Classification

Features: can be any type of Data
Label: It can be **categorical data** or **district numerical data**.

We have two types of classification:

1) Binary classification:
    - Label column will have two unique values(yes/no, 0/1, spam/ham)
2) Multi – class classification
    - Label column will have more than two unique values.

Classification algorithms:

1- Logistic Regression
2- K-Nearest Neighbor
3- SVC
4- Decision Tree Classifier
5- Random Forest Classifier
6- XGB Classifier
7- XGBR Classifier

Logistic Regression algorithm

- Classification algorithm
- Applying sigmoid function on a line function Fn(y= b0 + b1x1)
- sigmoid function ➔ y=1/(1+e^(-xy))
- if y in Sigmoid on Linear Regression ➔ Logistic Regression = 1/(1+e^(-(b0+b1x))
- λ= 1 (λ is lambda)
- In binary Classification, Any number between 0 – 0.05 consider as 0 and Any number between 0.05 – 1 consider as 1.
- Default output of Logistic Regression is probability Values. Because Logistic Regression is a classification, we prefer to see categories as an output rather that probability Values as an output.

Classification Dataset:

- Balanced Dataset.
  Numbers of records for each unique label must be same. For example: number of (Spam) = number of (Ham)
- UnBalanced Dataset
  Numbers of records for each unique label are different. For example: number of (Spam) =! number of (Ham)

Evaluation Metrics

Evaluating Classification Models:

- Accuracy
- Precision
- Recall
- F1-Score

When to use which metrics?

- Balanced Dataset ➔ Accuracy
- Unbalanced Dataset ➔ Precision-Recall Pair, F1-Score

Rules for Classification (From Sklearn)

1. Data must be complete
2. Data must be strictly numeric
3. Features must be in the form of 2d numpy array
4. Label must be in the form of 1d numpy array(In Regression,label should be 2d numpy array)

Issue in terms of Achieving CL

- Play with Random State[Playing with Sampling method]
- Change the ration of train test split[80:20, 75:25, 90:10]
- Change algorithm
- Tune Hyperparameters
- Ask for more Data

Use-case for Logistic Regression:

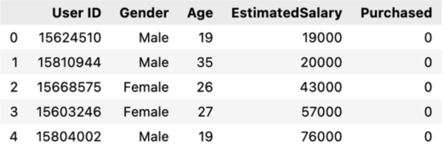
An Online Shopping Mall has provided this dataset. Your job is to create a model that can predict whether the customer will shop or not based on customer's age and estimated salary

```
data = pd.read_csv('Social_Network_Ads.csv')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   User ID          400 non-null    int64 
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    int64
 4   Purchased        400 non-null    int64
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

```
data.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

Assumption is: Data Preprocesssing is DONE

Check whether the dataset is a balanced dataset or not

# 0 --- bad customer (no purchase)

# 1 --- good customer (purchase)

```
data.Purchased.value_counts()

0    257
1    143
Name: Purchased, dtype: int64
```

Here the dataset is Unbalanced

Features and Label

```
features = data.iloc[:,[2,3]].values
label = data.iloc[:,4].values
```

Create Good Model

Finding a Generalized Model

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression


for i in range(1,401):
    X_train,X_test,y_train,y_test =
train_test_split(features,label,test_size=0.2,random_state=i)
    model = LogisticRegression()
    model.fit(X_train,y_train)

    train_score = model.score(X_train,y_train)
    test_score = model.score(X_test,y_test)

    if test_score > train_score:
        print("Test {} Train {} RS
{}".format(test_score,train_score,i))
```

```
Test 0.6875 Train 0.63125 RS 3
Test 0.7375 Train 0.61875 RS 4
Test 0.6625 Train 0.6375 RS 5
Test 0.65 Train 0.640625 RS 6
Test 0.675 Train 0.634375 RS 7
Test 0.675 Train 0.634375 RS 8
Test 0.65 Train 0.640625 RS 10
Test 0.6625 Train 0.6375 RS 11
Test 0.7125 Train 0.625 RS 13
Test 0.675 Train 0.634375 RS 16
Test 0.7 Train 0.628125 RS 17
Test 0.7 Train 0.628125 RS 21
Test 0.65 Train 0.640625 RS 24
Test 0.6625 Train 0.6375 RS 25
Test 0.75 Train 0.615625 RS 26
Test 0.675 Train 0.634375 RS 27
Test 0.7 Train 0.628125 RS 28
Test 0.6875 Train 0.63125 RS 29
Test 0.6875 Train 0.63125 RS 31
Test 0.6625 Train 0.6375 RS 37
Test 0.7 Train 0.628125 RS 39
Test 0.7 Train 0.628125 RS 40
Test 0.65 Train 0.640625 RS 42
Test 0.725 Train 0.621875 RS 46
Test 0.65 Train 0.640625 RS 48
...
Test 0.6625 Train 0.6375 RS 393
Test 0.675 Train 0.634375 RS 396
Test 0.7 Train 0.628125 RS 397

Test 0.7125 Train 0.625 RS 400
```

```
X_train,X_test,y_train,y_test =
train_test_split(features,label,test_size=0.2,random_state=199)
finalModel = LogisticRegression()
finalModel.fit(X_train,y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,

                   warm_start=False)
```

```
print(finalModel.score(X_train,y_train))
print(finalModel.score(X_test,y_test))
```

```
0.8375
0.8875
```

Check whether to Accept or Reject the Model

Since the dataset is Unbalanced You need to Check for Non-Tolerable Areas

0 ---> Bad Customer
1 ---> Good Customer

GC 1 ----> BC 0(Non-Tolerable)
BC ----> GC ()

Precision()
Recall ()

```
from sklearn.metrics import confusion_matrix

#confusion_matrix(actualLabel, predictedLabel)
confusion_matrix(label,finalModel.predict(features))
```

```
array([[237,  20],
       [ 41, 102]])
```

```
from sklearn.metrics import classification_report
print(classification_report(label,finalModel.predict(features)))
```

```
  precision    recall  f1-score   support

           0       0.85      0.92      0.89       257
           1       0.84      0.71      0.77       143

    accuracy                           0.85       400
   macro avg       0.84      0.82      0.83       400

weighted avg       0.85      0.85      0.84       400
```

```
0.78 >= CL ---- approve model else reject model
```

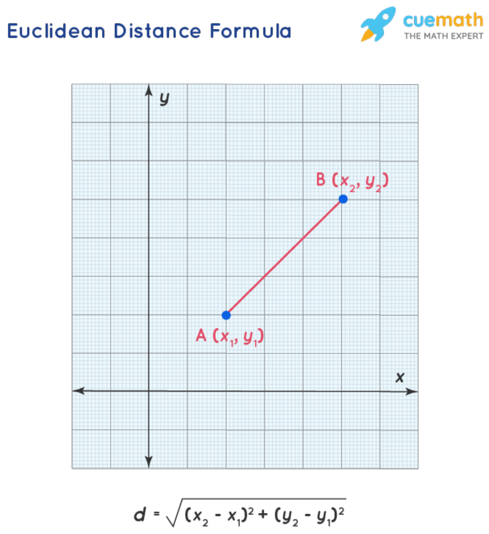
K nearest neighbor algorithm(KNN):

- Don’t use knn for huge datasets.
- Knn is applicable for both, regression(averaging) and classification(voting)

Training algorithm:
Copy the entire training data in an object.

Prediction algorithm:

1- Calculate the distance between the unknown point and all known points with:
  -Euclidean Distance Formula ➔ Default Formula
  -Manhattan Distance Formula



$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

2- Arrange the data in ascending order based on distance
3- Take first ‘k’ values and performing voting.

Use-case:

An Online Shopping Mall has provided this dataset. Your job is to create a model that can predict whether the customer will shop or not based on customer's age and estimated salary.

The process is the same like above with extra code for knn part

	# KNN Algorithm	from sklearn.neighbors import KNeighborsClassifier
		modelKNN = KNeighborsClassifier(n_neighbors=7)
		modelKNN.fit(X_train,y_train)

		modelKNN.score(X_train,y_train) 0.86875
	It is an overfitted model bcz Test score < train score	modelKNN.score(X_test,y_test) 0.7875