DATAHACKS 2020 WORKSHOP

ML Models

Presenter: Katyaini

Agenda

- Steps to Deal With a ML Problem
- Machine Learning Models
  - Linear Regression
  - Logistic Regression
  - SVM
  - Perceptron
  - Clustering
- Model Interpretation

Jupyter Notebooks

- Jupyter notebooks allow for live visualization of your datasets, making it a lot easier to work with, as opposed to constantly printing information out.

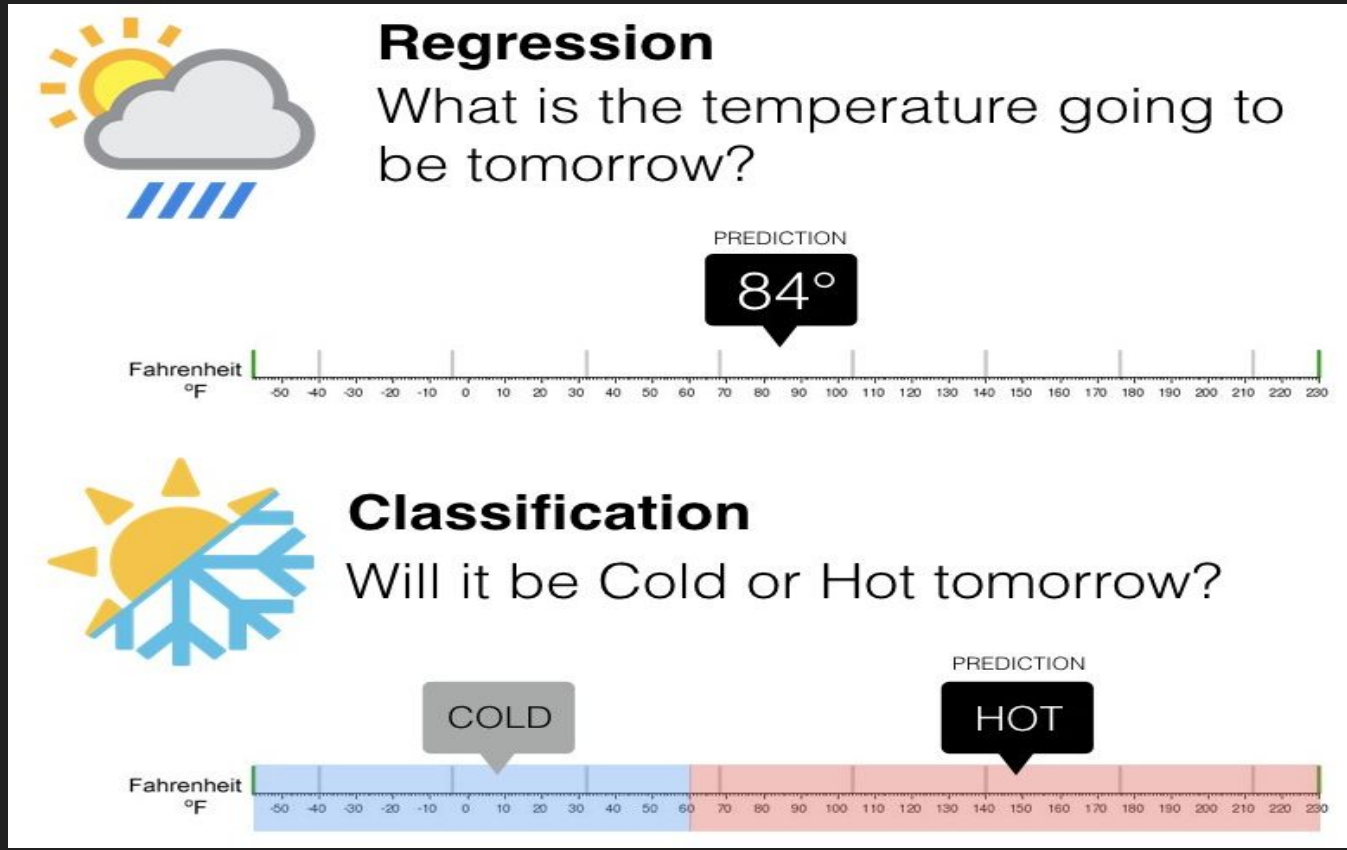
Regression vs. Classification

- Regression: predicting continuous values for Y
  - Ex. house prices in San Diego
  - Ex. temperature for tomorrow
- Classification: predicting discrete values for Y
  - Ex. email: spam vs. not spam
  - Ex. animal: cat or dog or rabbit

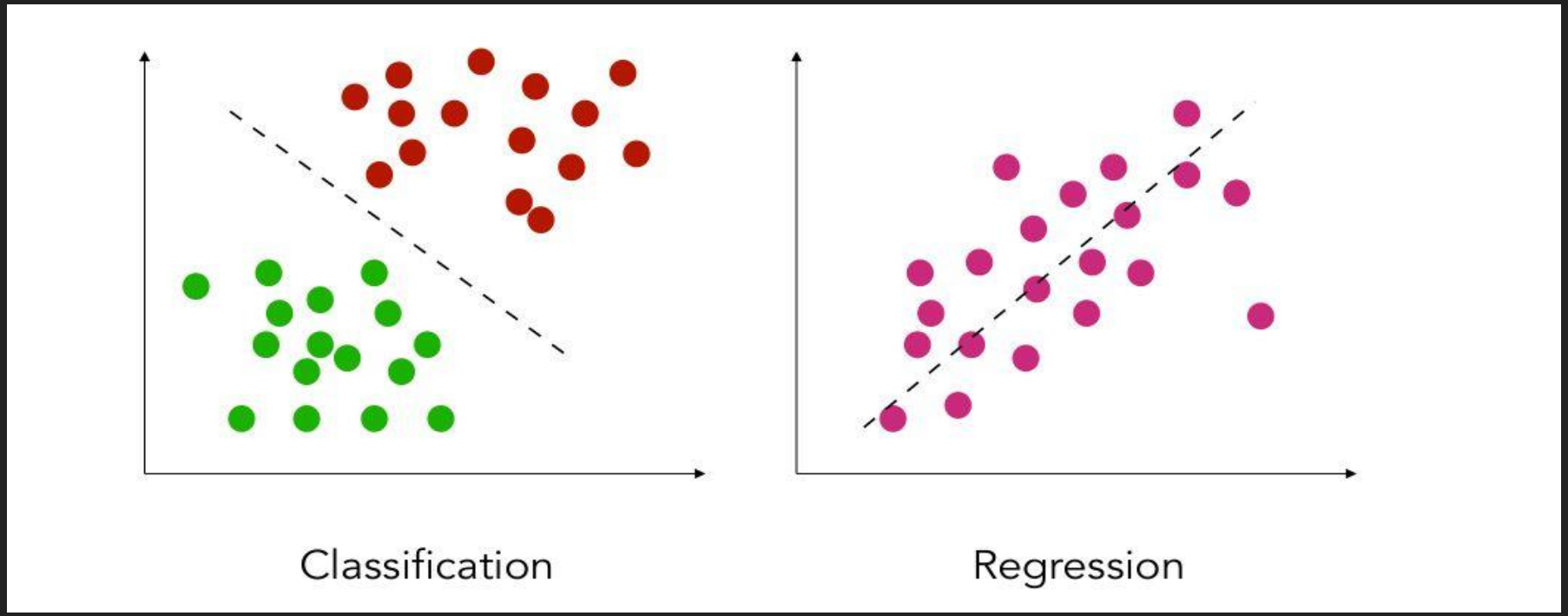
Example

Continuous

Discrete



Data Representation

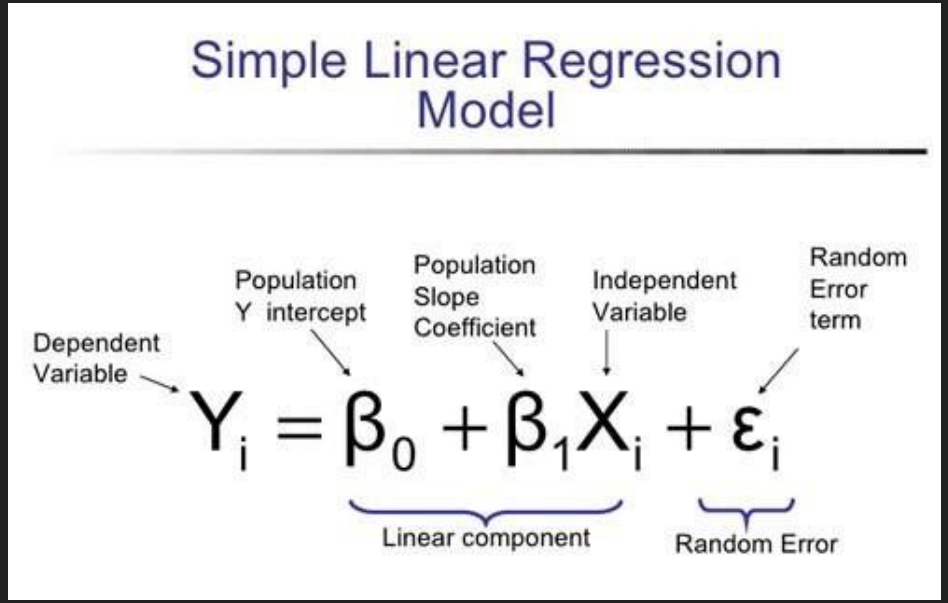


Linear Regression

- Supervised Learning - basically means you have Input as well as output in your dataset!
- The core idea is to try to find the line that best fits the dataset.
- Finding relationships between continuous variables
- The line is chosen in order to reduce the total prediction error.
- Super basic, use when you want to model how different kinds of input parameters impact the output!

Math Behind Linear Regression

Modeling your dependent variable(the Y values) based on your independent variables (the X values)



$$MSE = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}$$

DEMO - House Price Prediction

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib as plt
from math import sqrt
```

```
df_train = pd.read_csv('house_train.csv')
df_train.head()
```

```
features = ['sqft_living','bedrooms', 'yr_built']
x = df_train.loc[:, features]
y = df_train.price
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 0)
```

```
# instantiate, fit
linreg = LinearRegression()
linreg.fit(x_train, y_train)
print linreg.intercept_
print linreg.coef_
```

```
y_pred = linreg.predict(x_test)
lr_r2 =  r2_score(y_test, y_pred)
print "R squared: ", (lr_r2)
print "Root Mean Squared Error: ", sqrt(mean_squared_error(y_test, y_pred))
```

Logistic Regression

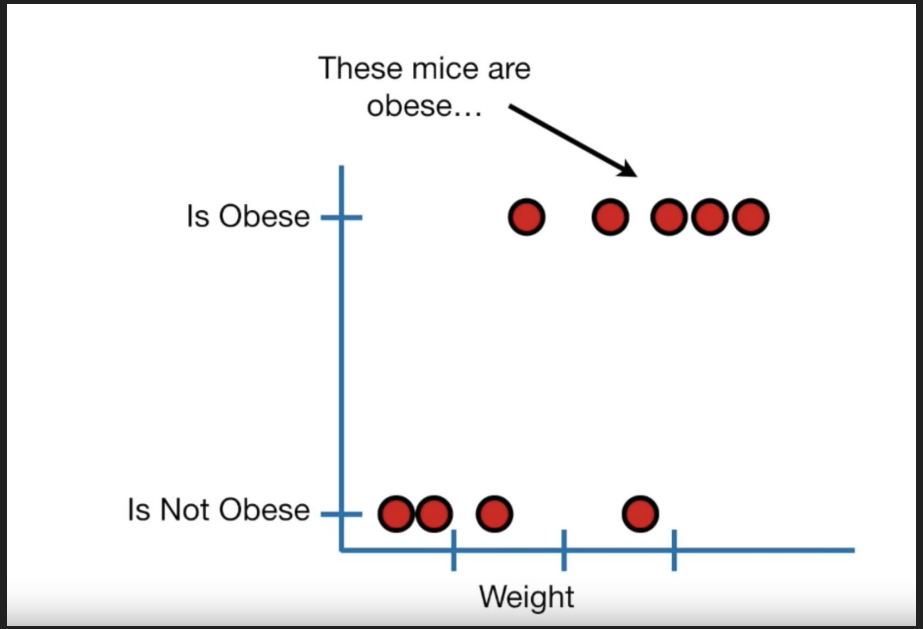
- Classification algorithm - works best for binary classification!

- Used to model the probability of a certain data point existing in either group-A or group-B using something called a sigmoid function.

- Trying to find a decision boundary that separates the data to minimize the no of incorrectly classified points.

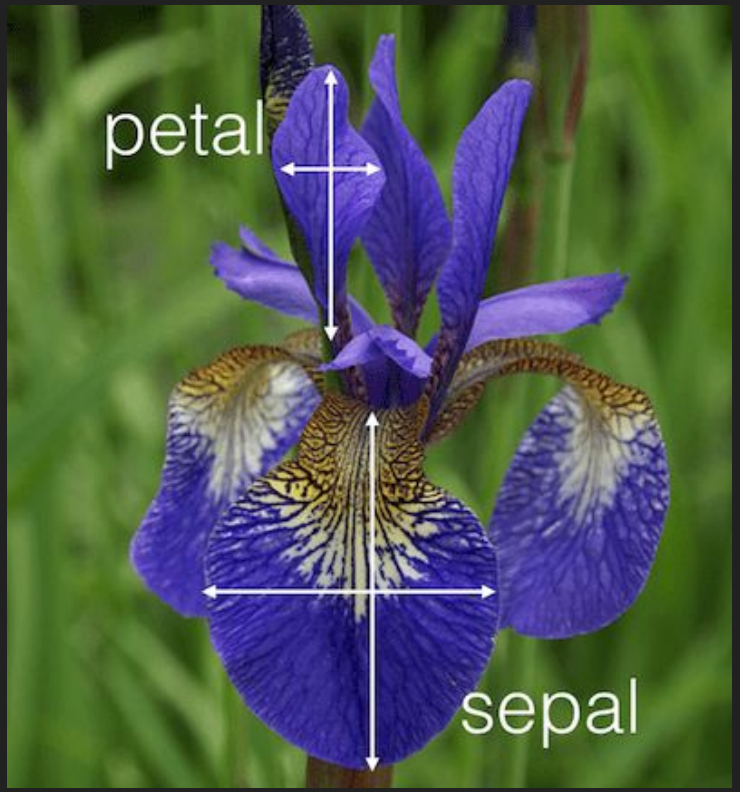
- Most common example - classify email as spam or not

Predicting whether mice are obese based on variable weight



DEMO

- 4 Features:
  - Sepal Length
  - Sepal Width
  - Petal Length
  - Petal Width
- 3 Classes:
  - Setosa
  - Versicolour
  - Virginica



Loading the dataset + Splitting

In [ ]:

In [1]:
```
# Import the dependencies
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

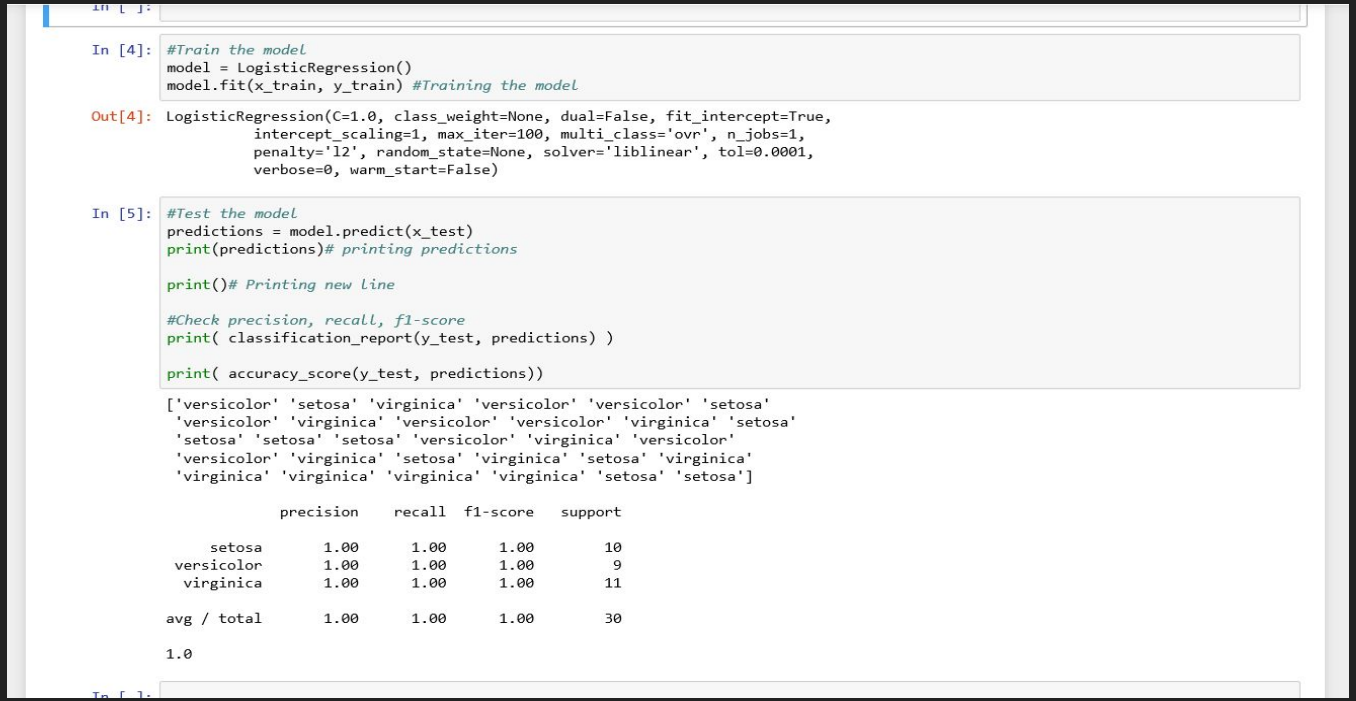
In [2]:
```
#Load the data set
data = sns.load_dataset("iris")
data.head()
```

Out[2]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [3]:
```
#Prepare the training set

# X = feature values, all the columns except the last column
X = data.iloc[:, :-1]

# y = target values, last column of the data frame
y = data.iloc[:, -1]

#Split the data into 80% training and 20% testing
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Running Logistic Regression



In [ ]:

In [4]:
```
#Train the model
model = LogisticRegression()
model.fit(x_train, y_train) #Training the model
```

Out[4]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

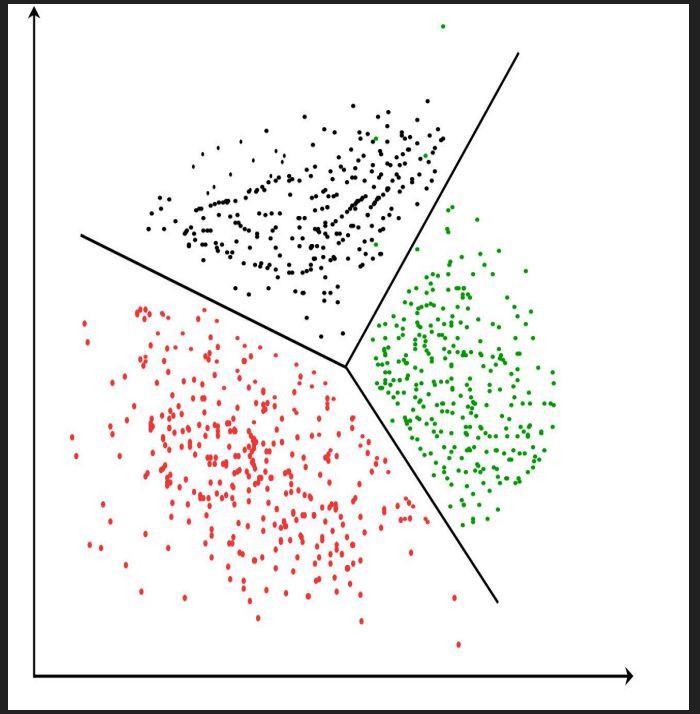
In [5]:
```
#Test the model
predictions = model.predict(x_test)
print(predictions)# printing predictions

print()# Printing new line

#Check precision, recall, f1-score
print( classification_report(y_test, predictions) )

print( accuracy_score(y_test, predictions))
```

```
['versicolor' 'setosa' 'virginica' 'versicolor' 'versicolor' 'setosa'
 'versicolor' 'virginica' 'versicolor' 'versicolor' 'virginica' 'setosa'
 'setosa' 'setosa' 'setosa' 'versicolor' 'virginica' 'versicolor'
 'versicolor' 'virginica' 'setosa' 'virginica' 'setosa' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'setosa' 'setosa']

             precision    recall  f1-score   support

     setosa       1.00      1.00      1.00        10
 versicolor       1.00      1.00      1.00         9
  virginica       1.00      1.00      1.00        11

avg / total       1.00      1.00      1.00        30

1.0
```

In [ ]:

Clustering - KMeans

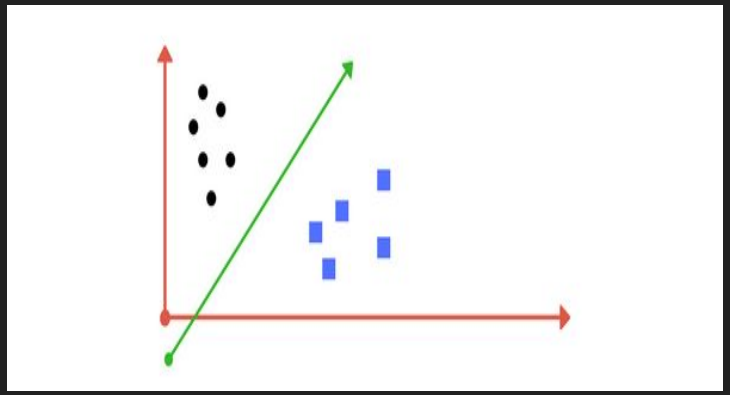
- Unsupervised learning
- Dividing the data-points into groups based on similar traits.
- K means is an iterative clustering algorithm that aims to find local maxima in each iteration.
  1. Randomly assign each point to a cluster
  2. Compute cluster centroid: Use mean(each point in cluster)
  3. Re-assign each point to closest centroid
  4. Re-compute centroid
- Other kinds like hierarchical, choose which one to use based on what dataset looks like



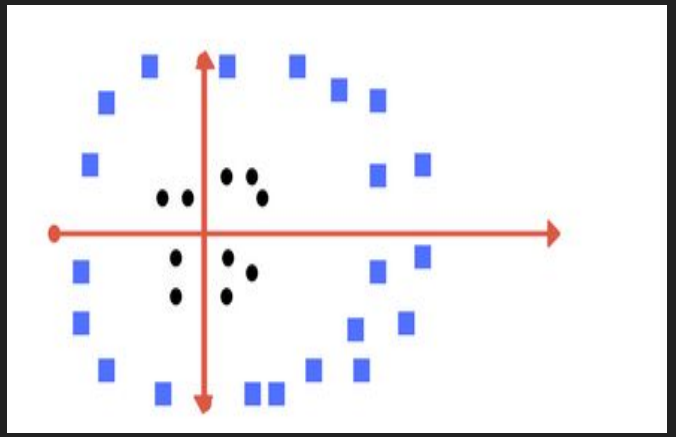
SVM

It finds out a line/ hyper-plane in multidimensional space that separates out classes!

Simple-line separation



Hyperplane separation

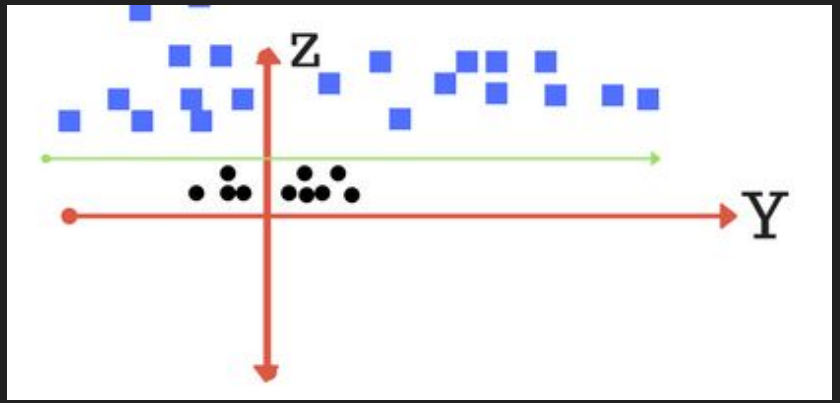


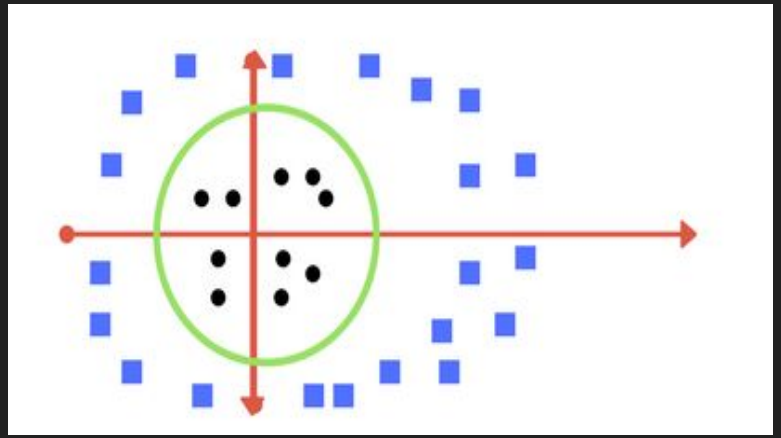
What is a hyperplane?

Adds one more dimension as we call it z-axis. Let us assume value of points on z plane, $w = x^2 + y^2$

When we transform back this line to original plane, it maps to circular boundary. These transformations are called kernels.

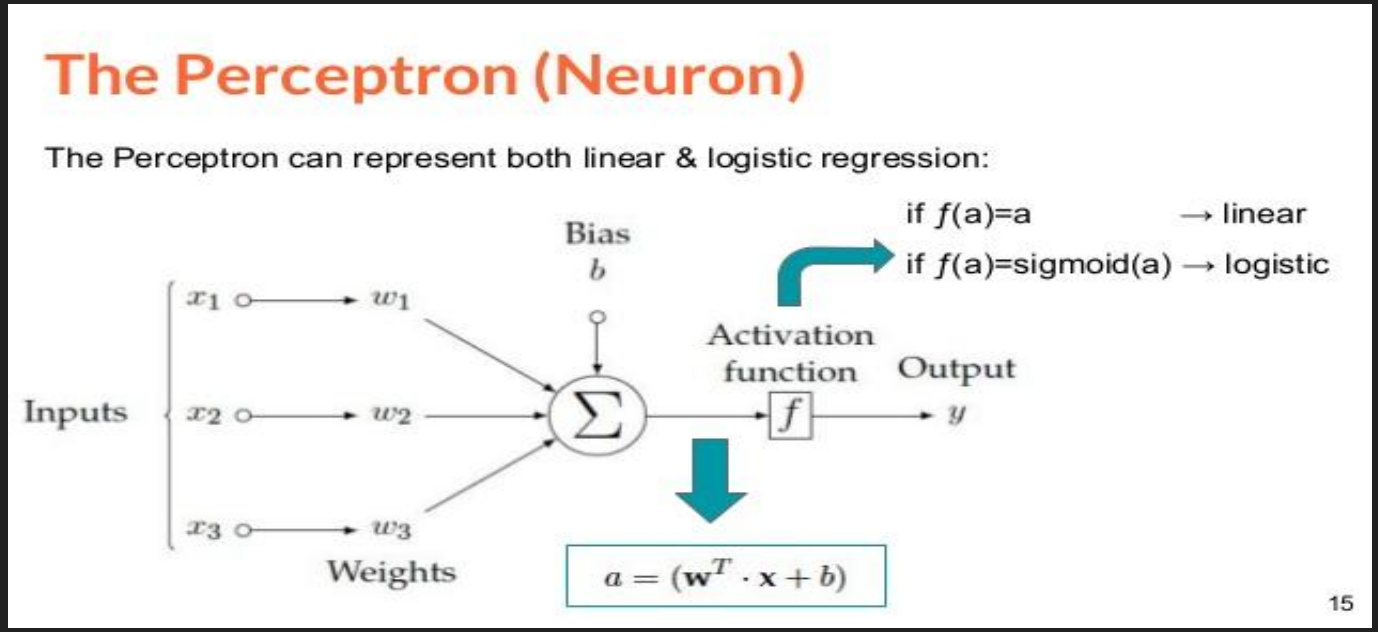
The sklearn library's SVM implementation provides the transformations inbuilt





Perceptrons

A perceptron is a single layer neural network



Fun Fact: A single perceptron with a sigmoid activation is the same as a logistic model

Data Splitting - Why?

- Training Data: The data used to fit the model. Model sees and learns from the data

- Validation Set: Used to tune the model parameters. Sees the data but never learns from it

- Test Set: Test out the accuracy of the model on a dataset it has never seen before.

K-fold Cross Validation: Resampling the test data multiple times to avoid the problem of getting different accuracies on different test sets.