       [153441.51, 101145.55, 407934.54, 'Florida'],
       [144372.41, 118671.85, 383199.62, 'New York'],
       [142107.34, 91391.77, 366168.42, 'Florida'],
       [131876.9, 99814.71, 362861.36, 'New York'],
       [134615.46, 147198.87, 127716.82, 'California'],
       [130298.13, 145530.06, 323876.68, 'Florida'],
       [120542.52, 148718.95, 311613.29, 'New York'],
       [123334.88, 108679.17, 304981.62, 'California'],
       [101913.08, 110594.11, 229160.95, 'Florida'],
       [100671.96, 91790.61, 249744.55, 'California'],
       [93863.75, 127320.38, 249839.44, 'Florida'],
       [91992.39, 135495.07, 252664.93, 'California'],
       [119943.24, 156547.42, 256512.92, 'Florida'],
       [114523.61, 122616.84, 261776.23, 'New York'],
       [78013.11, 121597.55, 264346.06, 'California'],
```

Linear Regression with Multiple Variables

Encoding Dummy Variables:

As we have one categorical variable (State), which cannot be directly applied to the model, so we will encode it. To encode the categorical variable into numbers, we will use the **LabelEncoder** class. But it is not sufficient because it still has some relational order, which may create a wrong model. So in order to remove this problem, we will use **OneHotEncoder**, which will create the dummy variables. Below is code for it:

```
#Catgorical data
from sklearn.preprocessing import LabelEncoder, OneHotEn
coder
labelencoder_x= LabelEncoder()
x[:, 3]= labelencoder_x.fit_transform(x[:,3])
x= labelencoder.fit_transform(x).toarray()
```

Linear Regression with Multiple Variables

	0	1	2	3	4	5
0	0	0	1	165349	136898	471784
1	1	0	0	162598	151378	443899
2	0	1	0	153442	101146	407935
3	0	0	1	144372	118672	383200
4	0	1	0	142107	91391.8	366168
5	0	0	1	131877	99814.7	362861
6	1	0	0	134615	147199	127717
7	0	1	0	130298	145530	323877
8	0	0	1	120543	148719	311613
9	1	0	0	123335	108679	304982
10	0	1	0	101913	110594	229161
11	1	0	0	100672	91790.6	249745
12	0	1	0	93863.8	127320	249839
13	1	0	0	91992.4	135495	252665

Format Resize Background color

Linear Regression with Multiple Variables

```
# Splitting the dataset into training and test set.
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_sta
te=0)
#Fitting the MLR model to the training set:
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(x_train, y_train)
#Predicting the Test set result;
y_pred= regressor.predict(x_test)
print('Train Score: ', regressor.score(x_train, y_train))
print('Test Score: ', regressor.score(x_test, y_test))
```

Linear Regression with Multiple Variables

The linear regression equation has the general form:

$$y = mx + b$$

Where:

- y is the predicted value of the dependent variable (target),
- x is the value of the independent variable (feature),
- m is the slope of the line (how much y changes with a change in x),
- b is the y-intercept (the value of y when x is 0).

In the context of multiple linear regression, where there are multiple independent variables, the equation becomes:

.

$$y = b_0 + b_1x_1 + b_2x_2 + \ldots + b_nx_n$$

Where:

- y is the predicted value of the dependent variable (target),
- $x_1, x_2, \ldots, x_n$ are the values of the independent variables (features),
- b_0 is the y-intercept (the value of y when all x values are 0),
- $b_1, b_2, \ldots, b_n$ are the coefficients (slopes) associated with each independent variable.

Unit 2 Logistic Regression

OUTLINE:

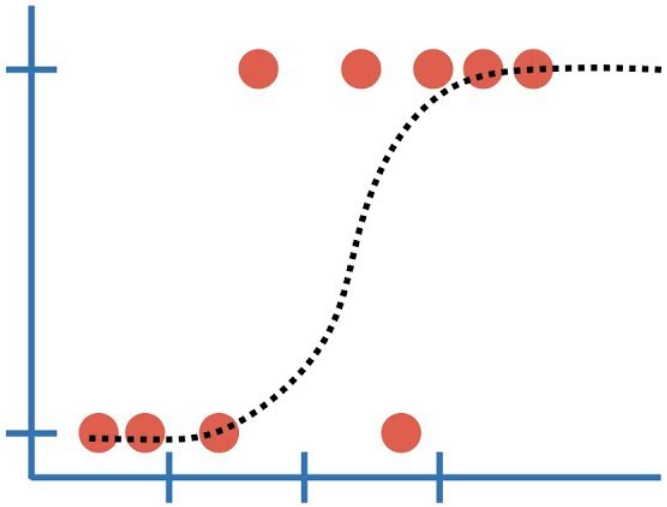
- Logistic Regression in Machine Learning

Logistic Regression in Machine Learning

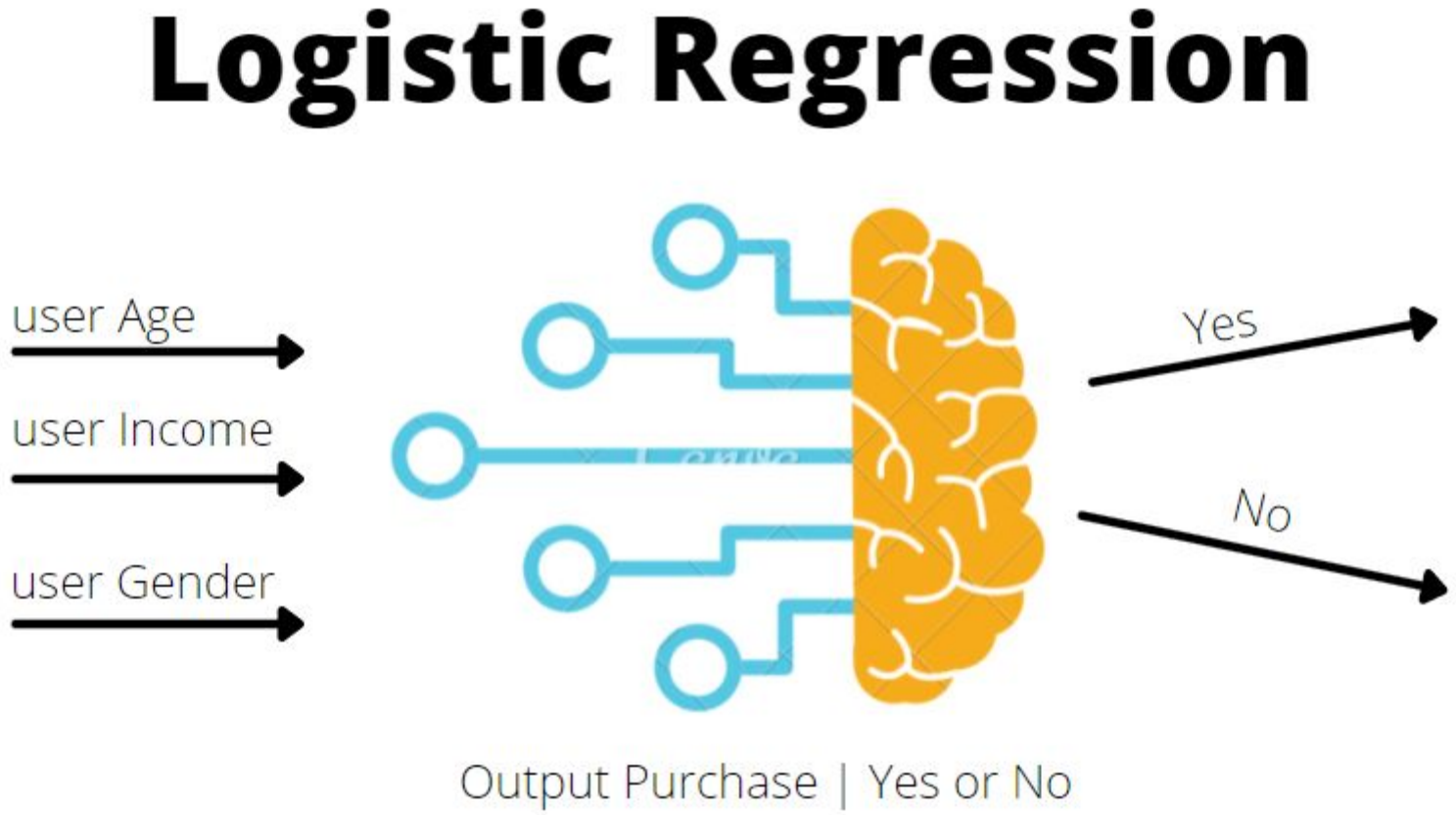
- Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class.
- It is used for classification algorithms its name is logistic regression. it's referred to as regression because it takes the output of the linear regression function as input and uses a sigmoid function to estimate the probability for the given class.
- The difference between linear regression and logistic regression is that linear regression output is the continuous value that can be anything while logistic regression predicts the probability that an instance belongs to a given class or not.

Logistic Regression in Machine Learning

Logistic Regression…



Logistic Regression in Machine Learning



Logistic Regression in Machine Learning

- Logistic regression predicts the output of a categorical dependent variable. 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**.
- Logistic Regression is much similar to the Linear Regression except that how they are used. **Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems**.
- In Logistic regression, instead of **fitting a regression line**, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

Logistic Regression in Machine Learning

Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
  - o The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form.
- The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Logistic Regression in Machine Learning

Type of Logistic Regression:

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

Binomial: In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.

Multinomial: In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"

Ordinal: In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Logistic Regression in Machine Learning

 # import the necessary libraries
 # split the train and test dataset
 # Choosing Model
 # Prediction

Logistic Regression in Machine Learning

```
# import the necessary libraries
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# load the breast cancer dataset
X, y = load_breast_cancer(return_X_y=True)
# split the train and test dataset
X_train, X_test,\y_train, y_test = train_test_split(X, y,test_size=0.20,random_state=23)
# LogisticRegression
clf = LogisticRegression(random_state=0)
clf.fit(X_train, y_train)
# Prediction
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Logistic Regression model accuracy (in %):", acc*100)
```

Unit 2 Spam Filtering in Logistic Regression

OUTLINE:

- ❖ Spam Filtering in Logistic Regression

Spam Detection with Logistic Regression | by Natasha Sharma | Towards Data Science

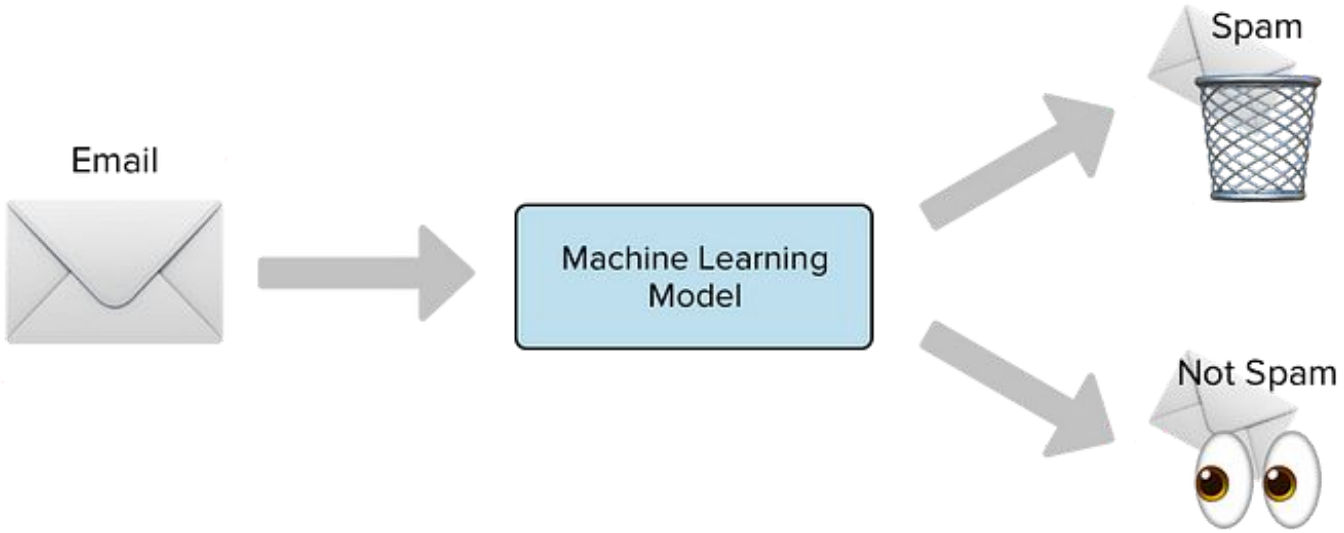
Spam Filtering in Logistic Regression

what is logistic regression?

- Logistic regression is a simple classification algorithm. Given an example, we try to predict the probability that it belongs to "0" class or "1" class.
- Remember that with linear regression, we tried to predict the value of y(i) for x(i). Such continuous output is not suited for the classification task.
- Given the logistic function and an example, it always returns a value between one and zero.

$$f(x) = \frac{1}{1+e^{-\beta x}}$$

Spam Filtering in Logistic Regression



Spam Filtering in Logistic Regression

Create a Spam Detector : Pre-processing

ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
ham	Ok lar... Joking wif u oni...
spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's
ham	U dun say so early hor... U c already then say...
ham	Nah I don't think he goes to usf, he lives around here though
spam	FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, å£1.50 to rcv
ham	Even my brother is not like to speak with me. They treat me like aids patent.
ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune
spam	WINNER!! As a valued network customer you have been selected to receivea å£900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only.
spam	Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030

Kaggle Spam Detection Dataset

The above image is a snapshot of tagged email that have been collected for Spam research.

It contains one set of messages in **English of 5,574 emails**, tagged according being legitimate(ham) or spam.

Now that we have data with tagged emails — Spam or Not Spam, what should we do next? We would need to train the machine to make it smart enough to categorize the emails on its own. But, the machine can’t read the full statement and start categorizing the emails.

Spam Filtering in Logistic Regression

Pre-processing on message text, like removing - punctuation and stop words.

```
def text_preprocess(text):
text = text.translate(str.maketrans('', '', string.punctuation))
text = [word for word in text.split() if word.lower() not in
stopwords.words]
```

maketrans()

- maketrans() function is used to construct the transition table i.e specify the list of characters that need to be replaced in the whole string or the characters that need to be deleted from the string
- Syntax : **maketrans(str1, str2, str3)**
- **Parameters :**

  str1 : Specifies the list of characters that need to be replaced.

  str2 : Specifies the list of characters with which the characters need to be replaced.

  str3 : Specifies the list of characters that needs to be deleted.
- **Stop Words:** A stop word is a commonly used word (such as “the”, “a”, “an”, “in”, “for”))

Spam Filtering in Logistic Regression

Once the pre-processing is done, Vectorize the data — i.e collecting each word and its frequency in each email. The vectorization will produce a matrix.

```
vectorizer = TfidfVectorizer("english")
message_mat = vectorizer.fit_transform(message_data_copy)
message_mat
```

TF-IDF will transform the text into meaningful representation of integers or numbers which is used to fit machine learning algorithm for predictions.
Frequency-Inverse Document Frequency

Spam Filtering in Logistic Regression

This vector matrix can be used create train/test split. This will help us to train the model/machine to be smart and test the accuracy of its results.

```
message_train, message_test, spam_nospam_train,
spam_nospam_test = train_test_split(message_mat,
message_data['Spam/Not_Spam'], test_size=0.3, random_state=0)
```

Spam Filtering in Logistic Regression

Choosing a model

Now that we have train test split, we would need to choose a model. There is a huge collection of models but for this particular exercise we will be using logistic regression.Why?

Generally when someone asks, what is logistic regression? what do you tell them — Oh! it is an algorithm which is used for categorizing things into two classes (most of the time) i.e. the result is measured using a dichotomous variable. But, how does logistic regression classify thing into classes like -**binomial**(2 possible values), **multinomial**(3 or more possible values) and **ordinal**(deals with ordered categories). For this post we will only be focusing on binomial logistic regression i.e. the outcome of the model will be categorized into two classes.

Spam Filtering in Logistic Regression

For the Spam detection problem, we have tagged messages but we are not certain about new incoming messages. We will need a model which can tell us the probability of a message being Spam or Not Spam. Assuming in this example , 0 indicates — negative class (absence of spam) and 1 indicates — positive class (presence of spam), we will use logistic regression model.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

Spam_model = LogisticRegression(solver='liblinear')
Spam_model.fit(message_train, spam_nospam_train)
pred = Spam_model.predict(message_test)
accuracy_score(spam_nospam_test,pred)
```

Unit 2 Naïve Bayes Algorithm

OUTLINE:

- ❖ Naïve Bayes Algorithm

Naïve Bayes Algorithm

Naïve Bayes Classifier Algorithm

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

Naïve Bayes Algorithm

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.

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

Naïve Bayes Algorithm

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.

The formula for Bayes' theorem is given as:

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

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.

Naïve Bayes Algorithm

Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

- Convert the given dataset into frequency tables.
- Generate Likelihood table by finding the probabilities of given features.
- Now, use Bayes theorem to calculate the posterior probability.

Naïve Bayes Algorithm

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

Naïve Bayes Algorithm

Frequency table for the Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	4/14=0.29	10/14=0.71	

Naïve Bayes Algorithm

Applying Bayes'theorem:

P(Yes|Sunny)= P(Sunny|Yes)*P(Yes)/P(Sunny)
P(Sunny|Yes)= 3/10= 0.3
P(Sunny)= 0.35
P(Yes)=0.71
So P(Yes|Sunny) = 0.3*0.71/0.35= **0.60**

P(No|Sunny)= P(Sunny|No)*P(No)/P(Sunny)
P(Sunny|NO)= 2/4=0.5
P(No)= 0.29
P(Sunny)= 0.35
So P(No|Sunny)= 0.5*0.29/0.35 = **0.41**
So as we can see from the above calculation that **P(Yes|Sunny)>P(No|Sunny)**
Hence on a Sunny day, Player can play the game.

Naïve Bayes Algorithm

Types of Naïve Bayes Model:

There are three types of Naive Bayes Model, which are given below:

Gaussian: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.

Multinomial: The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc.

The classifier uses the frequency of words for the predictors.

Bernoulli: The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

Unit 2 Principal Component Analysis

OUTLINE:

- ❖ Principal Component Analysis

Principal Component Analysis

- Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning.
- It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the **Principal Components**.
- PCA generally tries to find the lower-dimensional surface to project the high-dimensional data.
- PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are ***image processing, movie recommendation system, optimizing the power allocation in various communication channels.*** It is a feature extraction technique, so it contains the important variables and drops the least important variable.

Principal Component Analysis

The PCA algorithm is based on some mathematical concepts such as:

- Variance and Covariance
- Eigenvalues and Eigen factors

Some common terms used in PCA algorithm:

- **Dimensionality:** It is the number of **features or variables present** in the given dataset. More easily, it is the number of columns present in the dataset.
- **Correlation:** It signifies that **how strongly two variables are related to each other.** Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.
- **Orthogonal:** It defines that **variables are not correlated to each other**, and hence the correlation between the pair of variables is zero.
- **Eigenvectors:** If there is a square matrix M, and a non-zero vector v is given. Then v will be eigenvector if Av is the scalar multiple of v.
- **Covariance Matrix:** A matrix containing the covariance between the pair of variables is called the Covariance Matrix.

Principal Component Analysis

Principal Components in PCA

As described above, the transformed new features or the output of PCA are the Principal Components. The number of these PCs are either equal to or less than the original features present in the dataset. Some properties of these principal components are given below:

The principal component must be the linear combination of the original features.

These components are orthogonal, i.e., the correlation between a pair of variables is zero.

The importance of each component decreases when going to 1 to n, it means the 1 PC has the most importance, and n PC will have the least importance.

Principal Component Analysis

Steps for PCA algorithm

Getting the dataset:

Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.

Representing data into a structure:

Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.

Standardizing the data:

Standardisation. This technique is to re-scale features value with the distribution value between 0 and 1 is useful for the optimization algorithms, such as gradient descent, that are used within machine learning algorithms that weight inputs (e.g., regression and neu .

$$Z = \frac{X-\mu}{\sigma}$$

Principal Component Analysis

Calculating the Covariance of Z

Covariance, is that it helps us to understand and quantify the relationship between 2 variables in a dataset

$$\text{cov}(X, Y) = \frac{1}{n} \sum_{i=1}^{n} (x - \bar{x})(y - \bar{y})$$

$\text{cov}(X, Y)$	⟶	Covariance between X & Y variables
x & y	⟶	members of X & Y variables
$\bar{x}$ & $\bar{y}$	⟶	mean of X & Y variables
n	⟶	number of members

Principal Component Analysis

Calculating the Eigen Values and Eigen Vectors

Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues. $|A - \lambda I| = 0$ for λ.

Sorting the Eigen Vectors

In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as P*.

Principal Component Analysis

Calculating the new features Or Principal Components

Here we will calculate the new features. To do this, we will multiply the P* matrix to the Z. In the resultant matrix Z*, each observation is the linear combination of original features. Each column of the Z* matrix is independent of each other.

Remove less or unimportant features from the new dataset.

The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

Unit 2 Bayesian Classifier

OUTLINE:

- ❖ Bayesian Classifier

Bayesian Classifier

Bayesian Classifier:

- A Bayesian classifier, also known as a Bayesian classification algorithm, is a type of classification algorithm that uses the principles of Bayesian statistics to make predictions about the class labels of input data points.
- The key idea behind Bayesian classification is to calculate the posterior probability of each class given the observed data and then make predictions based on the class with the highest posterior probability.

Bayesian Classifier

Bayesian Classifier:

Here's how a Bayesian classifier typically works:

Bayes' Theorem: The algorithm is based on Bayes' theorem, which is a fundamental concept in probability theory. Bayes' theorem relates the conditional probability of an event to the joint probability of that event and another related event. In the context of classification, it can be expressed as:

$$P(C_i|X) = \frac{P(X|C_i) \cdot P(C_i)}{P(X)}$$

Where:

- C_i is the ith class.
- $P(C_i|X)$ is the posterior probability of class C_i given input data X.
- $P(X|C_i)$ is the likelihood of observing data X given class C_i.
- $P(C_i)$ is the prior probability of class C_i.
- $P(X)$ is the probability of observing data X (a normalizing factor).

Unit 2 Ridge Regression

OUTLINE:

- ❖Ridge Regression

Ridge Regression

Ridge Regression:

- Ridge regression is a model tuning method that is used to analyse any data that suffers from **multicollinearity.**
- This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are **unbiased**, and **variances** are large, this results in predicted values being far away from the actual values.
- **Ridge regression is mostly used to reduce the overfitting** in the model, and it includes all the features present in the model.
- It reduces the complexity of the model by shrinking the coefficients.
- Multicollinearity can be defined as the presence of **high correlations between two or more independent variables** (predictors).
- This is essentially a phenomenon in which independent variables are correlated with one another.

Ridge Regression

Ridge Regression:

- Ridge Regression is particularly effective at addressing multicollinearity, which is a situation where two or more predictor variables in a regression model are highly correlated with each other.
- Multicollinearity can cause instability in the coefficient estimates of the linear regression model, making it difficult to interpret the individual effects of each predictor variable. Ridge Regression helps mitigate the effects of multicollinearity through the introduction of the L2 regularization term.

- Least squares is a commonly used method in regression analysis for estimating the unknown parameters by creating a model which will minimize the sum of squared errors between the observed data and the predicted data.

Ridge Regression

Ridge Regression:

Here's how Ridge Regression reduces the impact of multicollinearity:

Shrinking Coefficients: The regularization term added to the Ridge Regression objective function penalizes large coefficient values. This means that the model is encouraged to have smaller coefficient values overall. In the presence of multicollinearity, where some predictor variables are highly correlated, Ridge Regression tends to distribute the coefficient values across correlated variables more evenly. This helps to reduce the dominance of any single variable in the model and makes the coefficients more stable.

Balancing Effects: In cases of strong multicollinearity, ordinary least squares (OLS) regression can lead to highly variable coefficient estimates. Ridge Regression, by penalizing large coefficients, encourages the model to find a balance between the correlated variables' effects. Instead of assigning a large coefficient to one variable and a very small coefficient to the other (as OLS might), Ridge Regression might assign moderate coefficients to both variables, reflecting their shared influence on the target variable.

Ridge Regression

Ridge Regression:

Reducing Overfitting: Multicollinearity can exacerbate overfitting because the model can fit the noise present in the correlated variables. Ridge Regression's regularization term prevents the model from fitting the noise too closely, as the cost of having large coefficients increases. This can lead to improved generalization performance on new, unseen data.