**Machine Learning Notes**

**Machine Learning**

Machine Learning is the science of getting Computers to learn and act like humans do, and improve their learning overtime on autonomous fashion, by feeding those data and information in the form of observations and real-world interactions.

# How does Machine Learning Work?

A Machine Learning System learn from historical data, builds the prediction models, and whenever it receives new data, predicts the output for it.

# Classifications of Machine Learning

* Supervised Learning
* Unsupervised Learning
* Reinforcement Learning

# Advantages of Machine Learning

1. Easily Identifies trends and patterns
2. No human intervention needed (automation)
3. Continuous improvement
4. Handling multi-dimensional and multiversity data
5. Wide Applications

# Disadvantages of Machine Learning

1. Data acquisition
2. Time and Resources
3. Interpretation of Results
4. High error-susceptibility

# Use of Machine Learning



# Machine Learning Road Map

1. Programming Language – Python, R
2. Exploratory Data Analysis
3. Feature Engineering / Data Cleaning
   * Exploratory Data Analysis
   * Handling Missing Values
   * Handling Outliers
   * Categorical Encoding
   * Normalizing & Standardization
4. Feature Selection
   * Correlation
   * Forward Elimination
   * Backward Elimination
   * Univariate Selection
   * Random Forest Importance
   * Feature selection with Decision Trees
5. Machine Learning Algorithms – Regression and Classification, Clustering
6. Linear, Logistic Regression, Decision Tree, Random Forest, Kmeans
7. Hyper Parameter Tuning
8. Gridsearch, RandomisedSreach, Hypropot, Genetics Algorithms
9. Dockers and Kubernetes
10. Model Deployments
11. End to End ML Projects

# Learn Python Libraries for Machine Learning

# Numpy Pandas

# Matplotlib Seaborn

# Tensor Flow Scikit Learn

# SciPy

# Types of Variables in Machine Learning

## Data Types

Variables (columns)

* Numerical Data (int, float)
  + Discrete (that repeat again and again like [1,3,5,3,4,5])
  + Continuous (that given rang of decimal data [1.1,1.2,1.3 to 5] )
* Categorical Data (speaking a objects data)
  + Ordinal (e.g. male, female)
  + Nominal (e.ge python, java, c++, c#, html)

## Date and Time Data

Data time can contain data only, time only

## Mixed Data

Variables which contains numbers and categories data.

# Feature Engineering / Data Cleaning

* + Exploratory Data Analysis
  + Handling Missing Values
  + Handling Outliers
  + Categorical Encoding
  + Normalizing & Standardization cover all topics in data cleaning folder

# Feature Selection Techniques (column selection)

A feature is an attribute that has an impact on a problem or is useful for the problem, and choosing the important features for the model is known as feature selection.

## Forward Elimination (using MLXTEND)



from mlxtend.feature\_selection import SequentialFeatureSelector

x = dataset.iloc[:,:-1] # saperate x-axis and y-axis

y = dataset["species"]

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

fs = SequentialFeatureSelector(lr,k\_features=4, forward=True) /‘False’ for backward elimination

fs.fit(x,y)

fs.feature\_names

fs.k\_feature\_names\_

fs.k\_score\_ here save the score during checking different feature and select high score

## Backward Elimination (using MLXTEND)



fs = SequentialFeatureSelector(lr,k\_features=4, forward=False) change only this for backward elimination and otherwise above syntax is same for backward.

# Train Test Split in Data Set

input\_data = dataframe.iloc[:,:-1] separate data for input

output\_data = dataframe["Embarked"] separate data for output only select last column

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(input\_data, output\_data, test\_size=0.25)

use train data when we train machine learning model and use test data when we check accuracy

# REGRESSION ANALYSIS

## Supervise Learning

Regression Analysis Classification Analysis

# Regression analysis

**Linear Relationship** **Non Linear Relationship**

Simple Linear Regression Polynomial Regression

Multi Linear Regression Decision Tree Regression

Lasso Regression Random Forest Regression

Ridge Regression Support Vector Machine

K-Nearest Neighbor

## Regression Analysis

In a dataset, when determining the type of prediction, you base your decision on the output. If the output or outcome is continuous, you use regression analysis. Otherwise, you use classification analysis.

**Linear Regression**Linear regression is used when the relationship between the independent variable(s) (predictor) and the dependent variable (outcome) is linear. In other words, the data points can be approximately modeled by a straight line.

Linear regression, multi-linear regression, Lasso regression, Ridge regression

### **Nonlinear Regression**

Nonlinear regression is used when the relationship between the independent variable(s) and the dependent variable is nonlinear. This means the data cannot be modeled by a straight line and may follow a curve or more complex relationships.

Polynomial regression, Decision tree regression, Random forest regression, Support Vector Machine, k-nearest neighbor’s regression



## Linear Regression Algorithm (Simple Linear)

Simple Linear Regression Algorithm is a type of Regression Algorithms that models the relationship between a dependent variable and a single independent variable.



sl\_dataset = pd.read\_csv("simple\_linear\_regression\_dataset.csv")

sl\_dataset.head(3)

sl\_dataset.isnull().sum()

x=sl\_dataset[["Feature"]]

y=sl\_dataset["Target"]

plt.figure(figsize=(10, 6))

plt.scatter(x="Feature", y="Target", data=sl\_dataset,color='blue', alpha=0.6, label='Data Points')

plt.title('Scatter Plot of Feature vs Target', fontsize=16)

plt.xlabel('Feature (X)', fontsize=14)

plt.ylabel('Target (Y)', fontsize=14)

plt.axhline(y=0, color='black', linewidth=0.8, linestyle='--')

plt.axvline(x=0, color='black', linewidth=0.8, linestyle='--')

plt.legend()

plt.grid(True, linestyle='--', alpha=0.5)

plt.show()

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) # 0 to 70,80

from sklearn.linear\_model import LinearRegression

lr = LinearRegression()

lr.fit(x\_train, y\_train)

lr.score(x\_test,y\_test)\*100

lr.predict([[37.454012]]) this is for user

# y = m\*x + c

lr.coef\_ # m

lr.intercept\_ # c

# y=2.52327729\*37.454012+8.772462946297566

y\_prd = lr.predict(x)

plt.figure(figsize=(10, 6))

plt.scatter(x="Feature", y="Target", data=sl\_dataset,color='blue', alpha=0.6, label='Data Points')

plt.plot(sl\_dataset["Feature"], y\_prd,color = "red", label="predict line")

plt.title('Scatter Plot of Feature vs Target', fontsize=16)

plt.xlabel('Feature (X)', fontsize=14)

plt.ylabel('Target (Y)', fontsize=14)

plt.axhline(y=0, color='black', linewidth=0.8, linestyle='--')

plt.axvline(x=0, color='black', linewidth=0.8, linestyle='--')

plt.legend()

plt.grid(True, linestyle='--', alpha=0.5)

plt.show()

## Multiple Linear Regression

Multiple linear regression is an extension of simple linear regression as it take more than one predictor variable to predict the response variable.

x = ml\_dataset.iloc[:,:-1]

y = ml\_dataset["Target"]

x.ndim

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)

from sklearn.linear\_model import LinearRegression

lr = LinearRegression()

lr.fit(x\_train,y\_train)

lr.score(x\_test,y\_test)

# y m1\*x1+m2\*x2+m3\*x3+c

ml\_lr.coef\_

ml\_lr.intercept\_

# y\_prd = 3.03737113\*Feature1+-1.44414216\*Feature2+2.0121138\*Feature3 + 15.89343046851019

## Polynomial Regression

Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable (x) as nth degree polynomial.

x = dataset[["Feature"]]

y = dataset["Target"]

from sklearn.preprocessing import PolynomialFeatures

pf = PolynomialFeatures(degree=2)

pf.fit(x)

x = pf.transform(x)

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)

from sklearn.linear\_model import LinearRegression

lr = LinearRegression()

lr.fit(pr\_x\_train,pr\_y\_train)

lr.score(pr\_x\_test,pr\_y\_test)

# y = m1\*x1+m2\*x2^2+c

# y = -1.05954074\*x1 + 0.46752086\*x2^2 - 0.8695309013342438

lr.coef\_ # m

lr.intercept\_ # c

prd = pr\_lr.predict(pr\_x)

plt.scatter(dataset["Feature"],dataset["Target"])

plt.plot(dataset["Feature"], prd, c="red")

plt.show()

**Model Deployment**

test = pf.transform([[4]]) 4 as a example

lr.predict(test)

# What is Cost Function

How to make best fit line.

Cost Function help us to make / predict a best fit line.

A cost function is an important parameter that determines how well a machine learning model performs for a given dataset.

Cost function is a measure of how wrong the model is in estimating the relationship between X (input) and Y (output) parameter.

## Types of Cost Function

* Regression cost function
* Classification cost function

## Regression cost function

Regression models are used to make a prediction for the continuous variables.

* **MSE** (Mean Square Error)
* **RMSE** (Root Mean Square Error)
* **MAE** (Mean Absolute Error)
* **R2** Accuracy

## Binary Classification Cost Function

Classification models are used to make predictions of categorical variables, such as predictions for 0 or 1, Cat or Dog, etc.

## Multi-class Classification Cost Function

A multi-class classification cost function is used in the classification problem for which instances are allocated to one of more than two classes.

* **Binary Cross Entropy Cost Function or Log Loss Function**

# Regression Cost Function

## Mean Square Error:

Mean Square Error (MSE) is the mean square difference between the actual and predicted values. MSE penalizes high errors caused by outliers by squaring the errors.

Mean Square Error is also known as L2 Loss.

Formula search on google MAE=*n*1​∑*i*=1*n*​(*yi*​−*y*^​*i*​)2

## Mean Absolute Error:

Mean Absolute Error (MAE) is the mean absolute difference between the actual values and the predicted values.

MAE is more robust to outliers. The insensitivity to outliers is because it does not penalize high error caused by outliers.

Formula search on google MAE=*n*1​∑*i*=1*n*​∣*yi*​−*y*^​*i*​∣

## Root Mean Square Error:

Root mean square error (RMSE) is the root squared mean of the difference between actual and predicted values.

RMSE can be used in situations where we want to penalize high errors but not as much as MSE does.

Formula search on google RMSE= √MSE

**Practical of (MSE, MAE, RMSE)**

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

print(mean\_squared\_error(y\_test,lr.predict(x\_test)))

print(mean\_absolute\_error(y\_test,lr.predict(x\_test)))

print(np.sqrt(mean\_squared\_error(y\_test,lr.predict(x\_test))))

# How to Find Best Fit Line

Note in your notebook.

# L1 (Lasso Regularization), L2 (Ridge Regularization)

This is the form of regression, that constrains / regularizes or shrinks the coefficient estimates towards zero.

This techniques discourages learning a more complete or flexible model, so as to avoid the risk of overfitting.

**Regularization can achieve this motive with 2 techniques:**

* Ridge Regularization / L2
* Lasso Regularization / L1

## Lasso Regularization / L1

This is a Regularization Techniques used in Feature Selection using a shrinkage method also referred to as the penalized regression method.

Lasso Regularization magnitude of coefficient can be exactly zero.

Cost Function = Loss + **λ ∑ || W ||**

Loss = sum of squared residual

**λ** = penalty

W = slope of the curve

**Practical**

from sklearn.linear\_model import Lasso

ls = Lasso(alpha=0.5)

ls.fit(x\_train, y\_train)

ls.score(x\_test,y\_test)\*100

## Ridge Regularization / L2

Ridge Regularization, is also known as L2 Regularization, is an extension to linear regression that introduces a regularization term to reduce model complexity and help prevent overfitting.

Ridge Regularization is working value/ magnitude of coefficient is almost equal to zero.

Cost Function = loss + **λ** ∑||W||2

Loss = sum of squared residual

**λ** = penalty

W = slope of the curve

**Practical**

from sklearn.linear\_model import Ridge

ri = Ridge(alpha=10)

ri.fit(x\_train, y\_train)

ri.score(x\_test,y\_test)\*100

# Classification Analysis

* The classification algorithm is used to identify the category of new observations on the basis of training data.
* In Classification, a program learn from the given dataset or observations and then classifies new observations 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:

There are two types of classifications

* **Binary Classifier:** If the Classification problem has only two possible outcomes, then it is called as Binary Classifier.

**Examples:** Spam or Not Spam, Cat or Dog etc.

* **Multi-Class Classifier:** If a Classification problem has more than two outcomes, then it is called as Multi Class Classifier.

**Examples:** Classifications of types of crops, Classifications of types of music.

# Types of ML Classifications Algorithms

## Non Linear Models Linear Models

K-Nearest Neighbors Logistic Regression

SVM (Kernel) Support Vector Machines

Naïve Bayes

Decision Tree Classification

Random Forest Classification

## Evaluating a Classification Model / Classification Cost Function

* Log Loss or Cross-Entropy Loss
* Confusion Matrix
* AUC-ROC curve

# Logistic Regression (Binary Classification)

* Logistic regression is one of the most popular machine learning algorithms, which comes under the supervised learning techniques.
* 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 given the exact values as 0 and 1, it gives the probabilistic which lie between 0 and 1.

## Types of Logistic Regression

On the basis of 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 and 1, Pass and Fail, etc.

**Multinomial:** In multinomial logistic regression, there can be 3 or more possible unordered types of the dependent variables, such as “cat”, “dog”, or ”sheep”.

**Ordinal:** In ordinal logistic regression, there can be 3 or more possible ordered types of the dependent variables, such as “Low”, “Medium”, or ”High”.

# Practice

## Logistic Regression (Binary Classification) (Binary input)

dataset = pd.read\_csv("logistic\_regression\_dataset.csv")

dataset.head()

plt.figure(figsize=(4,3))

sns.scatterplot(x="Feature",y="Target",data=dataset)

plt.show()

x = lo\_dataset[["Feature"]]

y = lo\_dataset["Target"]

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)

from sklearn.linear\_model import LogisticRegression

lo = LogisticRegression()

lo.fit(x\_train,y\_train)

lo.score(x\_test,y\_test)

lr.predict([[40]])

plt.figure(figsize=(4,3))

sns.scatterplot(x="Feature",y="Target",data=dataset)

sns.lineplot(x="Feature",y= lr.predict(x),data=dataset)

plt.show()

## Logistic Regression (Binary Classification) (Multiple input)

dataset = pd.read\_csv("logistic\_regression\_customer\_purchase\_dataset.csv")

dataset.head()

plt.figure(figsize=(5,4))

sns.scatterplot(x="Income",y="AdClicks",data=dataset, hue="Purchase")

plt.show()

x = lo\_dataset.iloc[:,:-1]

y = lo\_dataset["Purchase"]

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)

from sklearn.linear\_model import LogisticRegression

lo = LogisticRegression()

lo.fit(x\_train,y\_train)

lo.score(x\_test,y\_test)

lo.coef\_

lo.intercept\_

lo.predict([[23,100000, 25]])

from mlxtend.plotting import plot\_decision\_regions

plot\_decision\_regions(x.to\_numpy(),y.to\_numpy(), clf=lo)# this will work when have two inputs

plt.show()

## Logistic Regression (Binary Classification) (Polynomial input)

dataset = pd.read\_csv("logistic\_regression\_heart\_disease\_dataset.csv")

dataset.head()

plt.figure(figsize=(5,4))

sns.scatterplot(x="BMI",y="Cholesterol", data=dataset, hue="HeartDisease")

plt.show()

x = dataset[["BMI","Cholesterol"]]

y = dataset["HeartDisease"]

from sklearn.preprocessing import PolynomialFeatures

pf = PolynomialFeatures(degree=3)

pf.fit(x)

x = pd.DataFrame(pf.transform(x))

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)

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

lr.fit(x\_train, y\_train)

lr.score(x\_test, y\_test)

# 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 categorizing thinks correctly.
* It is also known as the error matrix.
* The matrix consists of predictions result in a summarized form, which has a total number of correct predictions and incorrect predictions.

## Formula

* Accuracy = TP + TN / N
* Error = FP +FN / 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 as **Type-I error**.

## Confusion Matrix (sensitivity, precision, recall, F1-score)

Precision: TP / TP + FP

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

Recall: TP / TP + FN

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

F1 – Score:

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

F1 – Score = 2 \* Precision \* Recall / Precision + Recall

## Practical

from sklearn.metrics import confusion\_matrix, precision\_score, recall\_score, f1\_score, accuracy\_score, classification\_report

cf = confusion\_matrix(y\_test, lr.predict(x\_test))

sns.heatmap(cf, annot=True)

plt.show()

precision\_score(y\_test, lr.predict(x\_test))\*100

recall\_score(y\_test, lr.predict(x\_test))\*100

f1\_score(y\_test, lr.predict(x\_test))\*100

# Imbalanced Data

## Techniques to Handle Imbalanced Data

## Random Under Sampling

We will reduce the majority of the class So that it will have same no of as the minority.

x = dataset[["BMI", "Cholesterol"]]

y = dataset["HeartDisease"]

y.value\_counts()

from imblearn.under\_sampling import RandomUnderSampler

ru = RandomUnderSampler()

ru\_x, ru\_y = ru.fit\_resample(x,y)

ru\_y.value\_counts()

## Random Over Sampling

We will increase the size of minority is inactive class to the size of majority class is active.

x = dataset[["BMI", "Cholesterol"]]

y = dataset["HeartDisease"]

y.value\_counts()

from imblearn.over\_sampling import RandomOverSampler

ro = RandomOverSampler()

ro\_x, ro\_y = ro.fit\_resample(x,y)

ro\_y.value\_counts()

# Naïve Bayes