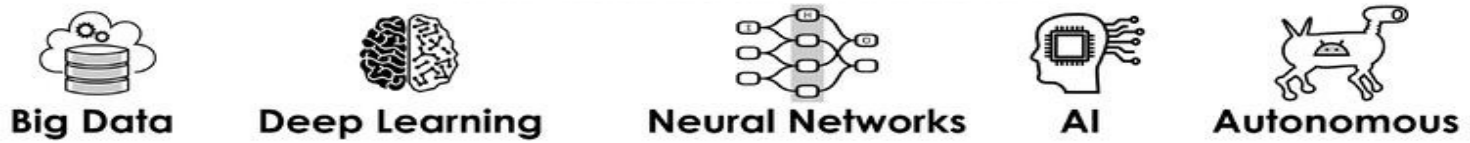
KASETSART UNIVERSITY
FACULTY OF SCIENCE AT SRIRACHA

MACHINE LEARNING



Linear Regression, Logistic Regression

01418496 Selected Topic in Computer Science

Chalothon Chootong (Ph.D.)

Department of Computer Science and Information, Faculty of Science at Sriracha, Kasetsart University Sriracha Campus

Kinds of Machine Learning

- **Supervised learning**
  - Each data point is labeled or associated with a category,
  - The goal of supervised learning is to study many labeled to be able to make predictions about future data points.

#	MAKE	CYLINDER VOLUME (CC)	YEAR	MILEAGE/KM	PRICE (RS)
1	TOYOTA	1300	2007	38000	410000
2	NISSAN	1500	2007	50000	325000
3	HONDA	1500	2005	59000	385000
4	TOYOTA	1000	2007	59000	360000
5	TOYOTA	1300	1989	62665	50000
6	TOYOTA	1500	2008	67000	615000
7	TOYOTA	1500	2008	69000	575000
8	TOYOTA	1490	2006	73000	450000
9	TOYOTA	1600	2006	82000	550000
10	TOYOTA	1000	2006	85000	325000
11	TOYOTA	1500	2000	113000	325000
12	TOYOTA	1500	2000	129000	218000
13	NISSAN	1500	2001	145000	195000

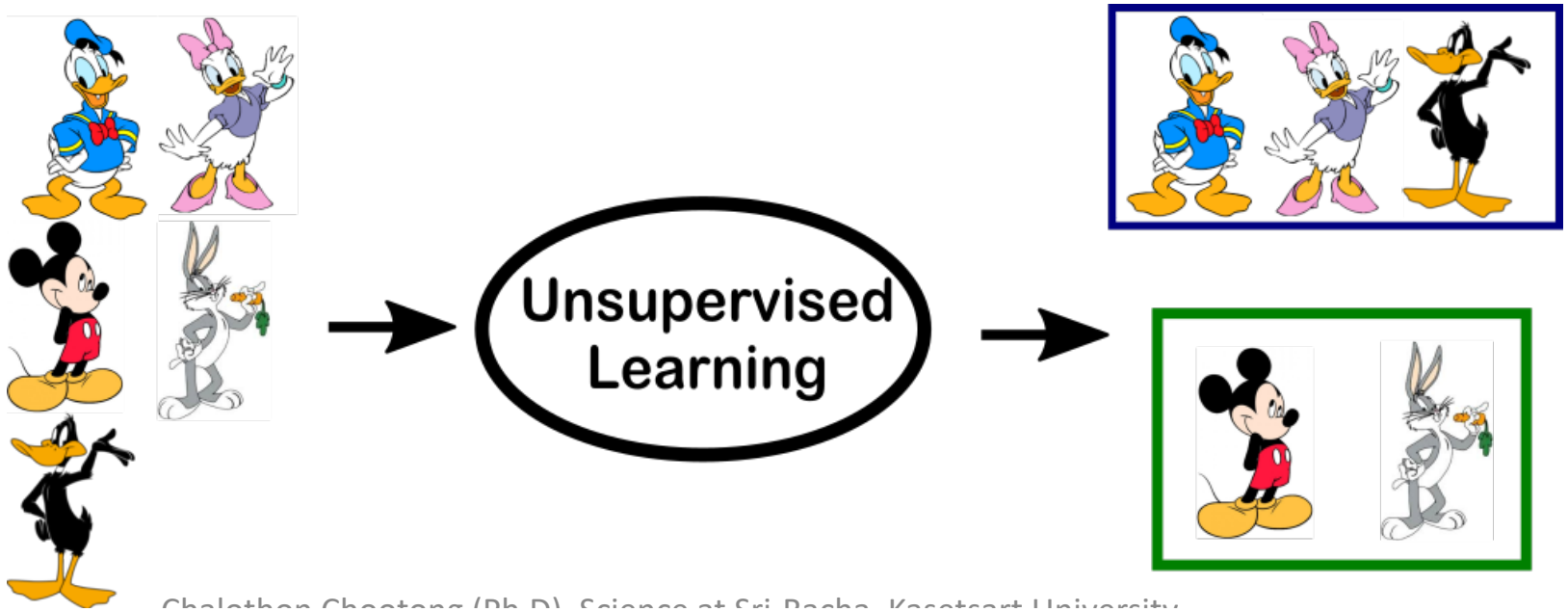
the sale price associated with a used car

Image	Label
	Cat
	Cat
	Dog
	Dog

https://www.packtpub.com/product/applied-deep-learning-with-keras/9781838555078

Kinds of Machine Learning

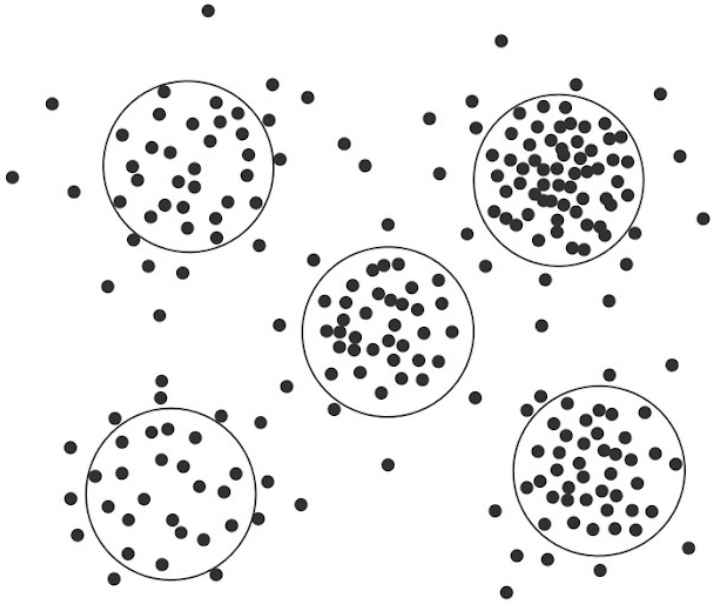
- **Unsupervised learning**
  - Data points have no labels associated with them,
  - The goal of an unsupervised learning algorithm is to organize the data in some way or to describe its structure.



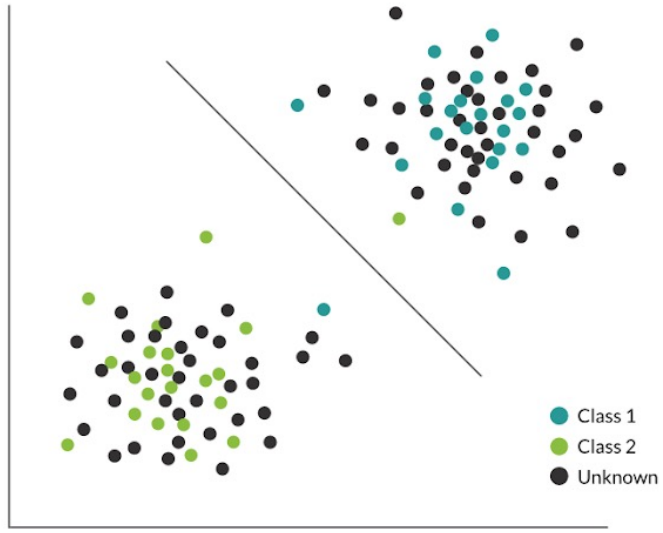
https://towardsdatascience.com/clustering-based-unsupervised-learning-8d705298ae51

Kinds of Machine Learning

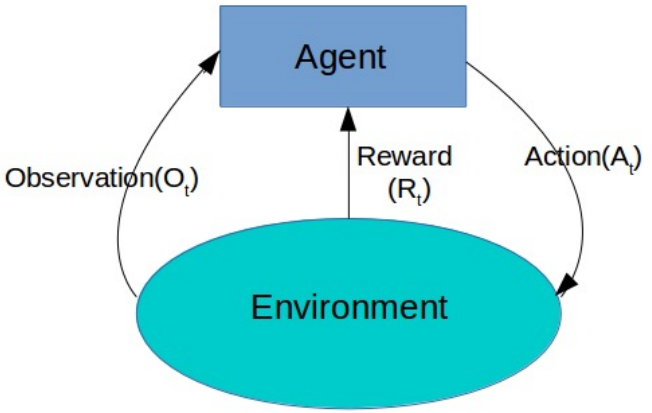
Unsupervised



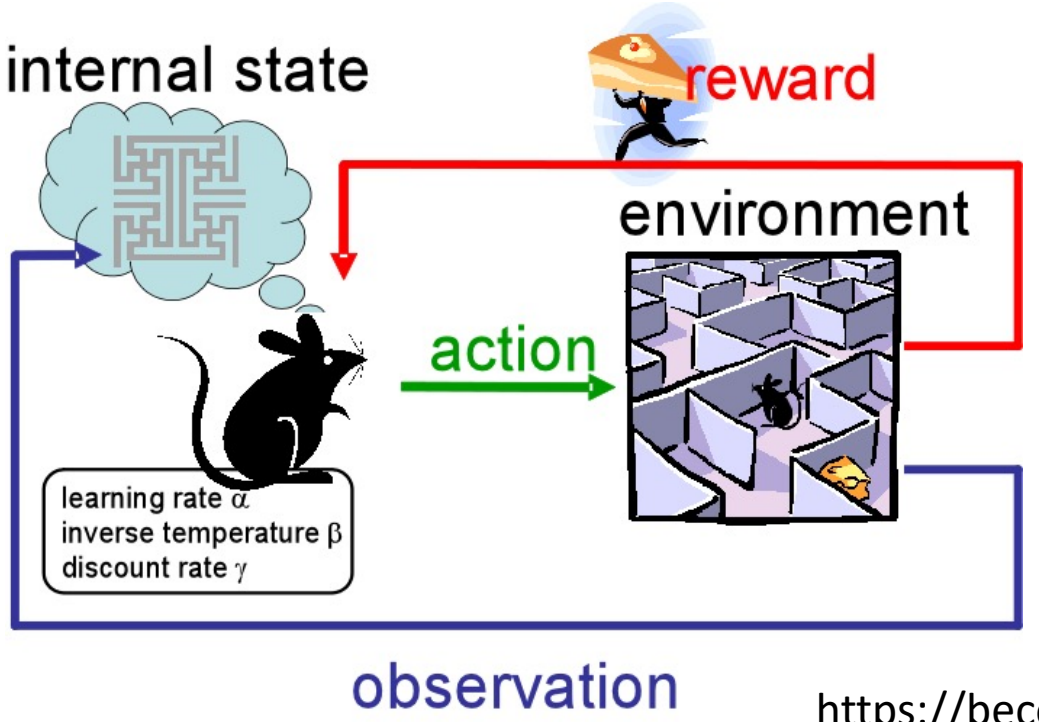
Supervised

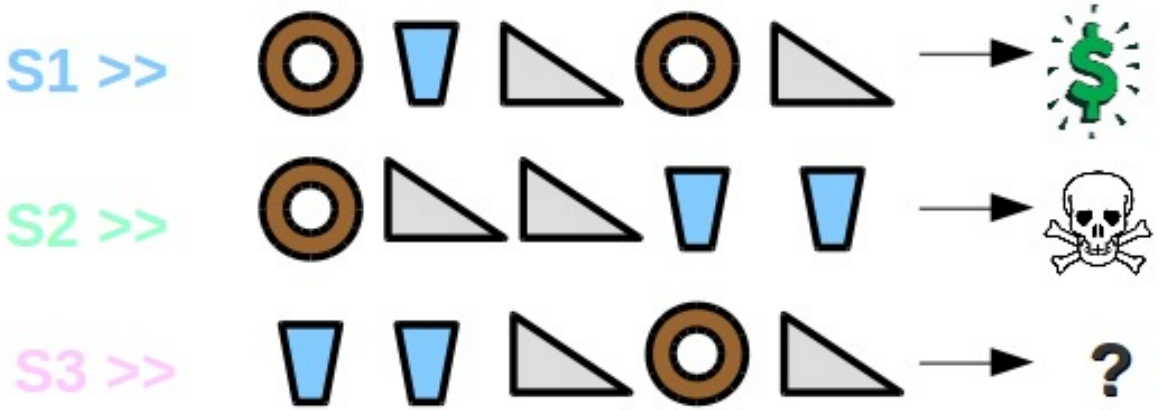


Kinds of Machine Learning



- **Reinforcement learning**
  - The algorithm gets to choose an action in response to each data point.

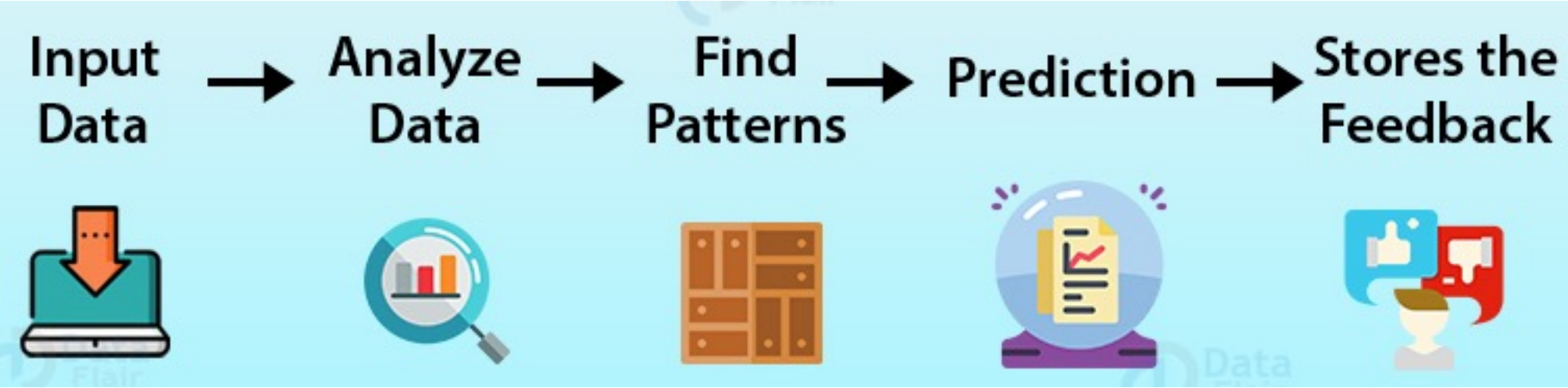




https://becominghuman.ai/the-very-basics-of-reinforcement-learning-154f28a79071

Kinds of Machine Learning

- Supervised Learning is “teach by example”
  - Here is some examples, now learn patterns in these example
- Reinforcement Learning is “teach by experience”
  - Here is a world, now learn patterns by exploring it.



Scikit-Learn

- Scikit-Learn
  - It contains useful algorithms that can easily be implemented for the purposes of classification and other machine learning tasks.
  - Scikit-Learn uses SciPy as a foundation, so this base stack of libraries must be installed before Scikit-Learn can be utilized

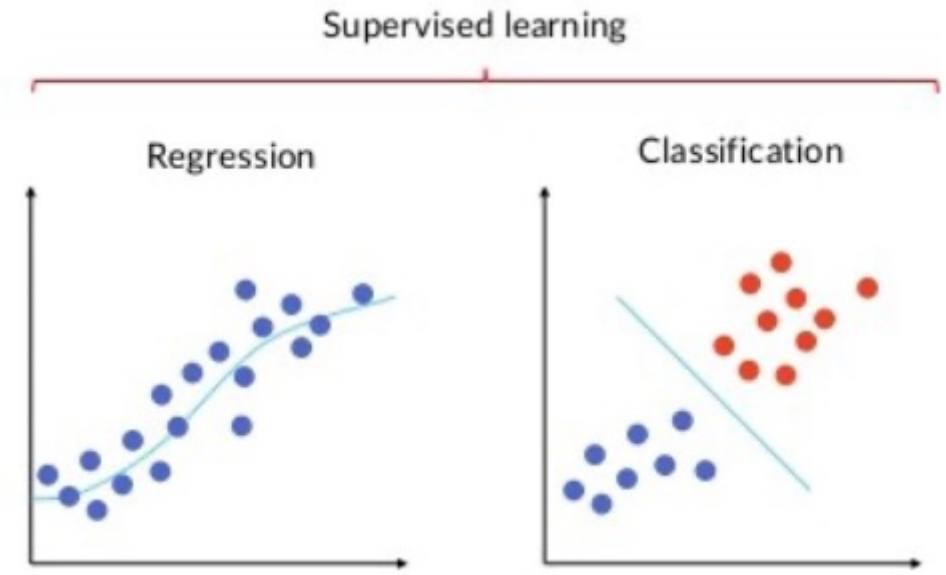
Scikit-Learn (Defining Technical Terms)

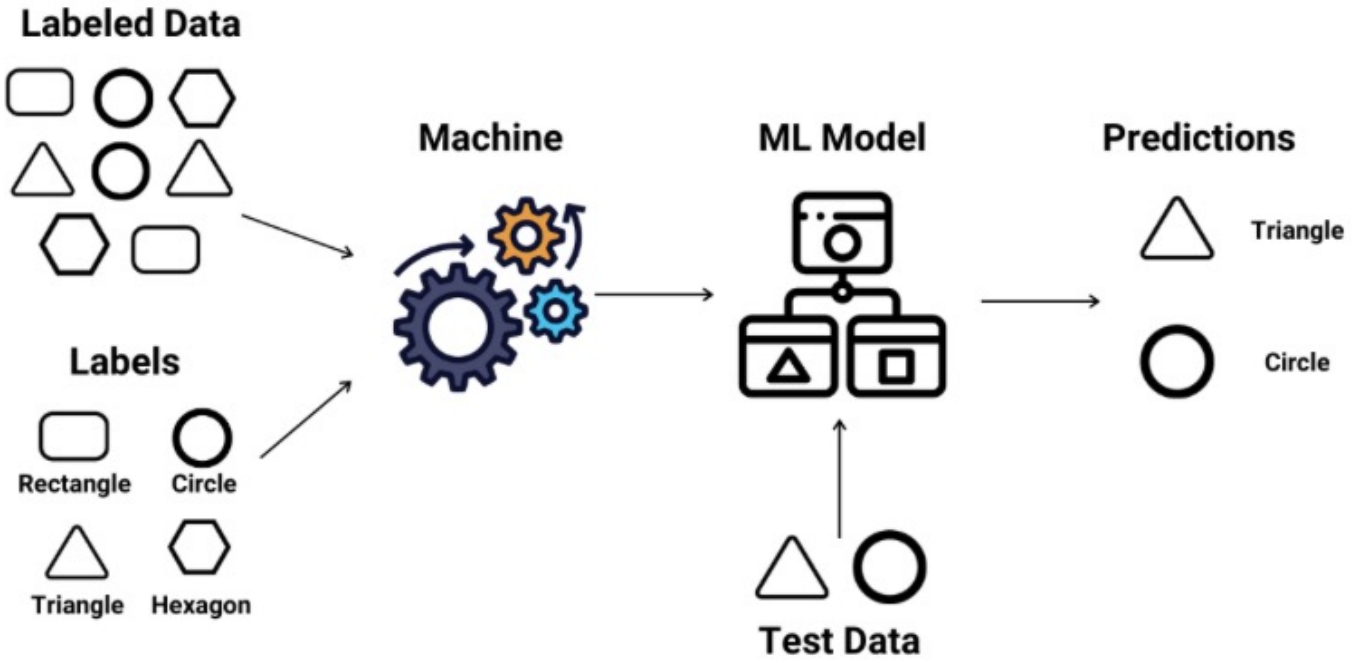
- "features": The inputs into the machine learning framework.
  - When these features are fed into a machine learning framework the network tries to discern relevant patterns between the features.
  - These pattens are then used to generate the outputs of the framework.
- "labels": The output of the framework (category, type etc.)
- "Supervised Learning": Data are fed to network is already labeled, with the important features/attributes already separated into distinct categories beforehand.

Scikit-Learn (Defining Technical Terms)

- The training process is the process of feeding data into a network and let it learn the patterns of the data.
  - For a supervised classification task is passed both the features and labels of the training data.
- The testing process is where the patterns that the network has learned are tested.
  - During testing, the network is only fed features.
  - The features are given to the network, and the network must predict the labels.

Supervised Learning





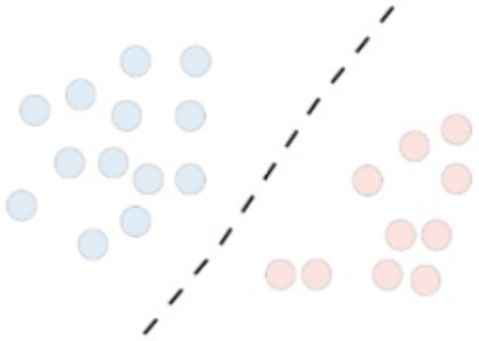
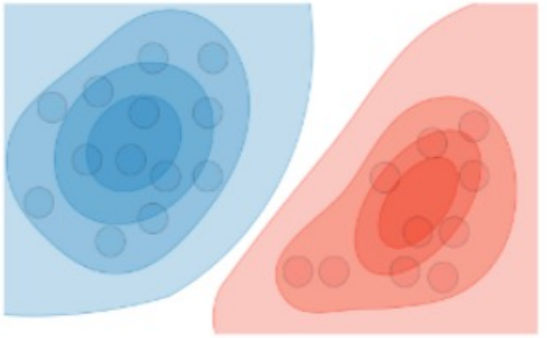
https://www.enjoyalgorithms.com/blogs/supervised-unsupervised-and-semisupervised-learning

Supervised Learning

- Type of prediction:

	Regression	Classifier
Outcome	Continuous	Class
Examples	Linear regression	Logistic regression, SVM, Naive Bayes

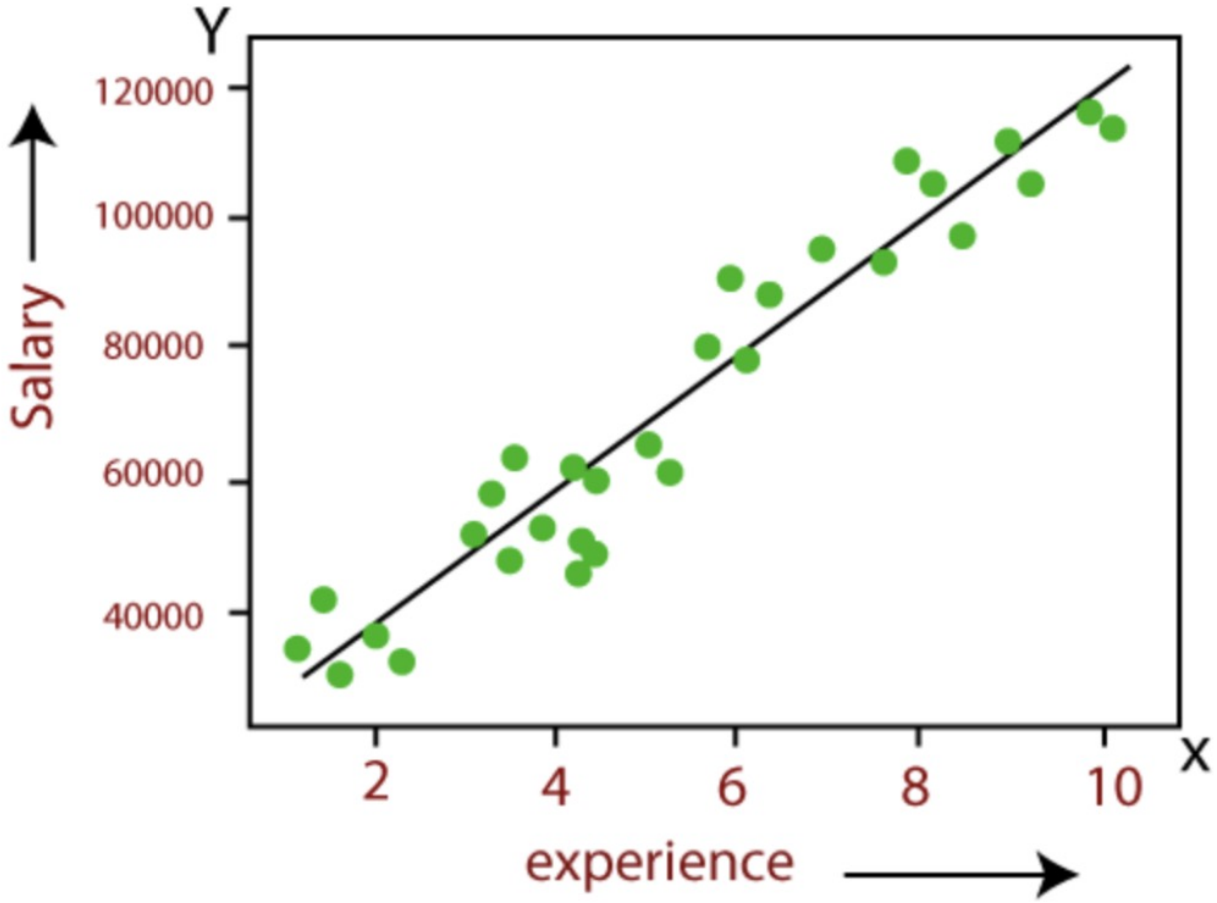
- Type of model:

	Discriminative model	Generative model
Goal	Directly estimate $P(y\|x)$	Estimate $P(x\|y)$ to deduce $P(y\|x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		

Different Types of Classifiers

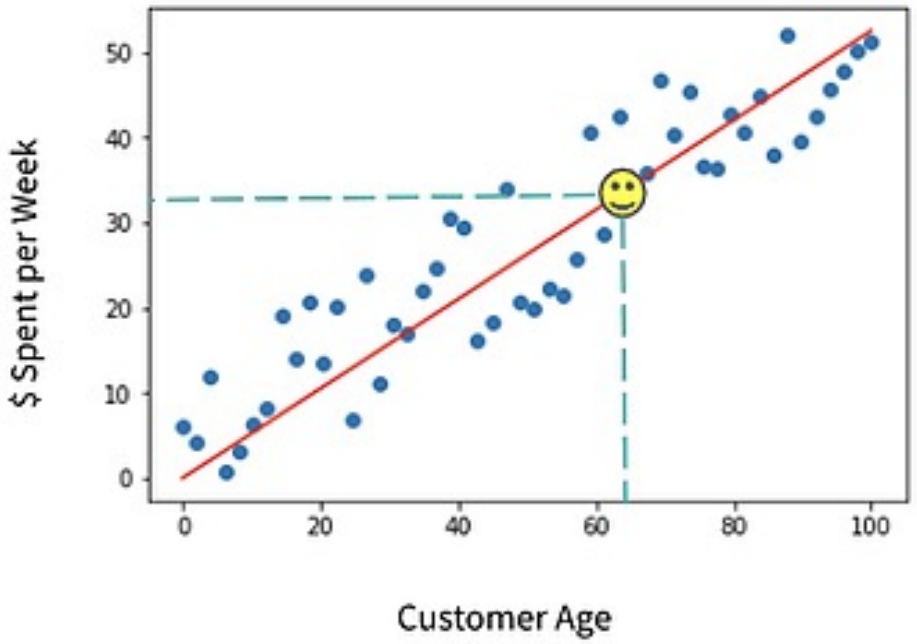
- Linear models
  - Linear regression
  - Logistic regression
- K-Nearest Neighbors
- Decision Tree Classifiers/Random Forests
- Support Vector Machines
- Naive Bayes

Linear Regression



Types of linear models:

- **Linear regression**, which is used for regression (numerical predictions).
- **Logistic regression**, which is used for classification (categorical predictions).



Linear regression

$$y = mx + b$$

$$y = b + m_1 * x$$

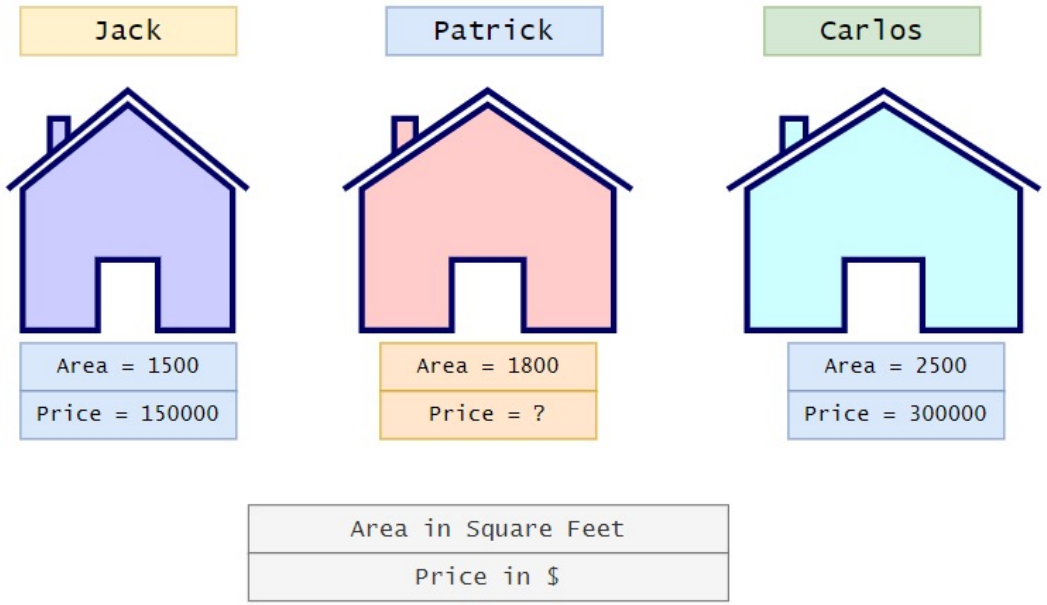
$$\$ \text{ spent this week} = b + m_1 * age + e$$

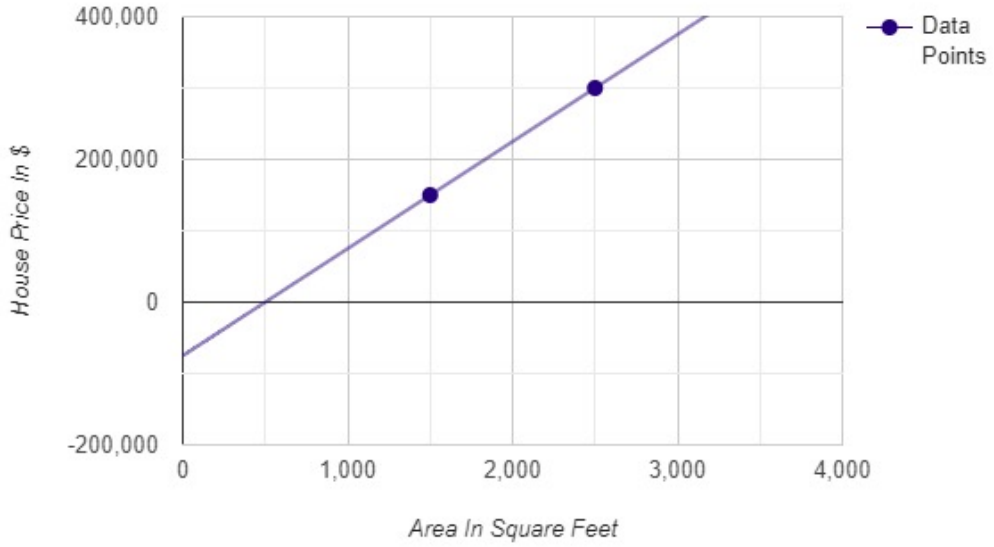
$$\$ \text{ spent this week} = b + m_1 * age + m_2 * has\_kids + \ldots + e$$

$$m = slope\ of\ the\ line$$

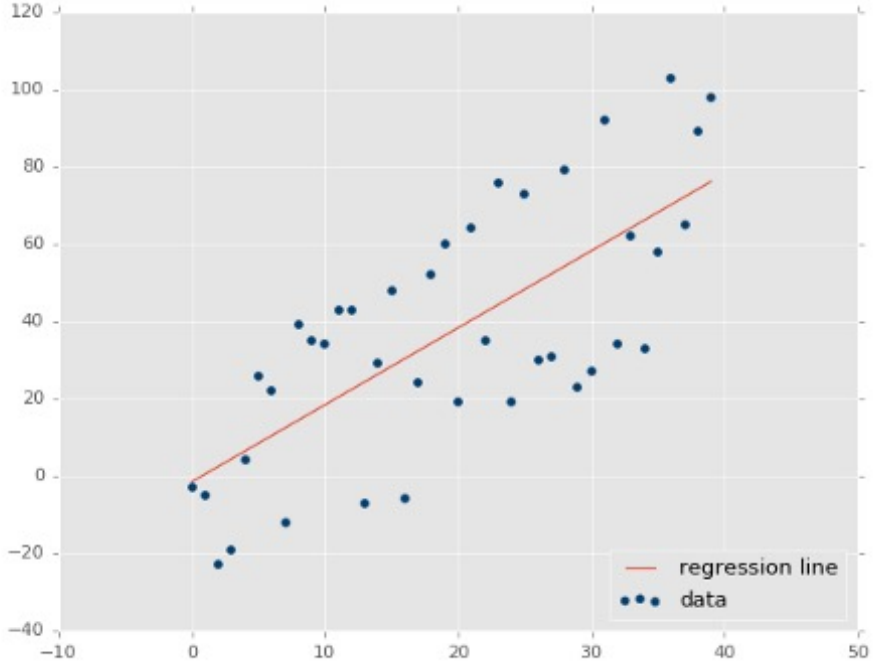
$$b = Y - intercept\ of\ the\ line$$

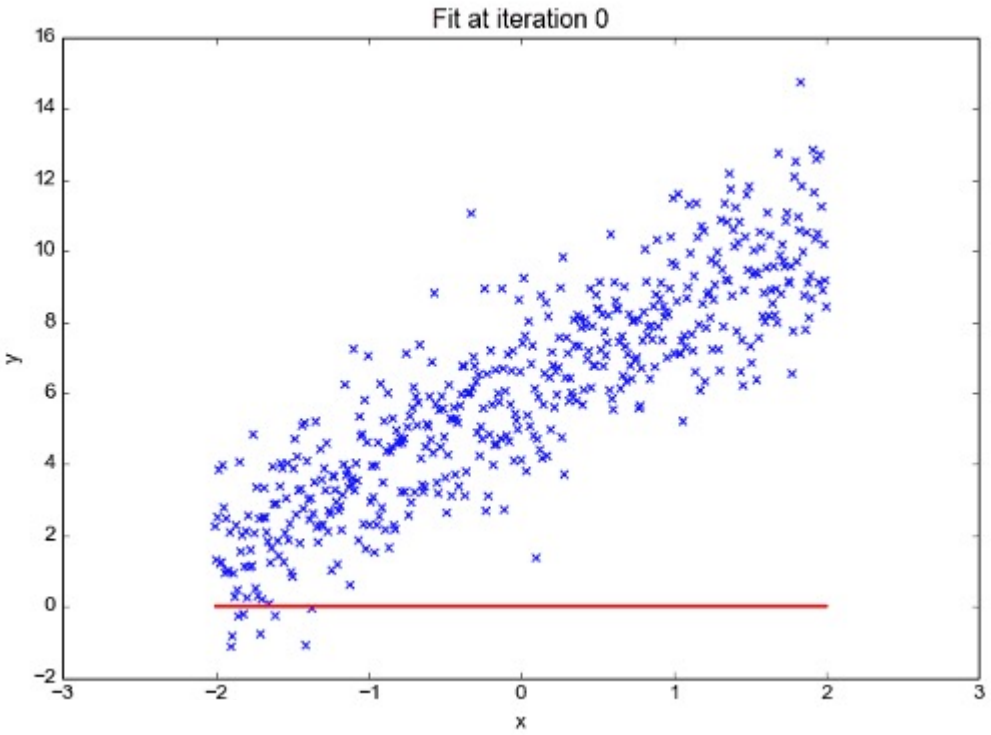
Example of Linear Regression

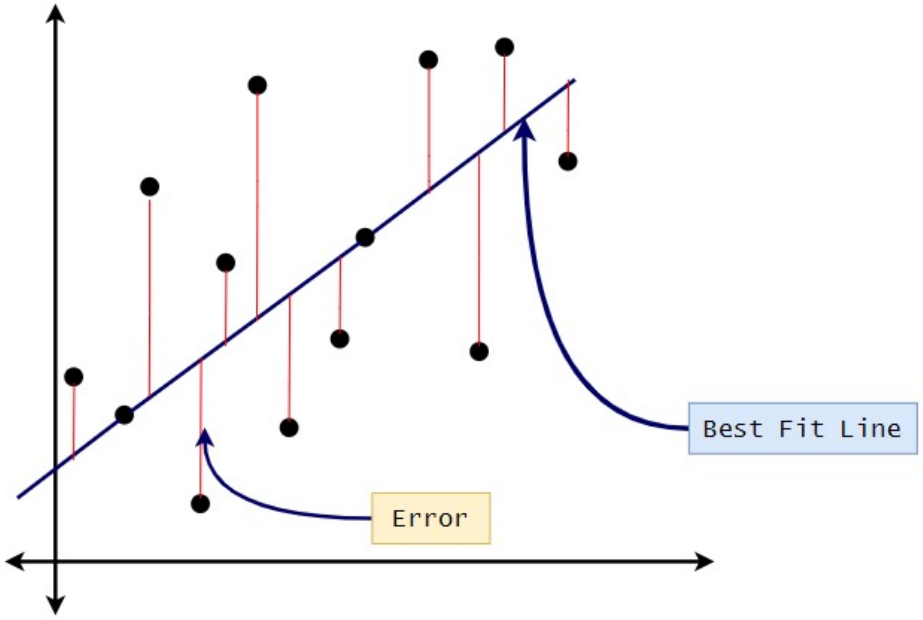




https://pub.towardsai.net/calculating-simple-linear-regression-and-linear-best-fit-an-in-depth-tutorial-with-math-and-python-804a0cb23660


regression line
data

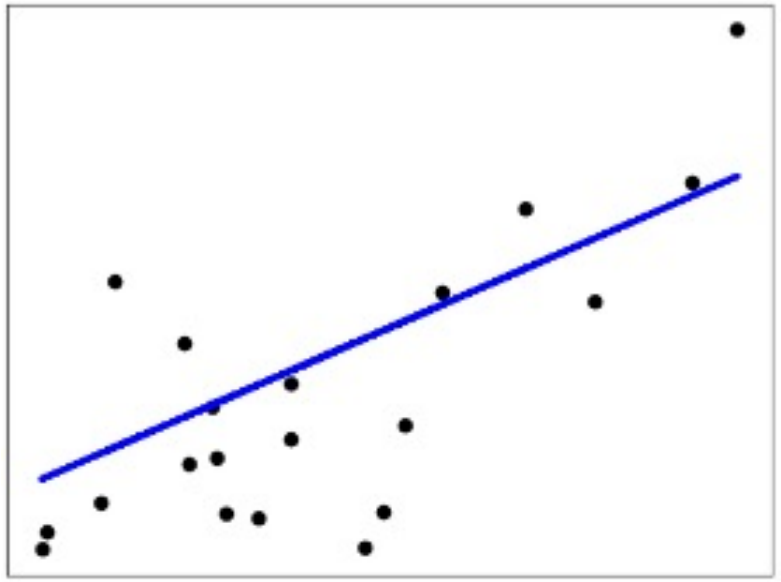

Fit at iteration 0
x
y


Best Fit Line
Error

Linear Regression

- Linear Model generate a formula to create a best-fit line to predict unknown values.

$$\hat{y}(w, x) = w_0 + w_1 x_1 + \ldots + w_p x_p$$



```
from sklearn import linear_model
reg = linear_model.LinearRegression()
reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
print(reg.coef_)
```

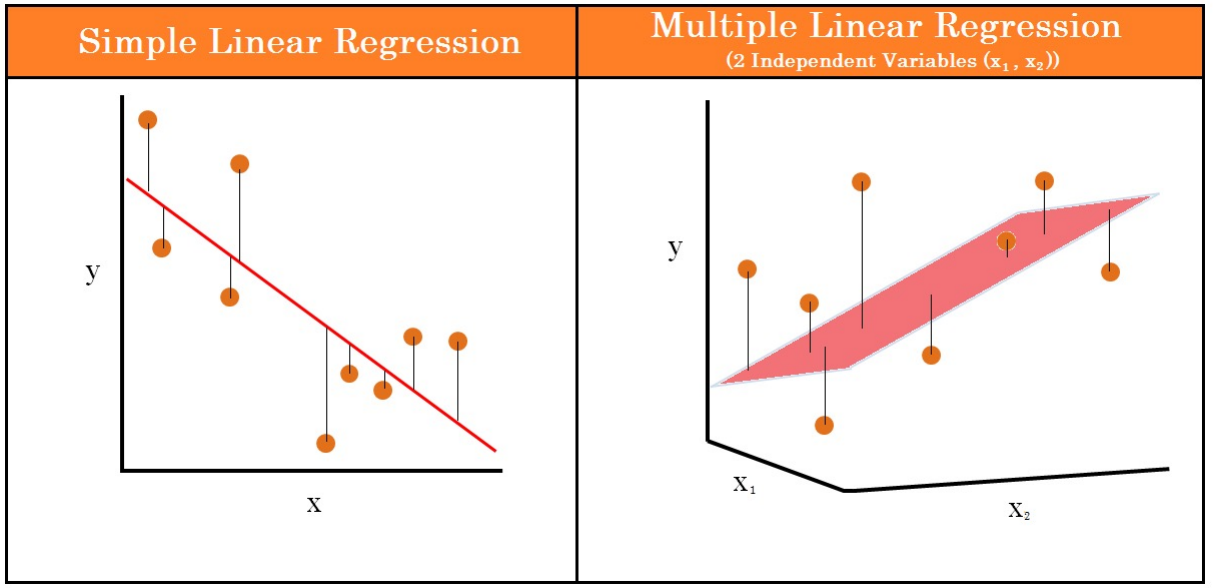
Simple linear regression vs. Multiple linear regression

Simple Linear Regression

$$y = b_0 + b_1 * x_1$$

Multiple Linear Regression

$$y = b_0 + b_1 * x_1 + b_2 * x_2 + \ldots + b_n * x_n$$



https://www.superdatascience.com/blogs/regression-classification-multiple-linear-regression/

https://www.keboola.com/blog/linear-regression-machine-learning

sklearn.linear_model.LinearRegression

class sklearn.linear_model.**LinearRegression**(*, *fit_intercept=True*, *normalize=False*, *copy_X=True*, *n_jobs=None*, *positive=False*)

[source]

Ordinary least squares Linear Regression.

LinearRegression fits a linear model with coefficients w = (w1, ..., wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

Parameters:

fit_intercept : *bool, default=True*

Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

normalize : *bool, default=False*

This parameter is ignored when `fit_intercept` is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, please use `StandardScaler` before calling `fit` on an estimator with `normalize=False`.

copy_X : *bool, default=True*

If True, X will be copied; else, it may be overwritten.

n_jobs : *int, default=None*

The number of jobs to use for the computation. This will only provide speedup for n_targets > 1 and sufficient large problems. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

positive : *bool, default=False*

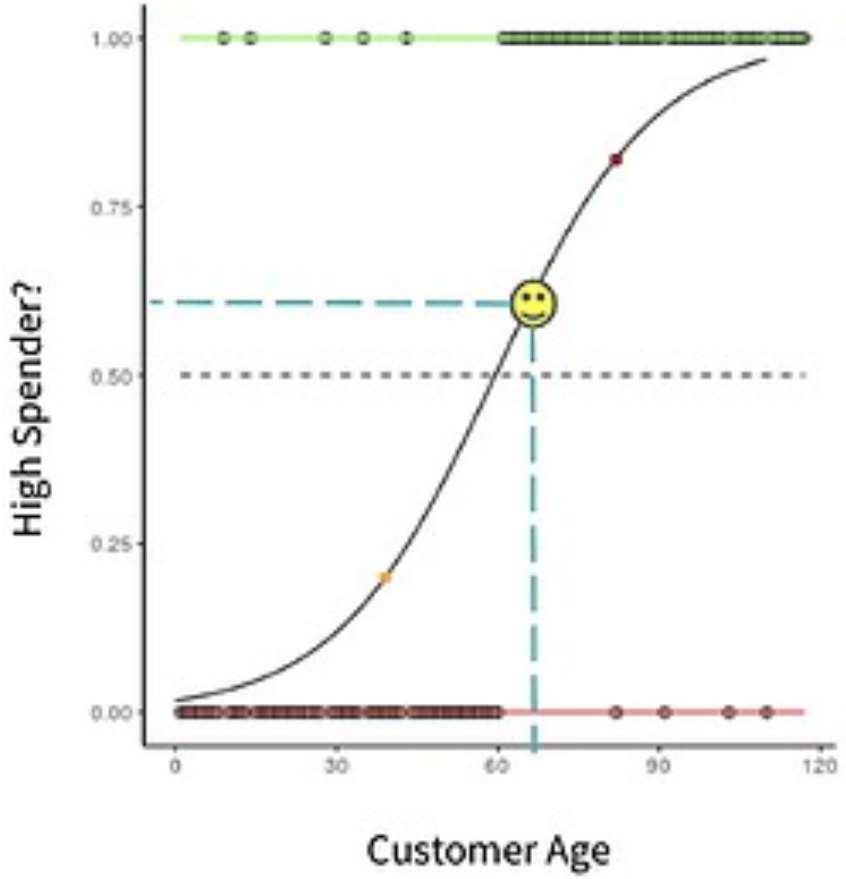
When set to `True`, forces the coefficients to be positive. This option is only supported for dense arrays.

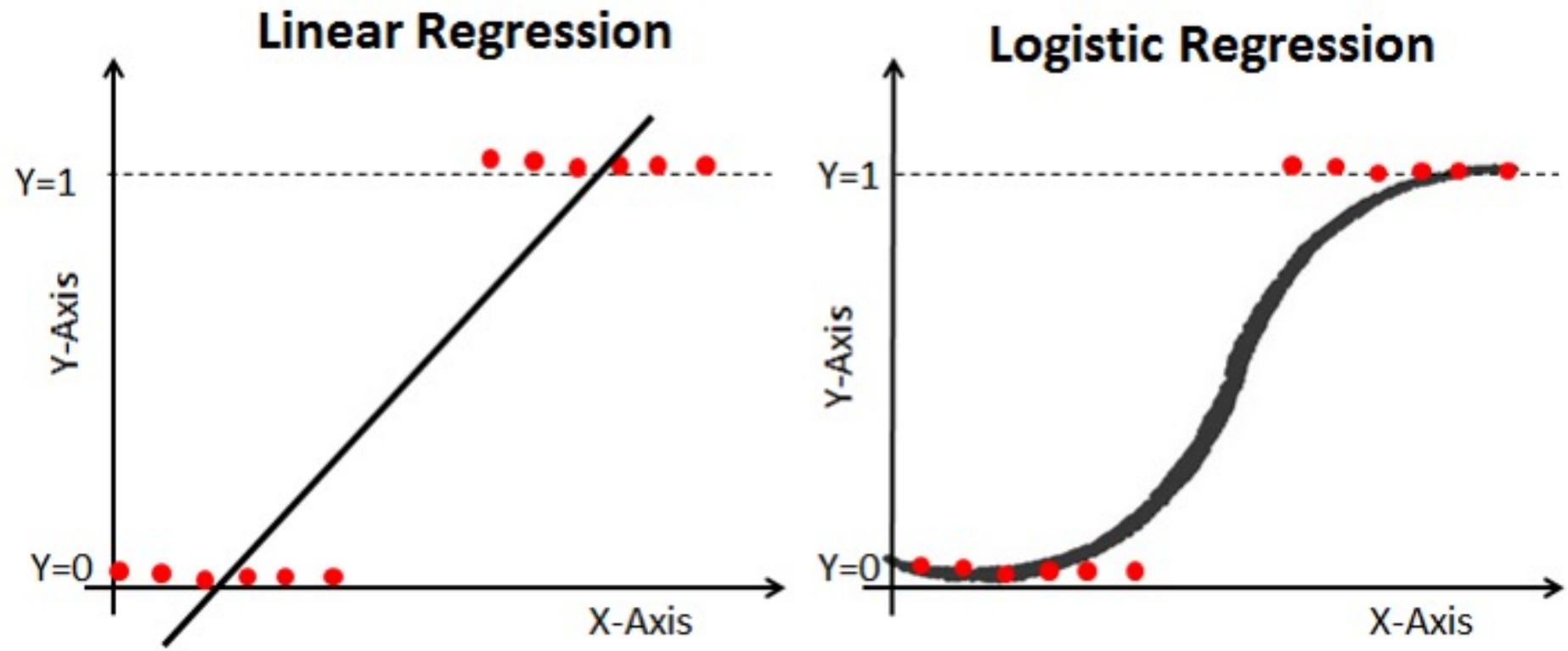
New in version 0.24.

Logistic Regression

- **Logistic Regression** outputs predictions about test data points on a binary scale, zero or one. If the value of something is 0.5 or above, it is classified as belonging to class 1, while below 0.5 if is classified as belonging to 0.

- จุดดำคือข้อมูลที่ใช้สำหรับ train, S-Shape (also call Sigmoid curve) คือ เส้นที่ดีที่สุดที่มาสามารถใช้ในการแบ่งข้อมูล
- เราสามารถใช้ S-Shape ในการทำนายความน่าจะเป็นของข้อมูลที่จะอยู่ใน class ใด class หนึ่ง
- Threshold คือ 50%





https://ichi.pro/th/kar-thakhwam-kheaci-logistic-regression-laea-building-model-ni-python-25643771585192

sklearn.linear_model.LogisticRegression

class `sklearn.linear_model.` **`LogisticRegression`**(*penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*) [source]

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)

This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. **Note that regularization is applied by default**. It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization with primal formulation, or no regularization. The 'liblinear' solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty. The Elastic-Net regularization is only supported by the 'saga' solver.

Read more in the User Guide.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Parameters:

penalty : *{'l1', 'l2', 'elasticnet', 'none'}, default='l2'*
Used to specify the norm used in the penalization. The 'newton-cg', 'sag' and 'lbfgs' solvers support only l2 penalties. 'elasticnet' is only supported by the 'saga' solver. If 'none' (not supported by the liblinear solver), no regularization is applied.

New in version 0.19: l1 penalty with SAGA solver (allowing 'multinomial' + L1)

dual : *bool, default=False*
Dual or primal formulation. Dual formulation is only implemented for l2 penalty with liblinear solver. Prefer dual=False when n_samples > n_features.

tol : *float, default=1e-4*
Tolerance for stopping criteria.

C : *float, default=1.0*
Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values

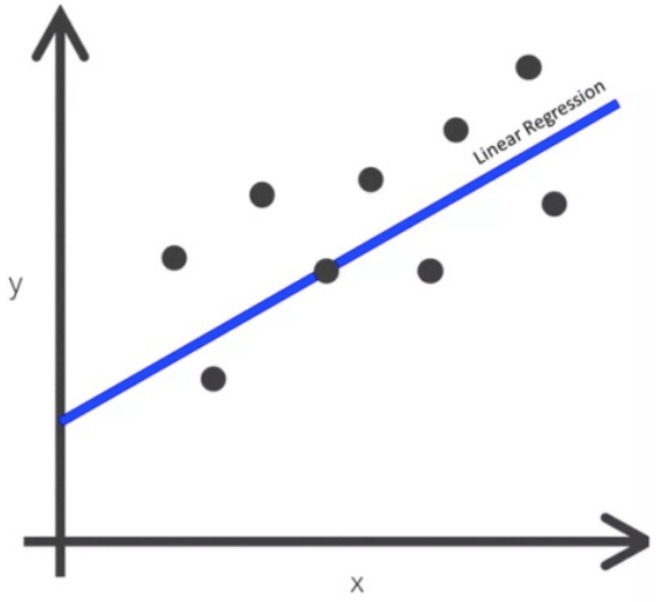
logreg = LogisticRegression(C=1.5, solver='lbfgs', max_iter=1000)

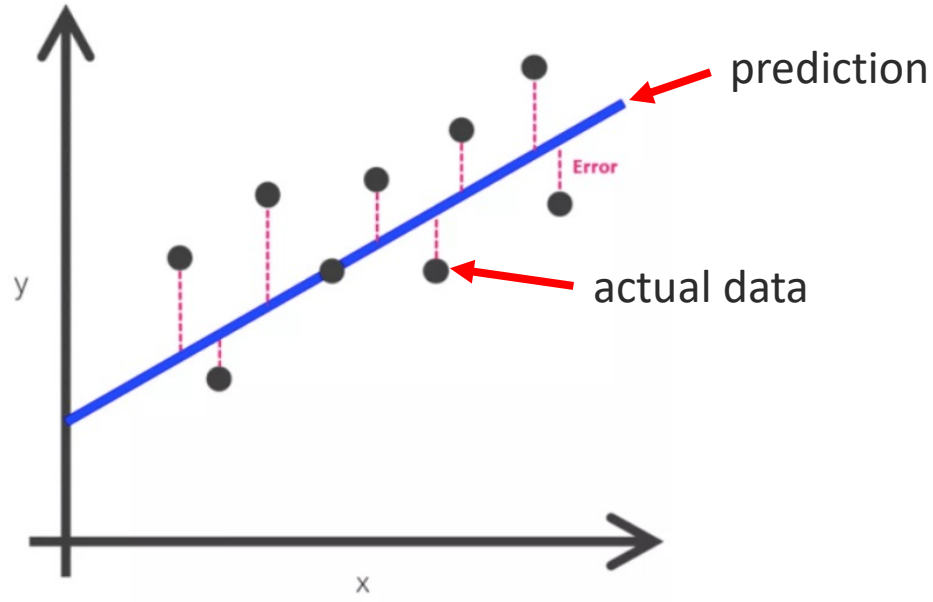
- Cfloat, default=1.0
  - Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization
- solver{‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}, default=’lbfgs’
  - Algorithm to use in the optimization problem.
  - For small datasets, ‘liblinear’ is a good choice, whereas ‘sag’ and ‘saga’ are faster for large ones.
  - For multiclass problems, only ‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’ handle multinomial loss; ‘liblinear’ is limited to one-versus-rest schemes.
    - ‘newton-cg’, ‘lbfgs’, ‘sag’ and ‘saga’ handle L2 or no penalty
    - ‘liblinear’ and ‘saga’ also handle L1 penalty
    - ‘saga’ also supports ‘elasticnet’ penalty
    - ‘liblinear’ does not support setting penalty='none’
- max_iterint, default=100
  - Maximum number of iterations taken for the solvers to converge.

Measurement

Error Measurement

- Regression Model (เพื่อทำนายตัวแปร Y แบบตัวเลข)
  - Y_{actual} , $Y_{prediction}$
  - error = prediction – actual ➔ “Loss function”





MAE, MSE

- “Mean Absolute Error”

$$MAE = \frac{1}{n} * \sum |prediction - actual|$$

- “Mean Squared Error”

$$MSE = \frac{1}{n} * \sum (prediction - actual)^2$$

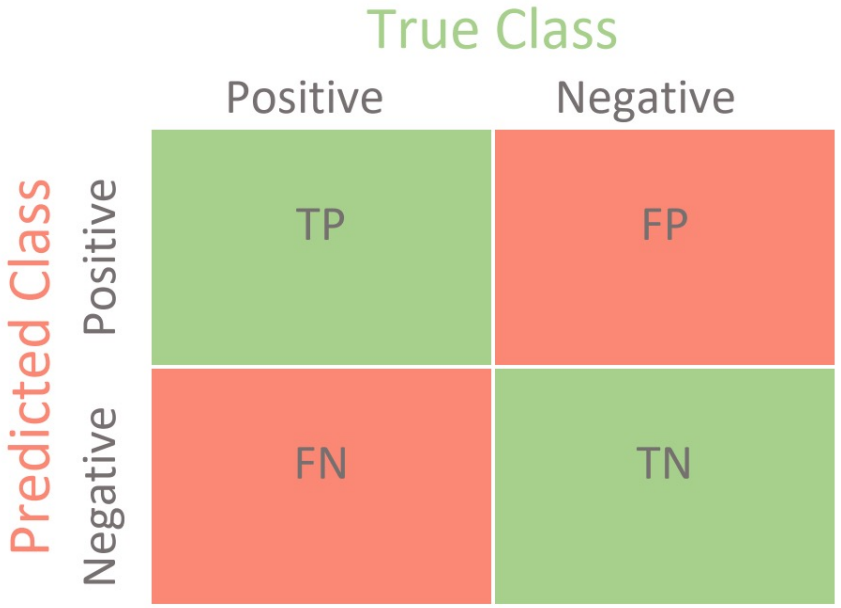
RMSE

- **Root Mean Square Error** (RMSE) is the standard deviation of the residuals (prediction errors).

$$RMSE = \sqrt{\frac{1}{n} * \sum (prediction - actual)^2}$$

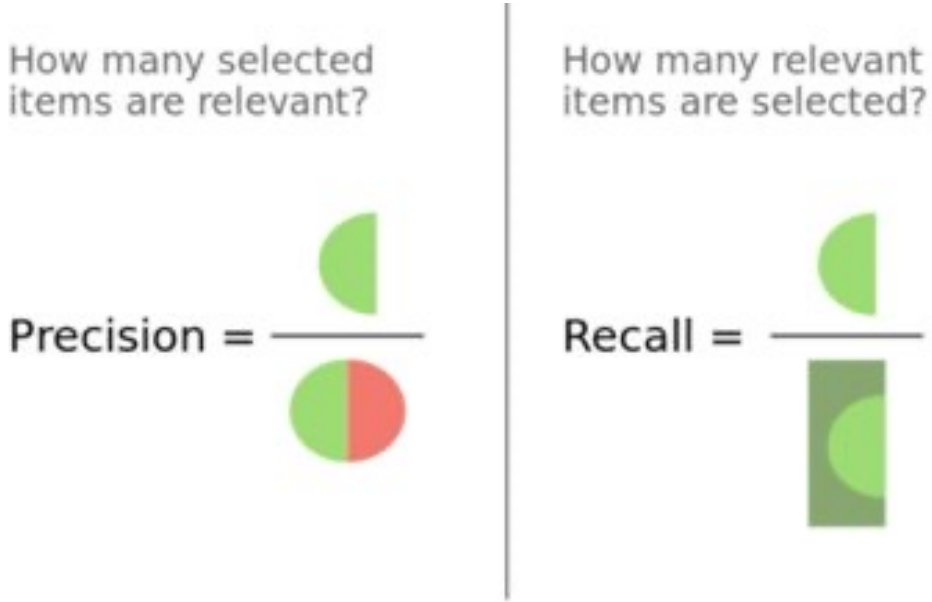
Confusion Matrix

- TP = True Positive – The model predicted the positive class correctly, to be a positive class.
- FP = False Positive – The model predicted the negative class incorrectly, to be a positive class.
- FN = False Negative – The model predicted the positive class incorrectly, to be the negative class.
- TN = True Negative – The model predicted the negative class correctly, to be the negative class.



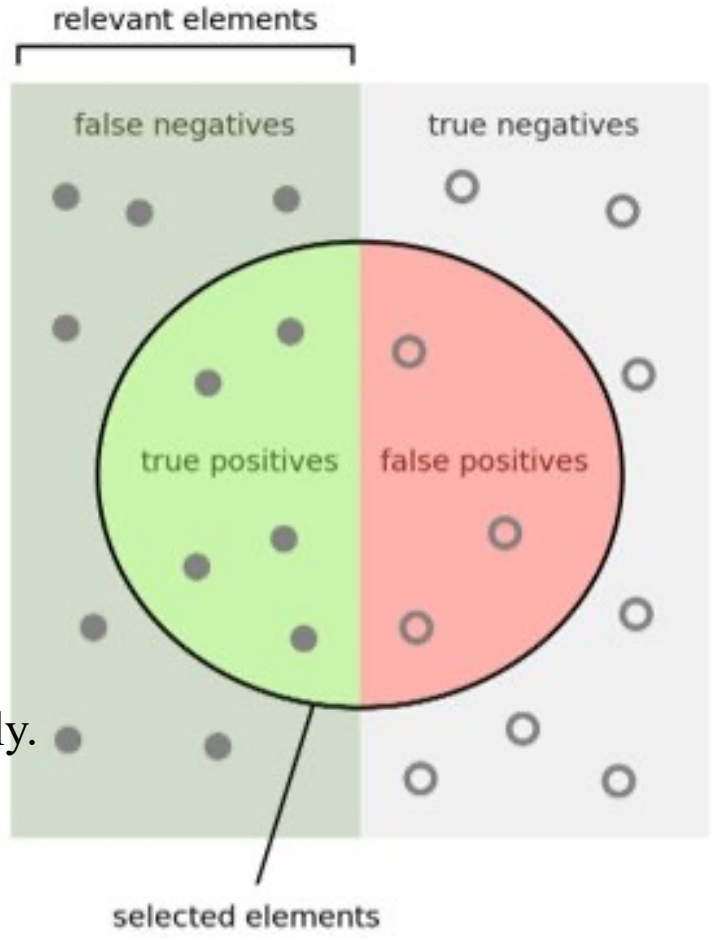
Statistics Computed from Confusion Matrix

- Precision and Recall



how many were predicted correctly

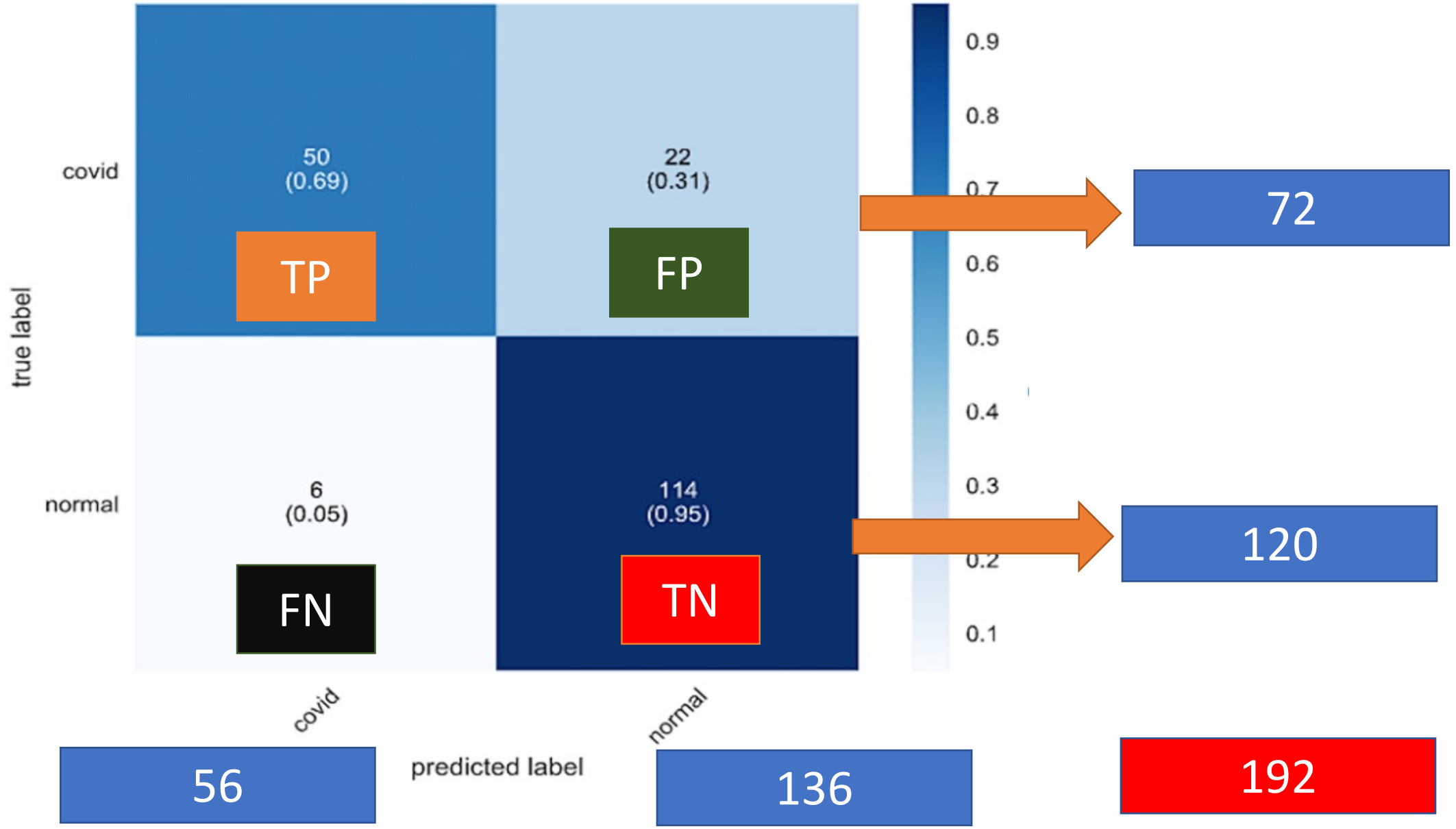
how many instances were identified correctly.



Precision = TP / (TP + FP)

Recall = TP / (TP + FN)

F-Score = (2 * Recall * Precision) / (Recall + Precision)



https://www.researchgate.net/figure/Confusion-Matrix-for-Covid-19-Detection-using-CNN-with-actual-data_fig4_341401062

- **Accuracy**
  - (TP+TN)/total = (50+114)/192 = 0.854
- **Error Rate**
  - (FP+FN)/total = (22+6)/192 = 0.145
- **True Positive Rate:**
  - TP/(TP+FN)= 50/(50+6) = 0.892
- **False Positive Rate:**
  - FP/(FP+TN) = 22/(22+114) = 0.161

Precision and Recall

- Precision (ความน่าจะเป็นที่ model ทำนายถูก)
  - TP / (TP + FP) = 50/(50+70) = 0.416
- Recall (ความน่าจะเป็นที่ model สามารถทำนายได้)
  - TP / (TP + FN) = 50/(50+6) = 0.892
- F-Score
  - (2 * Recall * Precision) / (Recall + Precision)
  - (2 * 0.892*0.416) / (0.892+0.416) = 0.567

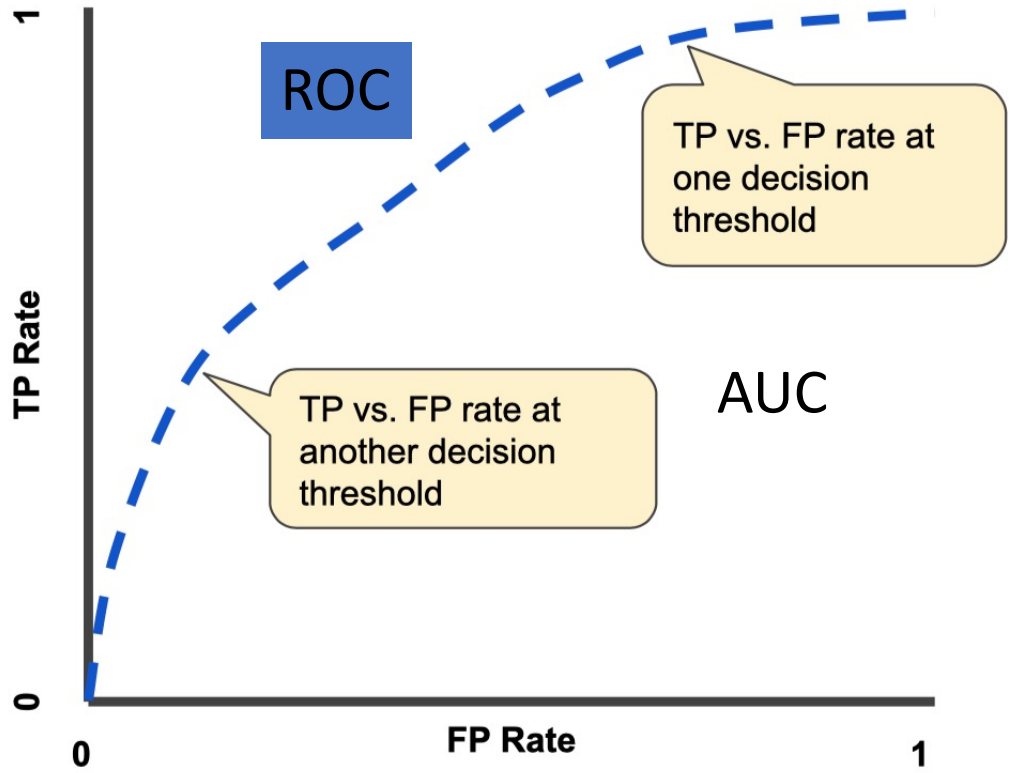
ROC Curve and AUC

- **ROC curve** (**receiver operating characteristic curve**) is a graph showing the performance of a classification model at all classification thresholds
  - This curve plots two parameters:
    - **True Positive Rate** (TPR)

$$TPR = \frac{TP}{TP + FN}$$

    - **False Positive Rate** (FPR)

$$FPR = \frac{FP}{FP + TN}$$



AUC - ROC Curve

- ROC is a probability curve for different classes.
- ROC tells us how good the model is for distinguishing the given classes, in terms of the predicted probability.
  - The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR)

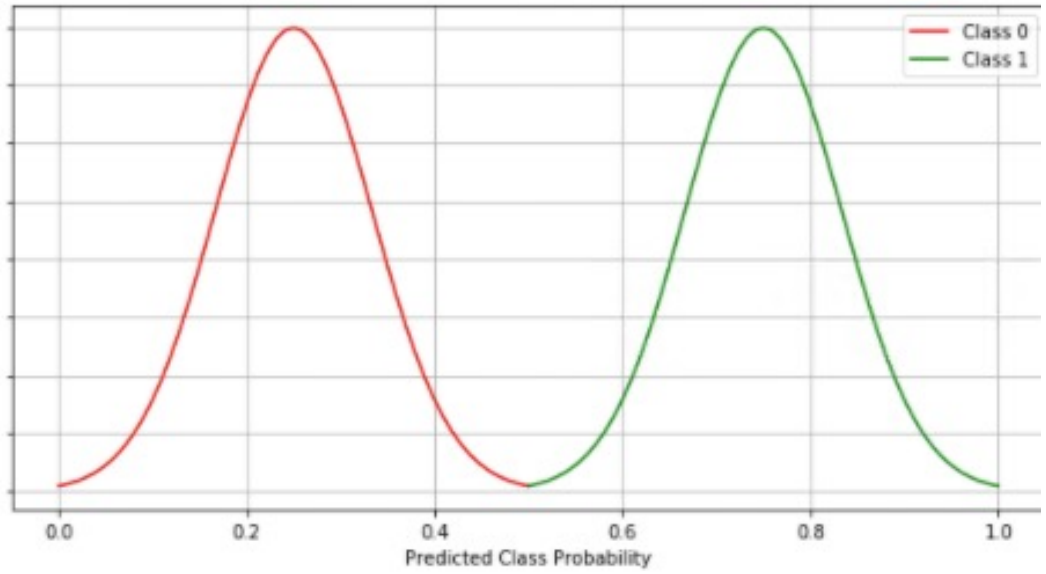
$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$

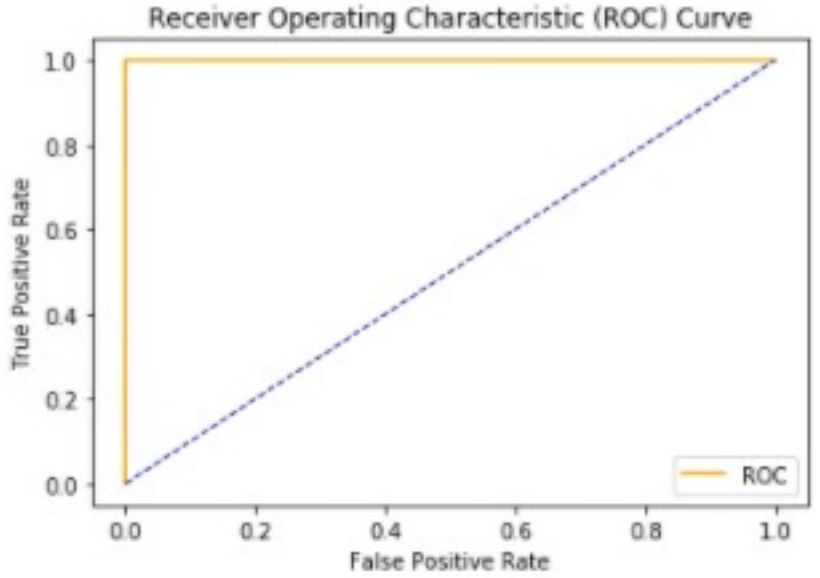
A			B		
TP=63	FP=28	91	TP=77	FP=77	154
FN=37	TN=72	109	FN=23	TN=23	46
100	100	200	100	100	200
TPR = 0.63			TPR = 0.77		
FPR = 0.28			FPR = 0.77		

AUC - ROC Curve

- Different Scenarios with ROC Curve and Model Selection
  - Scenario #1 (Best Case Scenario)



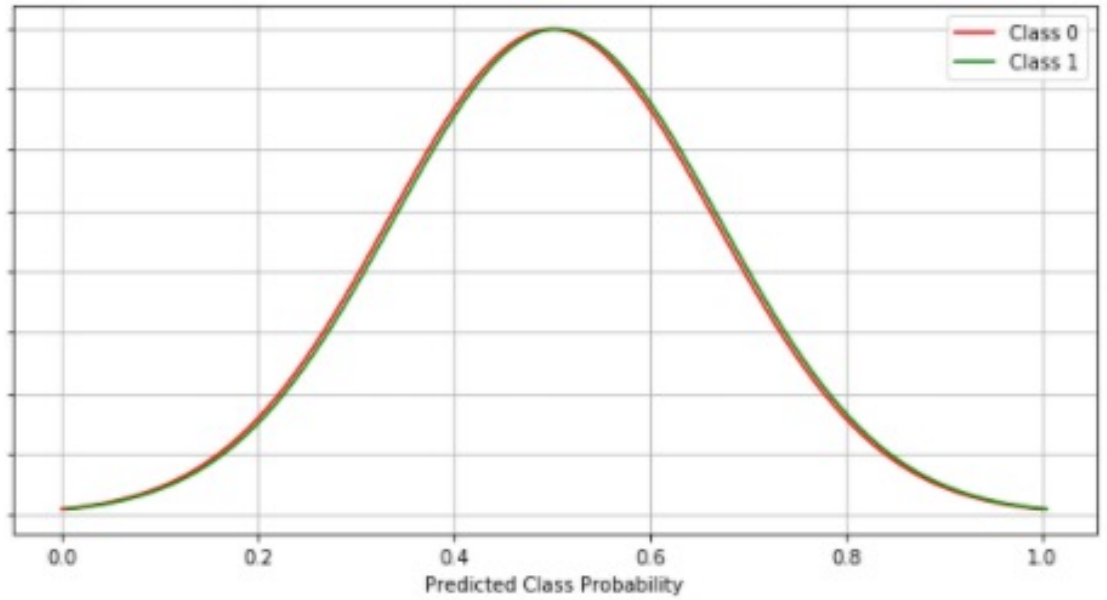
The AUC-ROC curve for this case is as below.



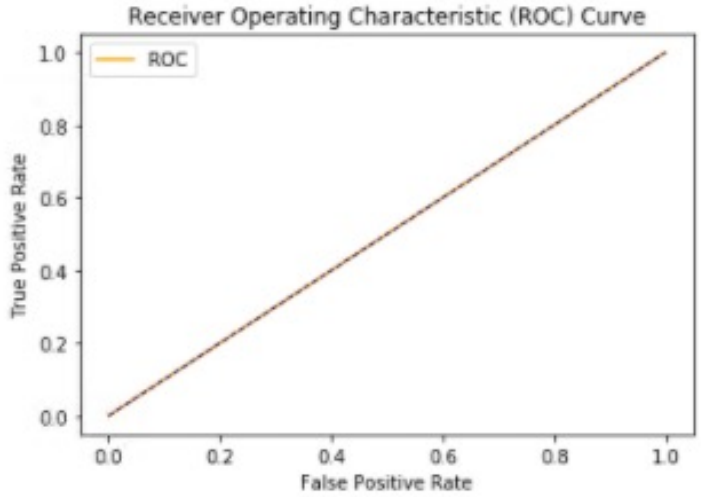
we have a clear distinction between the two classes as a result, we have the AUC of 1

AUC - ROC Curve

- Scenario #2 (Random Guess)



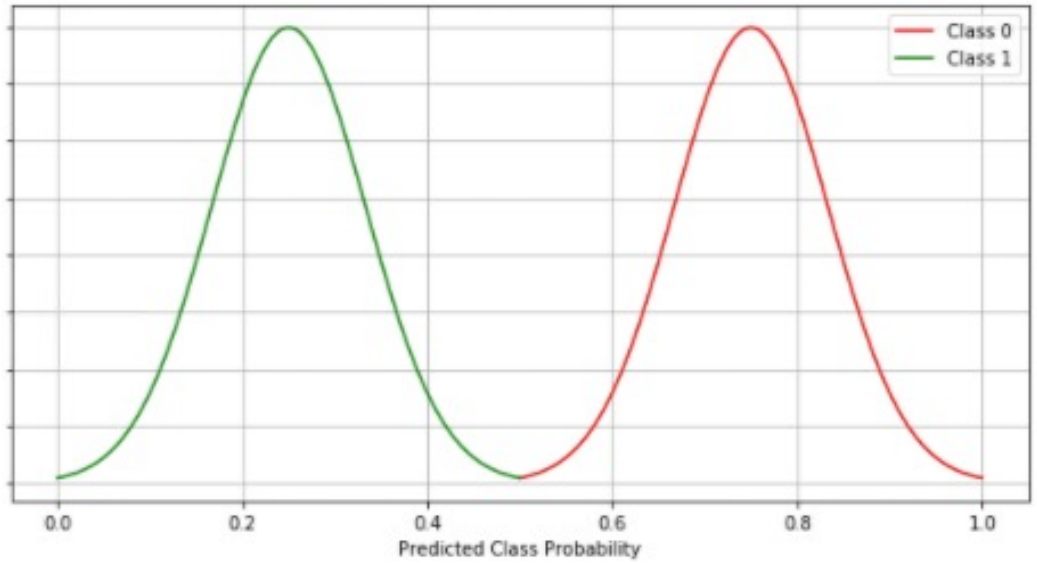
We can see there is no clear discrimination between the two classes.

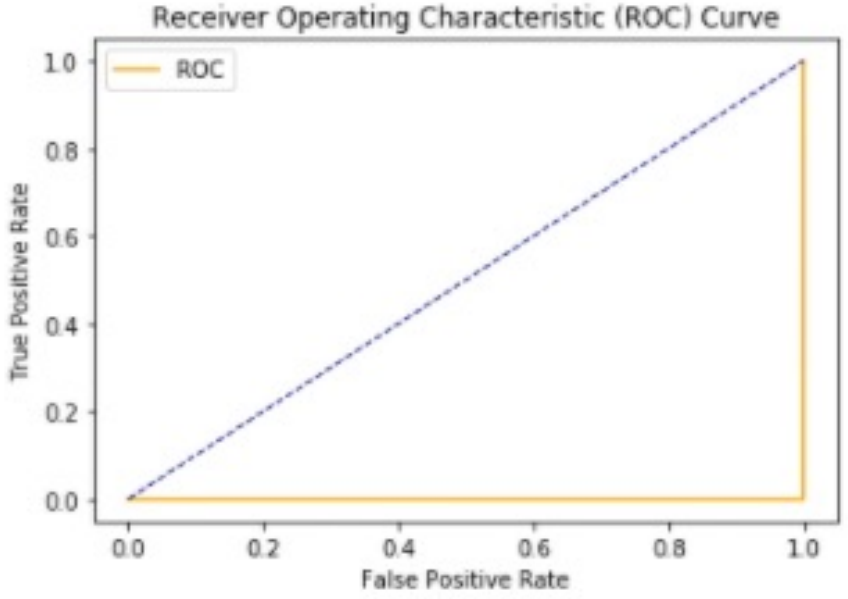


It is evident from the ROC AUC curve diagram, that the area between ROC and the axis is 0.5.

AUC - ROC Curve

- Scenario #3 (Worst Case Scenario)



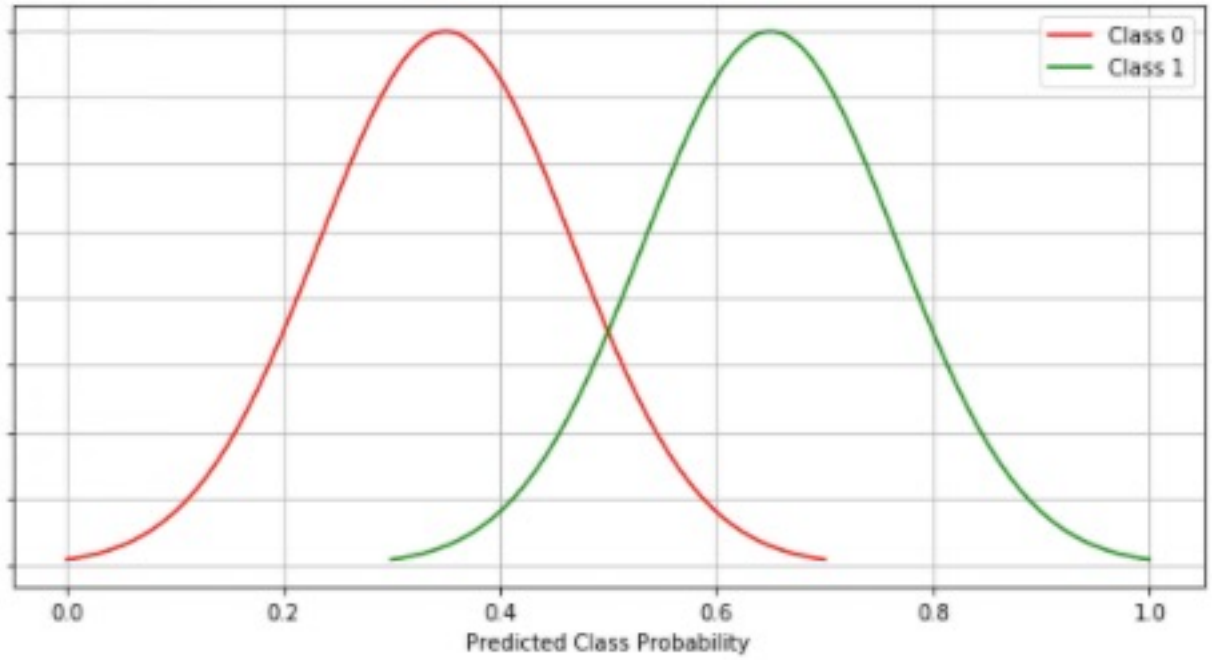


we get AUC to be 0, which the worst case scenario.

AUC - ROC Curve

- Scenario #4 (Industry / Norm Scenario)
  - best cases are never observed. We never get a clear distinction between the two classes.

we have some overlapping and that introduces Type 1 and Type 2 errors to the model prediction. In this case we get AUC to be somewhere between 0.5 and 1



Workshop Auto-mpg dataset

Auto-mpg dataset

Auto-mpg dataset

- Attribute Information:
  1. mpg: continuous (ไมล์ต่อแกลลอน)
  2. cylinders: multi-valued discrete (กระบอกสูบ)
  3. displacement: continuous (แรงดันกระบอกสูบ)
  4. horsepower: continuous (แรงม้า)
  5. weight: continuous (น้ำหนัก)
  6. acceleration: continuous (อัตราการเปลี่ยนแปลงของความเร็ว)
  7. model year: multi-valued discrete (รุ่นรถ)
  8. origin: multi-valued discrete (ชนิดเครื่องยนต์)
  9. car name: string (unique for each instance)

	A	B	C	D	E	F	G	H	I
1	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
2	18	8	307	130	3504	12	70	1	chevrolet chevelle malibu
3	15	8	350	165	3693	11.5	70	1	buick skylark 320
4	18	8	318	150	3436	11	70	1	plymouth satellite
5	16	8	304	150	3433	12	70	1	amc rebel sst
6	17	8	302	140	3449	10.5	70	1	ford torino
7	15	8	429	198	4341	10	70	1	ford galaxie 500
8	14	8	454	220	4354	9	70	1	chevrolet impala
9	14	8	440	215	4312	8.5	70	1	plymouth
10	14	8	455	225	4425	10	70	1	pontiac ca
11	15	8	390	190	3850	8.5	70	1	amc amba
12	15	8	383	170	3563	10	70	1	dodge cha
13	14	8	340	160	3609	8	70	1	plymouth
14	15	8	400	150	3761	9.5	70	1	chevrolet
15	14	8	455	225	3086	10	70	1	buick esta
16	24	4	113	95	2372	15	70	3	toyota cor
17	22	6	198	95	2833	15.5	70	1	plymouth
18	18	6	199	97	2774	15.5	70	1	amc horne
19	21	6	200	85	2587	16	70	1	ford mave
20	27	4	97	88	2130	14.5	70	3	datsun pl5
21	26	4	97	46	1835	20.5	70	2	volkswage
22	25	4	110	87	2672	17.5	70	2	peugeot 5

- Number of Instances: 398
- Number of Attributes: 9 including the class attribute

Data Dictionary

Column Position	Attribute Name	Description	Examples
#1	mpg	fuel efficiency measured in miles per gallon (mpg)	9.0, 13.0, 41.5
#2	cylinders	number of cylinders in the engine	3, 4, 8
#3	displacement	engine displacement (in cubic inches)	68.0, 112.0, 455.0
#4	horsepower	engine horsepower	46.0, 70.0, 230.0
#5	weight	vehicle weight (in pounds)	1613, 3615, 5140
#6	acceleration	time to accelerate from O to 60 mph (in seconds)	8.00, 15.50, 24.80
#7	model year	model year	73, 79, 82
#8	origin	origin of car (1: American, 2: European, 3: Japanese)	1, 2, 3
#9	car name	car name	audi fox, subaru

https://code.datasciencedojo.com/tshrivas/dojoHub/tree/classification/Auto%20MPG%20Data%20Set

Read Dataset and Cleaning Data

+ โค้ด + ข้อความ

```
[ ] data=pd.read_csv('auto-mpg.csv')
```

```
[ ] data.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    object 
 4   weight        398 non-null    int64  
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64  
 7   origin        398 non-null    int64  
 8   car name      398 non-null    object 
dtypes: float64(3), int64(4), object(2)
memory usage: 28.1+ KB
```

```
#Look at '?'
#Case 2 Replace data
data=pd.read_csv('auto-mpg.csv')
print(data.shape)
data[data['horsepower']=='?']
```

```
(398, 9)
```

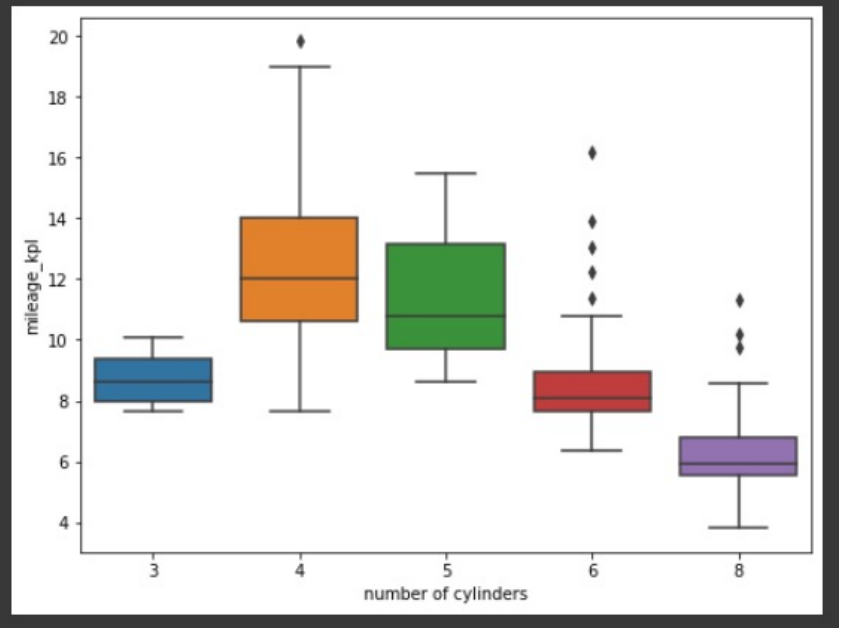
	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
32	25.0	4	98.0	?	2046	19.0	71	1	ford pinto
126	21.0	6	200.0	?	2875	17.0	74	1	ford maverick
330	40.9	4	85.0	?	1835	17.3	80	2	renault lecar deluxe
336	23.6	4	140.0	?	2905	14.3	80	1	ford mustang cobra
354	34.5	4	100.0	?	2320	15.8	81	2	renault 18i
374	23.0	4	151.0	?	3035	20.5	82	1	amc concord dl

Example dataset After Cleaning process

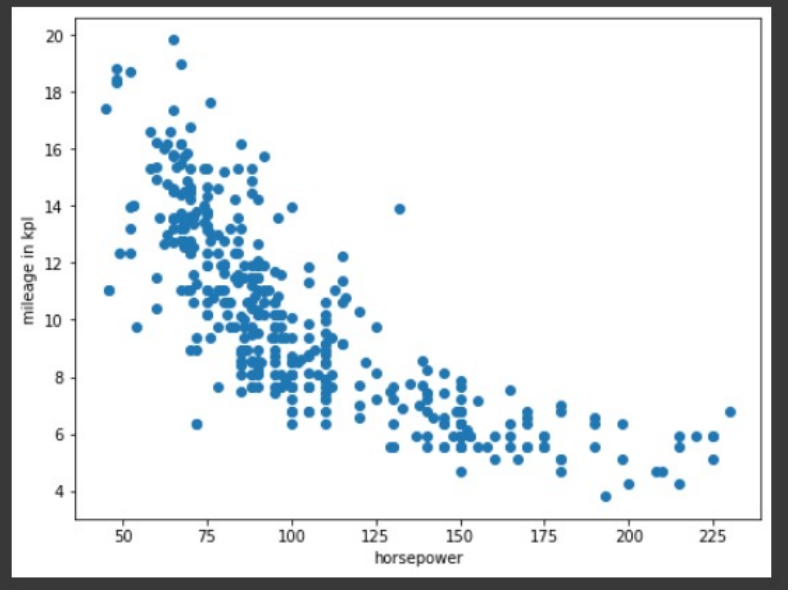
```
[258] data.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name	brand
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	chevrolet
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	buick
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	plymouth
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	amc
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	ford

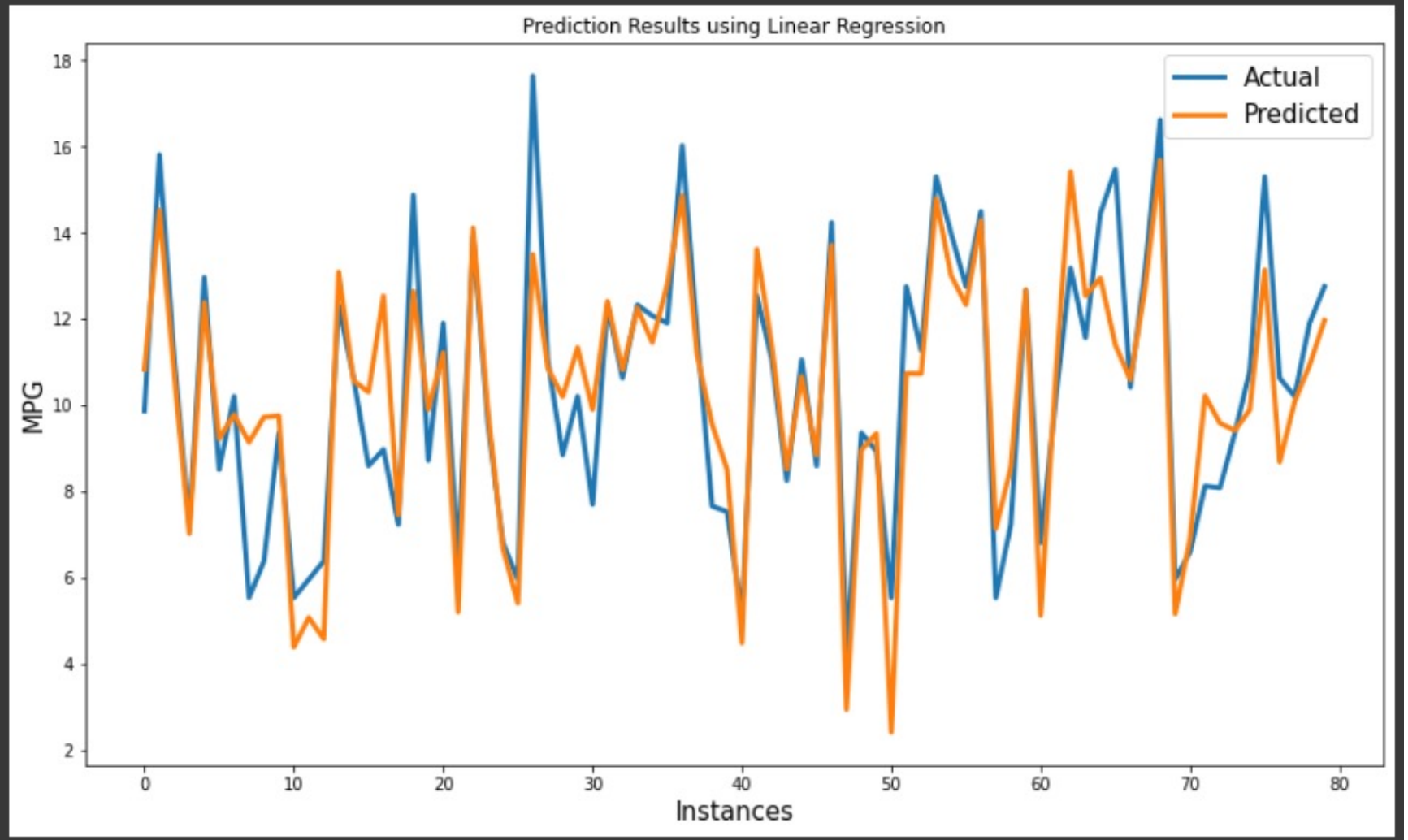
```
[267] plt.figure(figsize=(8,6))
      sns.boxplot(y=data['mileage_kpl'],x=data['cylinders'])
      plt.xlabel("number of cylinders");
```




```
[271] plt.figure(figsize=(8,6))
      plt.scatter(data.horsepower,data.mileage_kpl)
      plt.xlabel('horsepower')
      plt.ylabel('mileage in kpl');
```



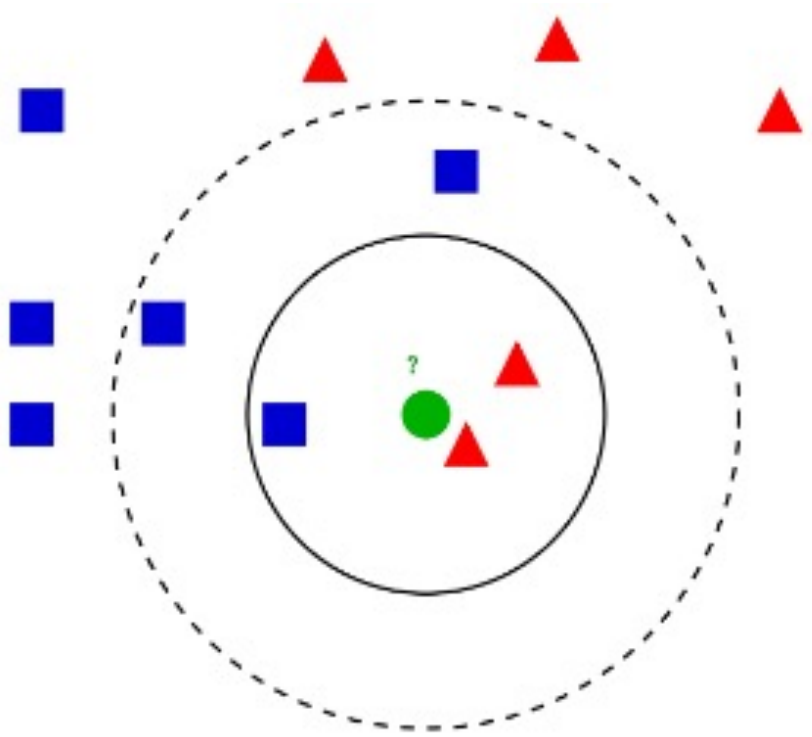
```
plt.figure(figsize=(14, 8))
plt.title("Prediction Results using Linear Regression")
result = [x for x in range(80)]
plt.plot(result, y_test[:80], linewidth=3, label="Actual")
plt.plot(result, y_pred[:80], linewidth=3, label="Predicted")
plt.ylabel('MPG', size=15)
plt.xlabel('Instances', size=15)
plt.legend(fontsize=15)
plt.show()
```



K-Nearest Neighbors

K-Nearest Neighbors

- **K-Nearest Neighbors** operates by checking the distance from some test example to the known values of some training example.
- The group of data points/class that would give the smallest distance between the training points and the testing point is the class that is selected.
- KNN is a supervised algorithm and non-parametric and lazy .



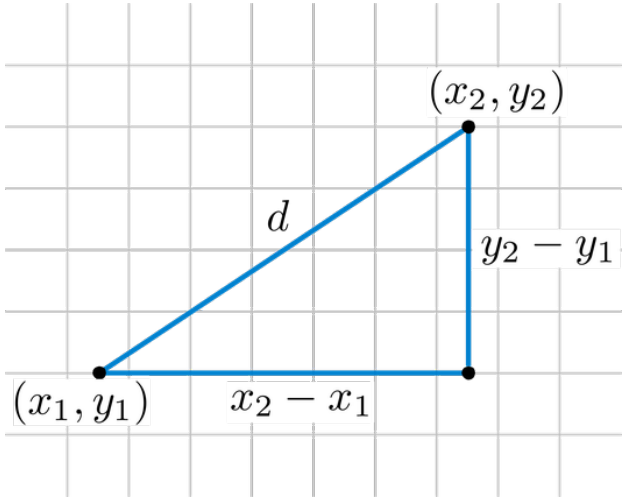
K-Nearest Neighbors

- Parameters:
  - n_neighborsint, default=5→ Number of neighbors to use
  - weights: {‘uniform’, ‘distance’}→ default = ‘uniform’, weight function used in prediction.
    - ‘uniform’ : uniform weights. All points in each neighborhood are weighted equally.
    - ‘distance’ : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
  - algorithm: {‘auto’, ‘ball_tree’, ‘kd_tree’, ‘brute’}, default = ‘auto’ → to compute the nearest neighbors:
  - leaf_size: int, default=30→ This can affect the speed of the construction and query
  - p: int, default=2→ when p=1, it means to use Manhattan_distance , p=2 is to use Euclidean_distance

K-Nearest Neighbors

- Parameters:
  - metric: str or callable, default='minkowski' → the distance metric to use for the tree
  - n_jobs: int, default=None
- Attributes:
  - classes_ : array of shape (n_classes,) → Class labels known to the classifier
  - effective_metric_ : str or callable → The distance metric used. It will be same as the metric parameter.
  - outputs_2d_ : bool

K-Nearest Neighbors



- The most used distance metrics are:
  - **Euclidean Distance**:
    - It's calculated as the square root of the sum of the squared differences between the two point of interes $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.
    - The formula is in 2D space:
  - **Manhattan Distance**:
    - Calculate the distance between real vectors using the sum of their absolute difference. Also called **City Block Distance**
    - The formula is in 2D space: $|(x_2 - x_1)| + |(y_2 - y_1)|$

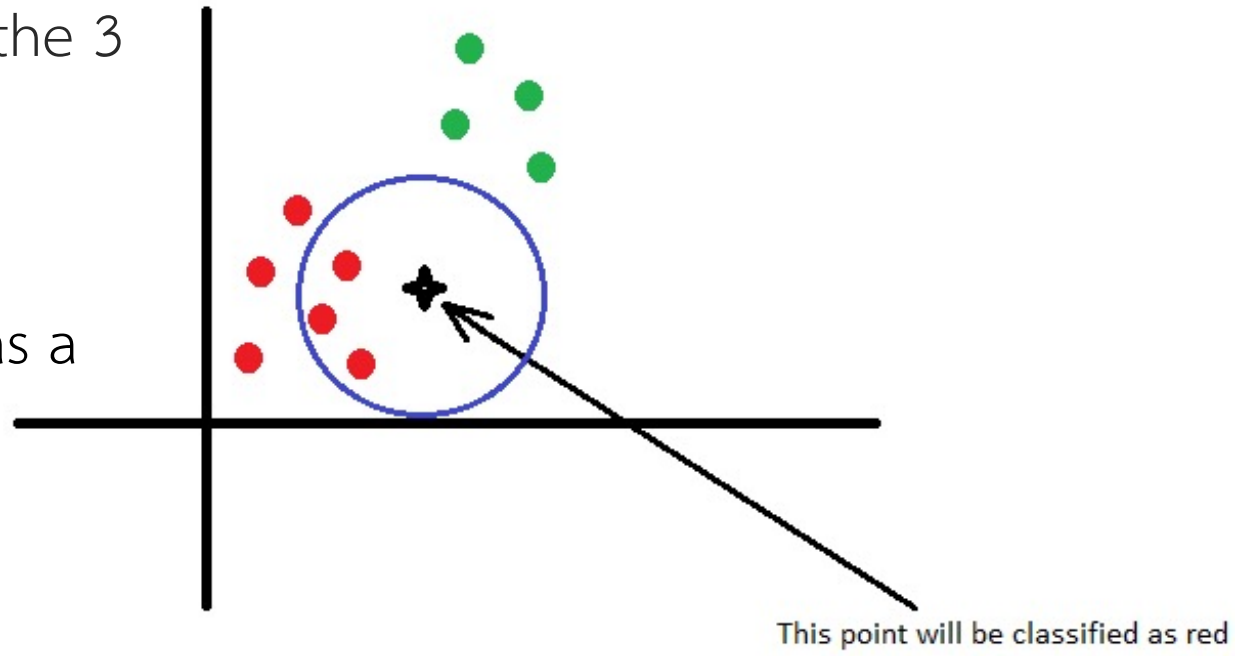
K-Nearest Neighbors

- The steps of the KNN algorithm are (**formal pseudocode**):
  - Initialize *selectedi = 0* for all ***i*** data points from the **training set**
  - Select a distance metric
  - For each training set data point *i* calculate the *distancei*
  - Choose the K parameter of the algorithm (*K = number of neighbors considered*), usually it's an odd number
  - For *j = 1 to K* loop through all the training set data points and in each step select the point with minimum distance
  - For each existing class count how many of the K selected data points are part of that class (voting)
  - Assign to the new observation the class with the maximum count

K-Nearest Neighbors

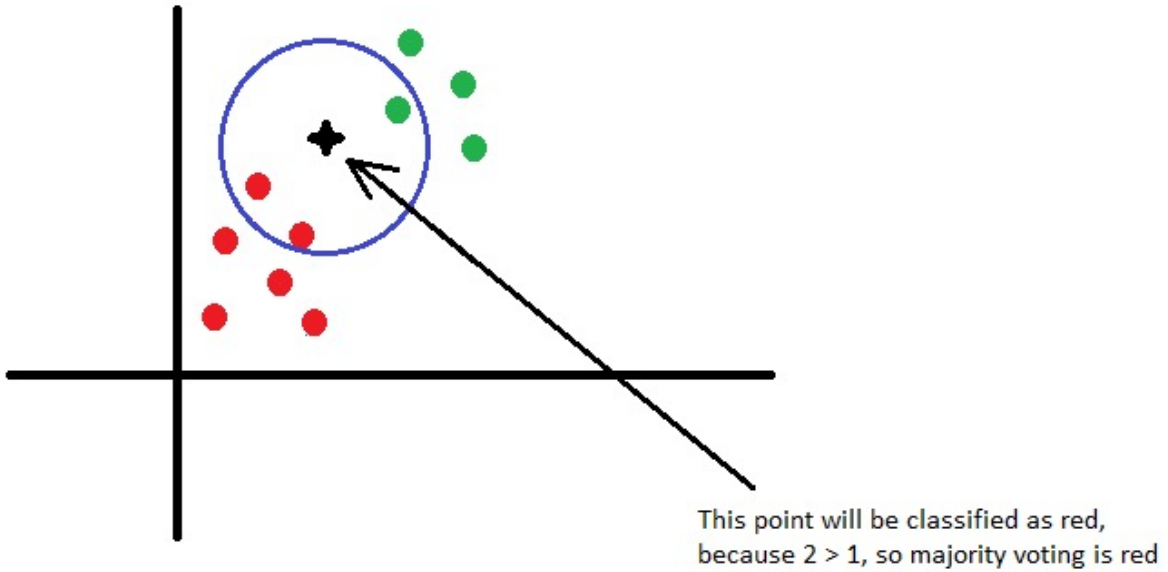
- For example, if we have two classes, red and green and after calculating the distances and getting the 3 nearest points

- We choose K = 3, so we will consider the 3 points with minimum distances
- The star is close to 3 red points
- So new observation will be classified as a red point.

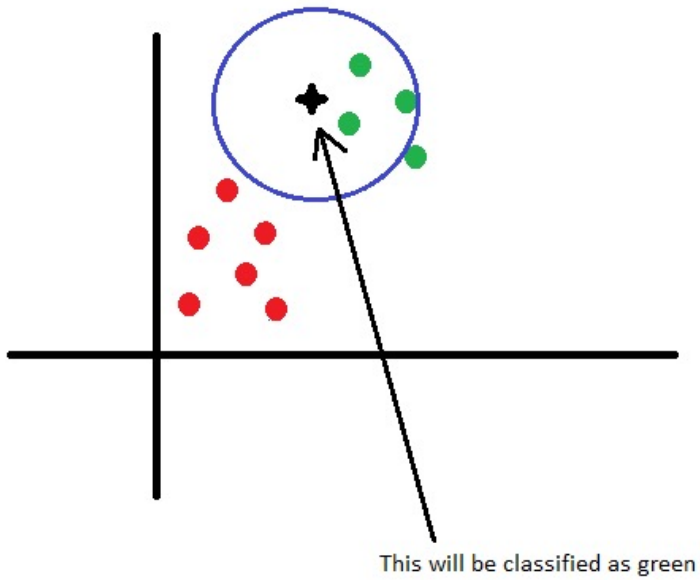


K-Nearest Neighbors

- moved the star a little bit closer the the green points.
- In this case we have 2 red and 1 green points selected. The star will be still classified as red, because 2 > 1.



- **Boundary** is used to separate class from each other.



Pros & Cons

- **Pros**
  - It is easy to implement
  - It is lazy learning algorithm and therefore requires no training prior to making real time predictions.
  - No training before making predictions, so new data can be added seamlessly.
  - There are only two parameters required to implement KNN (K and the distance function)

- **Cons**
  - The KNN algorithm doesn't work well with high dimensional data
  - The KNN algorithm has a high prediction cost for large datasets.
  - Finally, the KNN algorithm doesn't work well with categorical features since it is difficult to find the distance between dimensions with categorical features.

KNN example using Python

```
X = [[0], [1], [2], [3]]
y = [0, 0, 1, 1]

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X, y)
```

```
print(knn.predict([[1.1]]))
```

```
[0]
```

```
print(knn.predict_proba([[1.1]]))
```

```
[[0.66666667 0.33333333]]
```

```
print(knn.predict_proba([[3.9]]))
```

```
[[0.33333333 0.66666667]]
```

KNN example using Python

- https://www.kaggle.com/rakeshrau/social-network-ads#Social_Network_Ads.csv

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1

```
# Importing the dataset
dataset = pd.read_csv('data/Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
```

```
print(X.shape)
```

```
(400, 2)
```

KNN example using Python

- Split data for training and testing

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

- Feature Scaling
  - It is a method used to normalize the range of independent variables or features of data (as data normalization)
  - Rescaling (min-max normalization)

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

  - Mean normalization

$$x' = \frac{x - \text{average}(x)}{\max(x) - \min(x)}$$

  - Scaling to unit length

$$x' = \frac{x}{\|x\|}$$

KNN example using Python

- Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

- Training and Predictions

```
classifier = KNeighborsClassifier(n_neighbors = 2)
classifier.fit(X_train, y_train)
```

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(y_pred)
```

KNN example using Python

- Evaluating the Algorithm

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[66  2]
 [ 8 24]]
```