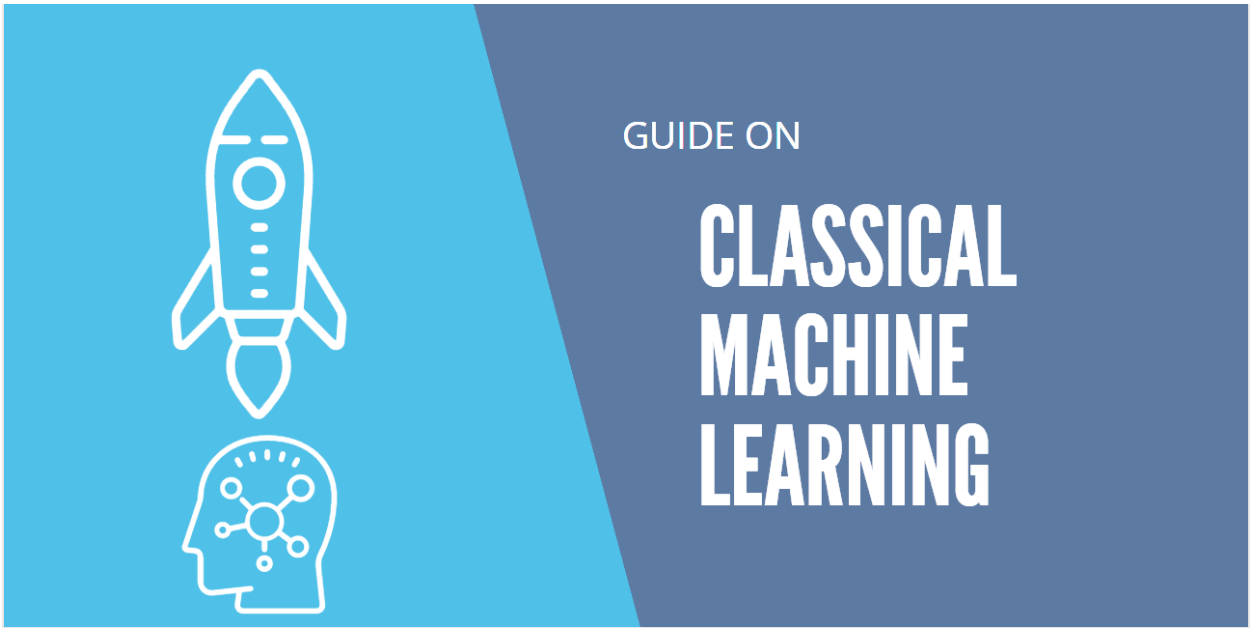
beingdatum
the data society



GUIDE ON CLASSICAL MACHINE LEARNING

About the Tutorial : Today's Artificial Intelligence (AI) has far surpassed the hype of blockchain and quantum computing. The developers now take advantage of this in creating new Machine Learning models and to re-train the existing models for better performance and results. This tutorial will give an introduction to machine learning and its implementation in Artificial Intelligence.

Audience : This tutorial has been prepared for professionals aspiring to learn the complete picture of Machine Learning and AI. This tutorial caters to the learning needs of both the novice learners and experts, to help them understand the concepts and implementation of AI

Prerequisites : The learners of this tutorial are expected to know the basics of Python programming. Besides, they need to have a solid understanding of computer programing and fundamentals. If you are new to this arena, we suggest you pick up tutorials based on these concepts first, before you embark on with Machine Learning.

Course Curriculum

Let's have a glance at some of the chapters of this course below:

Multiple Linear Regression

The difference between simple linear regression and multiple linear regression, multiple linear regression has (>1) independent variables, whereas simple linear regression has only 1 independent variable.

Simple Linear Regression → y = bo + b1*x1

Multiple Linear Regression → y = bo + b1*x1 + b2*x2 + ... + bn*xn

Where, y → Dependent variable

x1, x2, ...xn → Independent variables

Assumptions of Linear Regression

1. Linearity
2. Homoscedasticity
3. Multivariate normality
4. Independence of errors
5. Lack of multicollinearity

Data:

In case we have data as below:

dataset - DataFrame

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

Goal: If we have a correlation of Profit with other variables or not.
Profit → Dependent variables
Others → Independent variables

Y = bo + b1*x1 + b2*x2 + b3*x3 + b4*x4

We can ignore State, as State is a categorical variable.
Hence, we need to create dummy variables i.e get_dummies or one hot encoding.

Dummy Variables

dataset - DataFrame

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

New York	California	Florida
1	0	0
0	1	0
0	0	1
1	0	0
0	0	1
1	0	0
0	1	0

Y = bo + b1*x1 + b2*x2 + b3*x3 + b4*D1 + b5*D2 + b6*D3

Always omit one dummy variable.

Model Building

In the case of Simple Linear Regression, it's quite easy to build a model, but here we need to decide which variables are important and which are not.

Not necessarily, more variables are better for a good model. We need to construct a model wisely by selecting the perfect set of variables.

5 methods of building models:

1. All-in
2. Backward Elimination → Stepwise
3. Forward Selection → Stepwise
4. Bidirectional Elimination → Stepwise
5. Score Comparison

All-in cases → Domain Knowledge is quite important for these cases, where you already know which variables are going to be useful for your model.

Backward Elimination →

Step 1 : Select a significance level to stay in the model (SL = 0.05)
Step 2: Fit the full model with all possible predictors
Step 3: Consider the predictor with the highest p-value. If P > SL go to Step 4, otherwise go to END.
Step 4: Remove the predictor
Step 5: Fit the model without this variable → Rebuilding → Coefficients, constant are going to be different
Go to Step 3, keep doing for all variables
Step 6: END → Your model is READY

Forward Selection →

Step 1 : Select a significance level to stay in the model (SL = 0.05)
Step 2: Fit all simple regression models y~xn. Select the one with lowest p-value
Step 3: Keep this variable and fit all possible models with one extra predictor added to the one(s) you already have.
Step 4: Consider the predictor with the lowest p-value. If P < SL, go to Step 3, otherwise go to END.
Go to Step 3, keep doing till all the variables are used
Step 5: END → Keep the previous MODEL

Bidirectional Elimination →

Step 1: Select a significance level to enter and to stay in the model
Eg: SLENTER = 0.05, SLSTAY = 0.05
Step 2: Perform the next step of forward selection (new variable must have P < SLENTER to enter)
Step 3: Perform all steps of backward elimination (old variables must have P < SLSTAY to stay)
Go to Step 2
Step 4: No new variables can enter no old variables can exit.
Step 5: Model is READY

All Possible Models

Step 1: Select a criterion of goodness of fit
Step 2: Construct all possible regression models (2^n -1) total combinations
Step 3: Select the one with the best criterion
Step 4: Model is READY

Example: 10 columns means, 2^10-1 = 1024-1 = **1023 models**

Python Implementation

The data looks like below:

dataset - DataFrame

Index	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	carwidth	carheight	curbweight
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5	52.4	2823
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	176.6	66.2	54.3	2337
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	176.6	66.4	54.3	2824
5	6	2	audi fox	gas	std	two	sedan	fwd	front	99.8	177.3	66.3	53.1	2507
6	7	1	audi 100ls	gas	std	four	sedan	fwd	front	105.8	192.7	71.4	55.7	2844
7	8	1	audi 5000	gas	std	four	wagon	fwd	front	105.8	192.7	71.4	55.7	2954
8	9	1	audi 4000	gas	turbo	four	sedan	fwd	front	105.8	192.7	71.4	55.9	3086

In the data, we have multiple variables, to make it look easier, let's remove all the categorical variables and just deal with the numerical and proceed with the model building part.

Student's task: Build a multiple linear regression model considering the entire data.

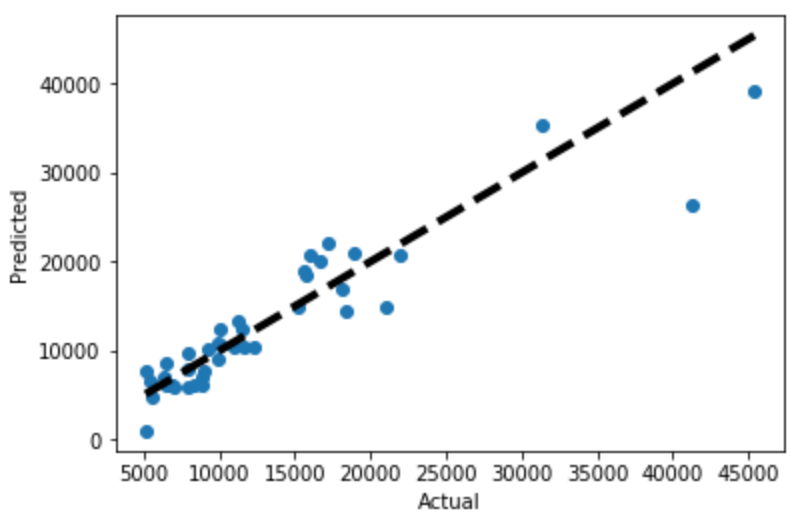
```
# Multiple Linear Regression

# Importing the libraries
import numpy as np
```

```
import matplotlib.pyplot as plt
import pandas as pd


# Importing the dataset
dataset = pd.read_csv('carprice.csv')

#Removing few features to make the solution less complicated
#The below features can be converted into OneHotEncoder and the results can be
enhanced
dataset.pop('fueltype')
dataset.pop('aspiration')
dataset.pop('drivewheel')
dataset.pop('enginelocation')
dataset.pop('enginetype')
dataset.pop('fuelsystem')
dataset.pop('CarName')

#Let's simply convert these categorical to numerical's using LabelEncoder/replace
functionality
dataset['doornumber'].replace(['four','two'],[4,2],inplace=True)
dataset['carbody'].replace(['sedan','hatchback', 'wagon', 'hardtop',
'convertible'],[1,2,3,4,5],inplace=True)
dataset['cylindernumber'].replace(['four','six','five', 'eight', 'two', 'three',
'twelve'],[4,6,5,8,2,3,12],inplace=True)

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 18].values

#No use of OneHotEncoder here, as we have removed the categorical fields.
'''
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder = LabelEncoder()
X[:, 2] = labelencoder.fit_transform(X[:, 2])
onehotencoder = OneHotEncoder(categorical_features = [2])
X = onehotencoder.fit_transform(X).toarray()

# Avoiding the Dummy Variable Trap
X = X[:, 1:]
'''


# Splitting the dataset into the Training set and Test set
```

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)


# Fitting Multiple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

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

Let's plot Actual vs Predicted results:

```
#Let's plot Actual vs Predicted
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax.scatter(y_test, y_pred)
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
ax.set_xlabel('Actual')
ax.set_ylabel('Predicted')
plt.show()
```



Do you think this model is the optimum model with the dataset we have?

When we built the model, we used all the independent variables that were available, but what if among these independent variables, there are some that are highly statistically significant, which has a great impact on the dependent variable (Profit), and some that are not statistically significant variables, which means, if we remove the non statistically significant variables, there won't be much impact on the output of the model.

Let's try : Backward Elimination

Remember in the equation of Multiple Linear Regression i.e.

*y = bo + b1*x1 + b2*x2 + b3*x3 + b4*x4*

y → Dependent variable
bo → Constant
x1, x2, ... xn → Independent variables

bo is basically bo*xo, where xo = 1.

Statsmodels library doesn't take into account bo as constant, so we need to add it to the matrix of the dependent variable, we will add xo = 1.

```
#Building optimal model using Backward Elimination
import statsmodels.api as sm

X = np.append(arr = np.ones((205, 1)).astype(int), values = X, axis = 1)
#Step 2 of backward elimination
X_opt = X[:, [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17, 18]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
```

Output:

```
Out[94]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
=====
Dep. Variable:                      y   R-squared:                       0.872
Model:                            OLS   Adj. R-squared:                  0.860
Method:                 Least Squares   F-statistic:                     70.41
```

```
Date:                Mon, 06 Jan 2020   Prob (F-statistic):          6.28e-73
Time:                        15:14:34   Log-Likelihood:                -1921.7
No. Observations:                 205   AIC:                             3881.
Df Residuals:                     186   BIC:                             3945.
Df Model:                          18
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -4.502e+04   1.55e+04     -2.904      0.004   -7.56e+04   -1.44e+04
x1          -17.8152      4.074     -4.373      0.000     -25.852      -9.778
x2           65.2123    279.806      0.233      0.816    -486.788     617.213
x3          224.4821    329.699      0.681      0.497    -425.948     874.912
x4          321.2178    258.763      1.241      0.216    -189.269     831.705
x5          115.0941    107.157      1.074      0.284     -96.306     326.494
x6          -63.1336     55.123     -1.145      0.254    -171.881      45.613
x7          588.8392    241.943      2.434      0.016     111.535    1066.143
x8          248.6145    135.514      1.835      0.068     -18.727     515.956
x9            0.1585      1.715      0.092      0.926      -3.224       3.541
x10       -1776.7108    669.892     -2.652      0.009   -3098.275    -455.147
x11         159.8603     24.628      6.491      0.000     111.274     208.446
x12       -3071.9252   1643.611     -1.869      0.063   -6314.441     170.591
x13       -4725.0468    910.237     -5.191      0.000   -6520.762   -2929.331
x14         331.8705     79.060      4.198      0.000     175.900     487.841
x15          57.1478     16.261      3.514      0.001      25.068      89.228
x16           1.9091      0.636      3.003      0.003       0.655       3.163
x17        -225.4306    171.695     -1.313      0.191    -564.151     113.289
x18         179.8634    152.851      1.177      0.241    -121.682     481.409
==============================================================================
Omnibus:                       19.492   Durbin-Watson:                   0.974
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               52.169
Skew:                           0.336   Prob(JB):                     4.69e-12
Kurtosis:                       5.378   Cond. No.                     4.27e+05
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.27e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

"""

x9 has the highest p value, hence we remove it in the next run → And the process continues till we have the variables with a p value less than 0.05

```
X_opt = X[:, [0,1,2,3,4,5,6,7,8,10,11,12,13,14,15,16,17,18]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_opt = X[:, [0,1,3,4,5,6,7,8,10,11,12,13,14,15,16,17,18]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_opt = X[:, [0,1,4,5,6,7,8,10,11,12,13,14,15,16,17,18]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_opt = X[:, [0,1,4,5,7,8,10,11,12,13,14,15,16,17,18]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_opt = X[:, [0,1,4,7,8,10,11,12,13,14,15,16,17,18]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_opt = X[:, [0,1,4,7,8,10,11,12,13,14,15,16,17]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_opt = X[:, [0,1,4,7,8,10,11,12,13,14,15,16]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_opt = X[:, [0,1,7,8,10,11,12,13,14,15,16]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_opt = X[:, [0,1,7,8,10,11,13,14,15,16]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_opt = X[:, [0,1,7,10,11,13,14,15,16]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
```

```
regressor_OLS.summary()
```

Output

```
Out[104]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.861
Model:                            OLS   Adj. R-squared:                  0.856
Method:                 Least Squares   F-statistic:                     152.3
Date:                Mon, 06 Jan 2020   Prob (F-statistic):           8.15e-80
Time:                        15:18:43   Log-Likelihood:                -1929.9
No. Observations:                 205   AIC:                             3878.
Df Residuals:                     196   BIC:                             3908.
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -4.985e+04   1.06e+04    -4.697      0.000   -7.08e+04   -2.89e+04
x1          -17.7694      3.842     -4.625      0.000     -25.346     -10.193
x2          724.7305    158.059      4.585      0.000     413.016    1036.445
x3        -1211.8347    409.088     -2.962      0.003   -2018.614    -405.055
x4          142.7100     16.585      8.605      0.000     110.001     175.419
x5        -4230.4288    770.725     -5.489      0.000   -5750.407   -2710.451
x6          314.4289     64.536      4.872      0.000     187.154     441.704
x7           44.8901     12.055      3.724      0.000      21.116      68.664
x8            2.0100      0.620      3.241      0.001       0.787       3.233
==============================================================================
Omnibus:                       15.874   Durbin-Watson:                   0.943
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               33.383
Skew:                           0.335   Prob(JB):                     5.64e-08
Kurtosis:                       4.860   Cond. No.                     2.58e+05
==============================================================================

Warnings:
```

```
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.58e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

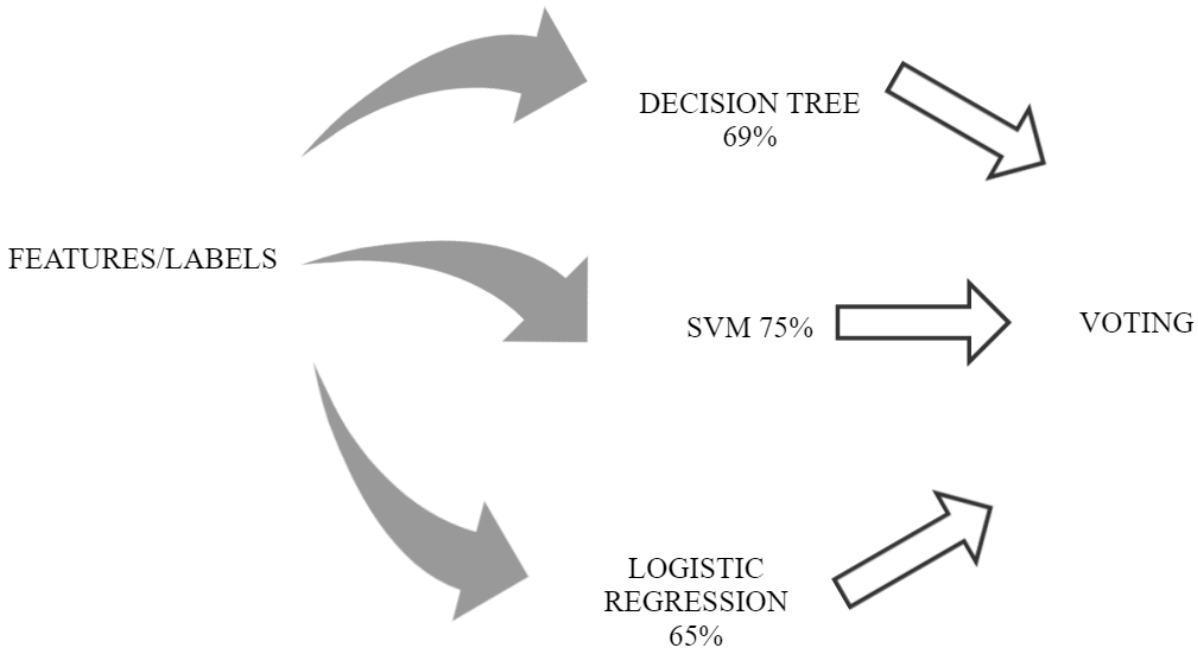
Ensemble Techniques

Ensemble Learning

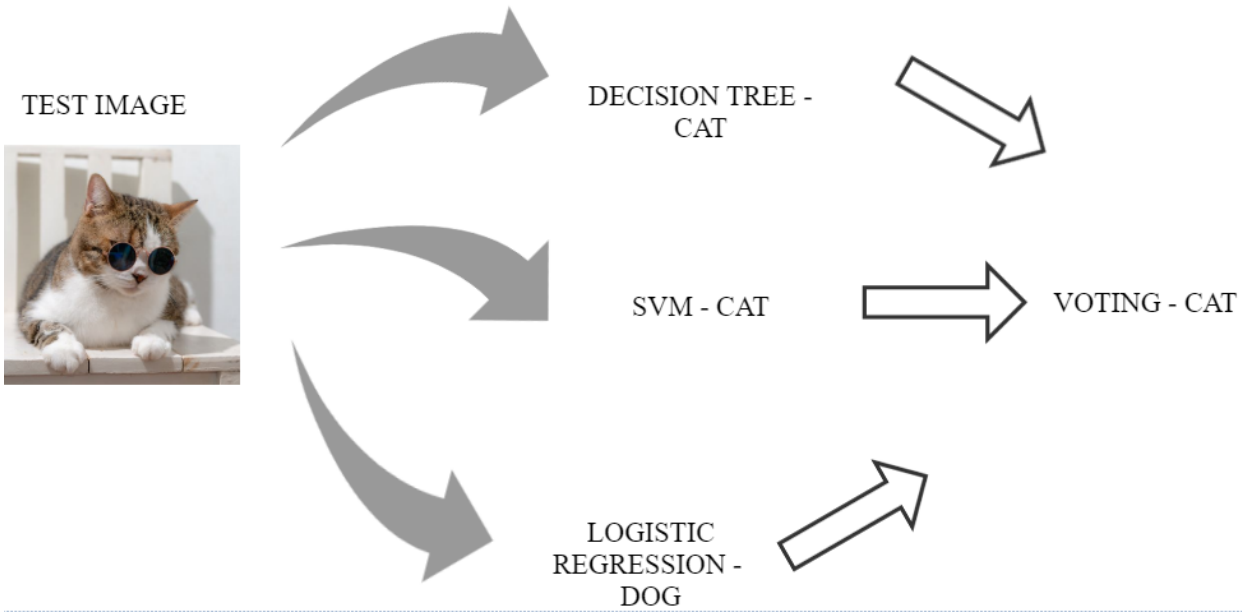
Ensemble Learning is the mechanism to use multiple algorithms together to have a better prediction than the individual models.

Let say, we extract the features from a use case, and try to check the model's accuracy or behaviour individually.

Features → Decision Trees → 69%
Features → SVM → 75%
Features → Logistic Regression → 65%



In the above picture, we tried different algorithms and got a maximum accuracy of 75%, so what if we combine 3 models, and take a vote on each output, will it increase the accuracy of the entire model?



Let's say, we have a test image of a cat, and we train different classifiers, where Decision Tree & SVN identified it as CAT, but Logistic Regression identified it as a DOG, and considering it as a whole model, we take the voting and predict the image as a CAT. Obviously, the efficiency of ensemble models will be better than individual models, but the computational power will increase, and training a model will take more time as compared to the individual models.

So why use Ensemble Learning?

1. Better Accuracy (Low Error)
2. Higher Consistency (Avoids Overfitting)
3. Reduces Bias & Variance Errors

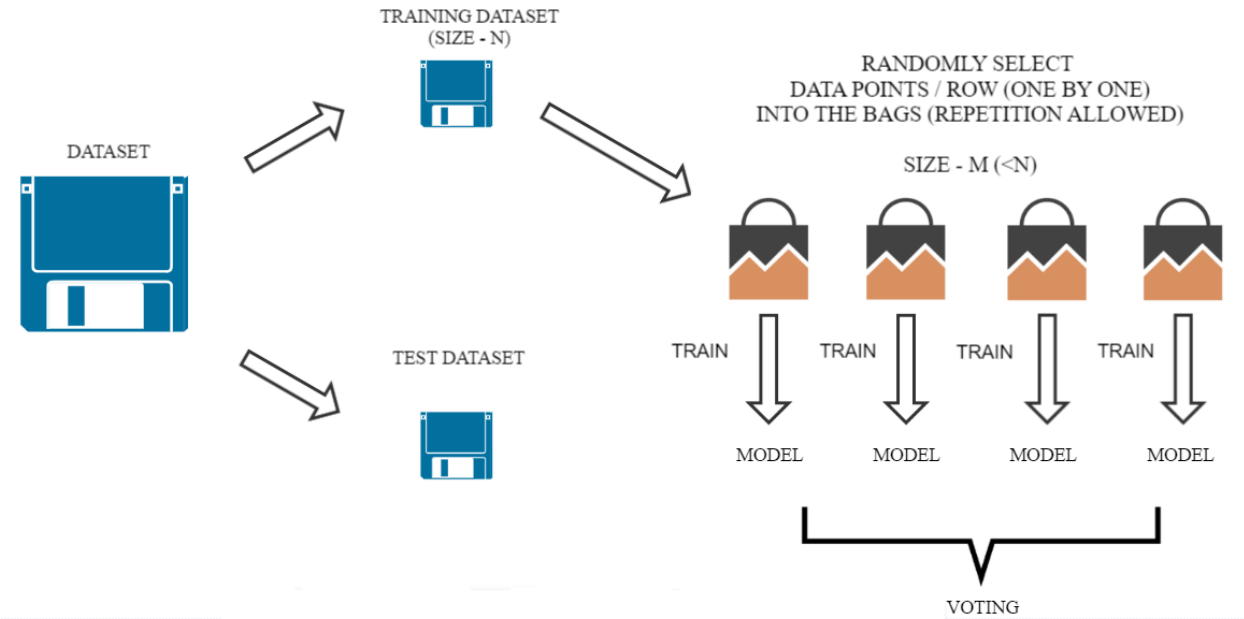
When and Where do we use Ensemble Learning?

1. Single model overfits
2. Results worth the extra training
3. Can be used for classification & regression both.

Popular Ensemble Methods

1. Bootstrap Aggregation (Bagging)

Multiple models of the same learning algorithm trained with subsets of dataset randomly picked from the training dataset.



Bagging Algorithms:

a. Bagged Decision Trees
b. Random Forest
c. Extra Trees

Code Snippet to be used for each of the below examples like Bagged Decision Trees, Random Forest, Extra Trees, and other Boost examples:

```
#Importing Libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn import model_selection
from sklearn.ensemble import BaggingClassifier

#Reading data
data = pd.read_csv(r"Social_Network_Ads.csv")
data.head()
```

```
#Removing User ID, as it's an incremental value that doesn't add to our classification prediction
data.pop('User ID')

#Replacing categorical values to numericals
data['Gender'].replace(['Male','Female'],[1,0],inplace=True)

#Using features: Gender, Age for prediction of Purchased label
feature_cols = ['Gender', 'Age']
X = data[feature_cols] # Features
Y = data.Purchased # Target variable
```

Bagged Decision Trees

Bagging performs best with algorithms with high variance. Decision tree is one of the most popular examples.
In this example, we will see the usage of BaggingClassifier along with DecisionTreeClassifier. A total of 100 trees are used.

```
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
cart = DecisionTreeClassifier()
num_trees = 100
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print("Accuracy for BaggingClassifier:", results.mean())
```

Output: Accuracy for BaggingClassifier: 0.8

Random Forest

RF is an extension of Bagged decision trees.

Samples of the training dataset are taken with replacement, but the trees are constructed in a way that reduces the correlation between individual classifiers. Specifically, rather than greedily choosing the best split point in the construction of the tree, only a random subset of features are considered for each split.

You can construct a Random Forest model for classification using the RandomForestClassifier class.

The example below provides an example of Random Forest for classification with 100 trees.

```
from sklearn.ensemble import RandomForestClassifier

seed = 7
num_trees = 100
max_features = 3
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = RandomForestClassifier(n_estimators=num_trees)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print("Accuracy for RandomForestClassifier:", results.mean())
```

Output: Accuracy for RandomForestClassifier: 0.7975

Extra Trees

Extra Trees are another modification of bagging where random trees are constructed from samples of the training dataset.

You can construct an Extra Trees model for classification using the ExtraTreesClassifier class.

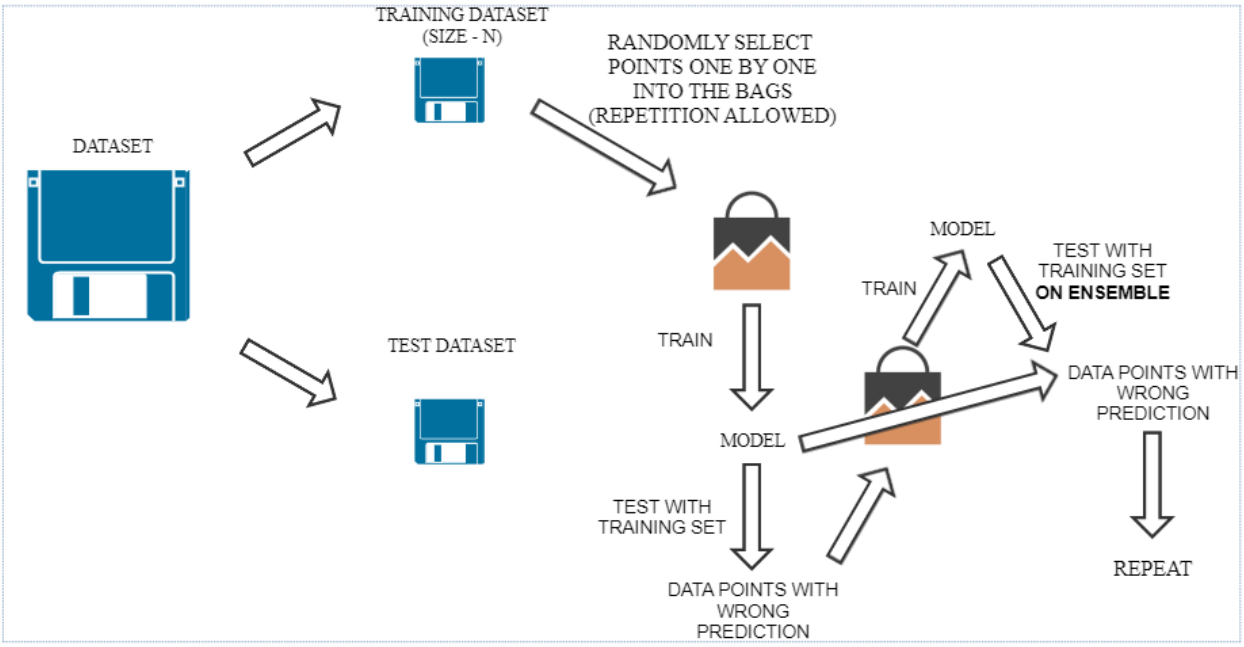
The example below provides a demonstration of extra trees with the number of trees set to 100 and splits chosen from 7 random features.

```
from sklearn.ensemble import ExtraTreesClassifier

seed = 7
num_trees = 100
max_features = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = ExtraTreesClassifier(n_estimators=num_trees)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print("Accuracy for ExtraTreesClassifier:", results.mean())
```

Output: Accuracy for ExtraTreesClassifier: 0.7849999999999999

2. **Boosting**

Boosting is used to create a collection of predictors. In this technique, learners are learned sequentially with early learners fitting simple models to the data and then analysing data for errors. Consecutive trees (random sample) are fit and at every step, the goal is to improve the accuracy from the prior tree. When an input is misclassified by a hypothesis, its weight is increased so that the next hypothesis is more likely to classify it correctly. This process converts weak learners into better performing model.



Most common Boosting techniques are:

a. AdaBoost
b. Stochastic Gradient Boosting

AdaBoost

AdaBoost was perhaps the first successful boosting ensemble algorithm. It generally works by weighting instances in the dataset by how easy or difficult they are to classify, allowing the algorithm to pay or or less attention to them in the construction of subsequent models.

You can construct an AdaBoost model for classification using the AdaBoostClassifier class.

The example below demonstrates the construction of 30 decision trees in sequence using the AdaBoost algorithm.

```
from sklearn.ensemble import AdaBoostClassifier

seed = 7
num_trees = 30
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print("Accuracy for AdaBoostClassifier:", results.mean())
```

Output: Accuracy for AdaBoostClassifier: 0.835

Stochastic Gradient Boosting

Stochastic Gradient Boosting (also called Gradient Boosting Machines) are one of the most sophisticated ensemble techniques. It is also a technique that is proving to be perhaps of the best techniques available for improving performance via ensembles.

You can construct a Gradient Boosting model for classification using the GradientBoostingClassifier class.

The example below demonstrates Stochastic Gradient Boosting for classification with 100 trees.

```
from sklearn.ensemble import GradientBoostingClassifier

seed = 7
num_trees = 100
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print("Accuracy for GradientBoostingClassifier:", results.mean())
```

Output: Accuracy for GradientBoostingClassifier: 0.8149999999999998

Voting Ensemble

To create a **VotingClassifier**, simply aggregate the predictions of each classifier and predict the class that gets the most votes. This majority-vote classifier is called a *hard voting* classifier.

Here's how it's done in Scikit-Learn:

```
# Voting Ensemble for Classification
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
# create the sub models
estimators = []
model1 = LogisticRegression()
estimators.append(('logistic', model1))
model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))
model3 = SVC()
estimators.append(('svm', model3))
# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, Y, cv=kfold)
print("Accuracy for VotingClassifier:", results.mean())
```

Output: Accuracy for VotingClassifier: 0.8324999999999999

Conclusion:

In this module, you learnt about various Ensemble algorithms, and using an example, we found that **AdaBoost** gave us the best results, but it doesn't mean that boosting provides us with the best results every time. It's all about hit and trial, these algorithms have to be tried separately for each use case, and then we can come to a conclusion. In some cases, standalone classifiers also outperforms as compared to the ensemble techniques.