Model Selection And Boosting

How can we know which machine learning model is the best one to solve the problem?
How can we tell whether the problem is "regrssion", "classification" or "clustering" problem?

- Take look at the dependent variables:
  - Clustering: if No dependent variables.
  - Regression: if the dependent variable is continuous outcome
  - Classification: if the dependent variable is categorical outcome

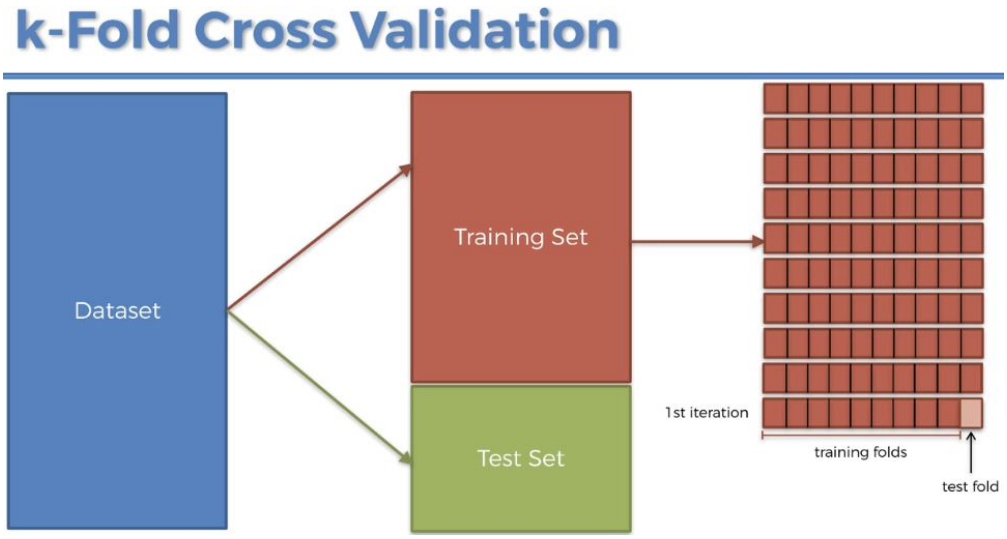
Is my problem a linear or nonlinear problem?
Grid Search will tell us if we should choose linear model (like SVM) or nonlinear model (like kernel SVM).

K-Fold Cross Validation

K-Fold Cross Validation is a very efficient way to evaluate the model performance.

We will get a very different accuracy on the test set, when we run the model and test again on the different test set. Therefore, judging the model performance only on one accuracy on one test set is not super relevant.

K-Fold Cross Validation: Spinning the training set into ten fold (most time K = 10). And we train the model with nine fold and test it on the last remaining fold. Since we have 10 folds, then we can make 10 different combinations of 9 folds to train the model and 1 fold to test it.



That means we can train and test the model on 10 combinations of the training and test sets. Then we can take an average of different accuracy up to 10 evaluations and compute the standard deviation to have look at the variance.

In [1]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

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

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, ra
ndom_state = 0)

# Fitting Kernel SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, y_train)
```

```
C:\Apps\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: T
he default value of gamma will change from 'auto' to 'scale' in version 0.22
to account better for unscaled features. Set gamma explicitly to 'auto' or 's
cale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[2]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=0,
    shrinking=True, tol=0.001, verbose=False)
```

Predicting the test results and Confusion matrix is the first way of evaluating the model.
But NOT the best way!!

In [3]:
```
# Predicting the Test set results
y_pred = classifier.predict(X_test)

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

Out[3]:
```
array([[64,  4],
       [ 3, 29]], dtype=int64)
```

In [4]:
```
# Applying K-Fold Cross Validation
from sklearn.model_selection import cross_val_score
```

cross_val_score will return 10 accuracy for each one of the 10 combinations that will be created through K-Fold Cross Validation.

Each combination is composed of 9 folds to train and 1 fold to test.

cross_val_score arguments:

estimator: it's the model which is the classifier in this example.
X: The data to fit (training set).
y: The dependent variable to try to predict (training set).
cv: The number of folds we want to split the training set into. The most common choice for the cv number is 10.
n_jobs: For very large dataset, set it = -1, means 'all CPUs'. Then you can run faster.

In [5]:
```
# Define a vector that will take the 10 accuracy
# through the 10 combinations created by K-Fold Cross Validation
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train,
                             cv = 10)
accuracies
```

```
C:\Apps\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: T
he default value of gamma will change from 'auto' to 'scale' in version 0.22
to account better for unscaled features. Set gamma explicitly to 'auto' or 's
cale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Apps\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: T
he default value of gamma will change from 'auto' to 'scale' in version 0.22
to account better for unscaled features. Set gamma explicitly to 'auto' or 's
cale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Apps\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: T
he default value of gamma will change from 'auto' to 'scale' in version 0.22
to account better for unscaled features. Set gamma explicitly to 'auto' or 's
cale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Apps\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: T
he default value of gamma will change from 'auto' to 'scale' in version 0.22
to account better for unscaled features. Set gamma explicitly to 'auto' or 's
cale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Apps\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: T
he default value of gamma will change from 'auto' to 'scale' in version 0.22
to account better for unscaled features. Set gamma explicitly to 'auto' or 's
cale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Apps\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: T
he default value of gamma will change from 'auto' to 'scale' in version 0.22
to account better for unscaled features. Set gamma explicitly to 'auto' or 's
cale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Apps\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: T
he default value of gamma will change from 'auto' to 'scale' in version 0.22
to account better for unscaled features. Set gamma explicitly to 'auto' or 's
cale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Apps\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: T
he default value of gamma will change from 'auto' to 'scale' in version 0.22
to account better for unscaled features. Set gamma explicitly to 'auto' or 's
cale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Apps\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: T
he default value of gamma will change from 'auto' to 'scale' in version 0.22
to account better for unscaled features. Set gamma explicitly to 'auto' or 's
cale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Apps\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: T
he default value of gamma will change from 'auto' to 'scale' in version 0.22
to account better for unscaled features. Set gamma explicitly to 'auto' or 's
cale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[5]:
```
array([0.80645161, 0.96666667, 0.8       , 0.93333333, 0.86666667,
       0.8       , 0.93333333, 0.93333333, 0.96666667, 0.96551724])
```

It means when we test the performance of the model on the different 1 test set, we will get different accuracy.

In [6]:
```
print(accuracies.mean())
print(accuracies.std())
```

```
0.8971968854282535
0.0680430514209002
```

Grid Search

Grid Search is the technique to improve model performance by finding the optimal values of the hyper parameters. It helps us to know which parameter to select and what's its optimal value when we make a machine learning model

Any Machine Learning model is composed of 2 types of parameters:

- Parameters are learned through the machine learning
- Hyper parameters: Parameters we choosed, ie. kernel in SVM model, penalty paramaters...

Normally, Grid Search can be applied after fitting the model to the training set. Because Grid Search needs the trained machine learning as input.

注：In the following example, we evaluate the model without Grid Search first. Then do the Grid Search to see the difference. So adding Grid Search section after K-Fold Cross Validation section (直接承接上一章)

In [7]:
```
# Applying Grid Search to find the best model and the best parameters
from sklearn.model_selection import GridSearchCV
```

Specifying the different parameters of which we want to find the optimal values.

For example, we used 'kernel'='rbf' and 'C' = 1 (default) in SVC class.

- C: The parameter for regularization to prevent overfitting. Increasing C the more it will prevent overfitting, so don't increase too much otherwise, it will cause underfitting problem.
- gamma: For nonlinear only, Kernel coefficient for 'rbf', 'poly' and 'sigmoid'

And we will include both 'kernel' and 'C' and 'gamma' in the Grid Search model.
Grid Search will tell the kernel should be linear or nonlinear, what's the best value of 'C' and 'gamma'.

In [8]:
```
# a list of dictionaries, so the key identifiers in the dictionary
# will be the parameters that we want to optimize, and each of these key ident
ifiers
# we will give it several values to be tested by the Grid Search Model.
# And among these values the grid search model will find the best one.

# The first option (first dictionary) of grid search will investigate
# whether it's linear model and what's the best value of 'C'

# The second option (second dictionary) of grid search is nonlinear option
# so we will set 'gamma'(nonlinear only) in this option
# since 'gamma' default is 'auto' which is 1/n_features, and we have 2 feature
s
# so the range in the example will be [0.001 - 0.5]
parameters = [{'C':[1, 10, 100, 1000], 'kernel': ['linear']},
              {'C':[1, 10, 100, 1000], 'kernel': ['rbf'],
               'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]}]
```

In [9]:
```
# Grid Search selects the best parameters based on one performance metric (sco
ring)
# it can be 'accuracy' or 'precision' or 'recall'...
# The comment practice is useing 'accuracy'

# cv is the K-Fold Cross Validation. It means Grid Search will evaluate
# the performance of each model with their own set of parameters.

grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10, # 10 Fold Cross Validation will be used
                           n_jobs = -1)
```

In [10]:
```
# Once we create grid search object, then we can train it.
grid_search = grid_search.fit(X_train, y_train)

# best accuracy score with the best selections of the parameter list we want t
o try
best_accuracy = grid_search.best_score_
print(best_accuracy)
# best parameters have been selected by grid search
best_parameters = grid_search.best_params_
print(best_parameters)
```

```
0.9033333333333333
{'C': 1, 'gamma': 0.4, 'kernel': 'rbf'}

C:\Apps\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:813: D
eprecationWarning: The default of the `iid` parameter will change from True t
o False in version 0.22 and will be removed in 0.24. This will change numeric
results when test-set sizes are unequal.
  DeprecationWarning)
```

XGBoost

XGBoost is a gradient boosting with trees. The most powerful tool for the Machine Learning!!!

It can be used on both Regression and Classification:

- from xgboost import XGBRegressor OR *xgboost.XGBRegressor*
- from xgboost import XGBClassifier OR *xgboost.XGBClassifier* Recommend to use XGBoost to work on a large dataset!!!

In [11]:
```
# Importing the dataset
dataset = pd.read_csv('Churn_Modelling.csv')

# Independent variables: CreditScore,Geography,Gender,Age,
# Tenure,Balance,NumOfProducts,HasCrCard,IsActiveMember,EstimatedSalary
# ANN will determine which independent variable will be more important.
X = dataset.iloc[:,3:13].values
y = dataset.loc[:,'Exited'].values

# Encodes any categorical data in the dataset
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
# In the dataset, only 'Geography' and 'Gender' need to be encoded.
labelencoder_X_1 = LabelEncoder()
X[:,1] = labelencoder_X_1.fit_transform(X[:,1])
labelencoder_X_2 = LabelEncoder()
X[:,2] = labelencoder_X_2.fit_transform(X[:,2])

# The categorical variables are NOT ordinal
# (No relational order between the categorical variables)
# For example, France (2) is NOT higher than Germany (1)
# So we need create dummy variables for these categorical variables
onehotencoder = OneHotEncoder(categorical_features = [1])
# In order to convert X to be a matrix, we need add '.toarray()' in the end.
X = onehotencoder.fit_transform(X).toarray()
# To remove one dummy variable in order to avoid falling into the dummy variab
le trap.
X = X[:,1:]
```

```
C:\Apps\Anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py:415: F
utureWarning: The handling of integer data will change in version 0.22. Curre
ntly, the categories are determined based on the range [0, max(values)], whil
e in the future they will be determined based on the unique values.
If you want the future behaviour and silence this warning, you can specify "c
ategories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the cate
gories to integers, then you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
C:\Apps\Anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py:451: D
eprecationWarning: The 'categorical_features' keyword is deprecated in versio
n 0.20 and will be removed in 0.22. You can use the ColumnTransformer instea
d.
  "use the ColumnTransformer instead.", DeprecationWarning)
```

All Deep Learning MUST have feature scaling.

But XGBoost is based on the models of decision trees. So the feature scaling is unnecessary.

In [12]:
```
# Splitting the dataset into the Training set and Test set
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_stat
e=0)
```

XGBClassifier Arguments:

- learning_rate: We had in deep learning
- n_estimators: the number of trees

In the example, it will be the most simple case -- Use all default settings.

In [13]:
```
# Fitting XGBoost to the Training set
from xgboost import XGBClassifier as XGBC
classifier = XGBC().fit(X_train, y_train)
```

Predicting And Evaulating

In [14]:
```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

# Making the Confusion Matrix
cm = confusion_matrix(y_test,y_pred)
print(cm)

# Applying K-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train,
                             y = y_train, cv = 10)
print(accuracies.mean())
print(accuracies.std())
```

```
[[1521   74]
 [ 197  208]]
0.8629994451163204
0.010677872171663988
```