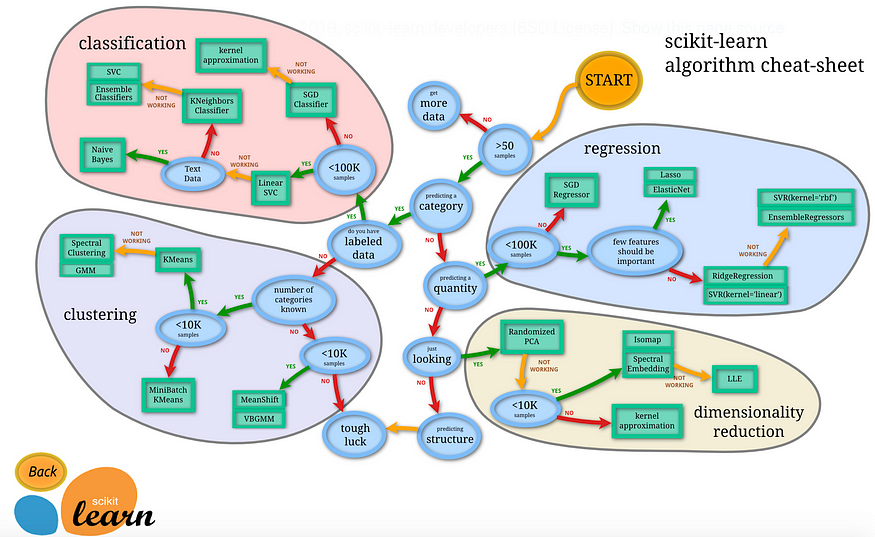
**Machine Learning Classifiers**

Machine learning classifiers are models used to predict the category of a data point when labeled data is available (i.e. supervised learning). Some of the most widely used algorithms are logistic regression, Naïve Bayes, stochastic gradient descent, k-nearest neighbors, decision trees, random forests and support vector machines.

**Choosing the Right Estimator**

Determining the right estimator for a given job represents one of the most critical and hardest part while solving machine learning problems. Each estimator is suitable for a specific type of data and problem. [Scikit-learn](https://scikit-learn.org/stable/index.html), one of the most popular Python libraries for machine learning, provides the following chart to guide the user on the decision process for choosing the most appropriate estimator.



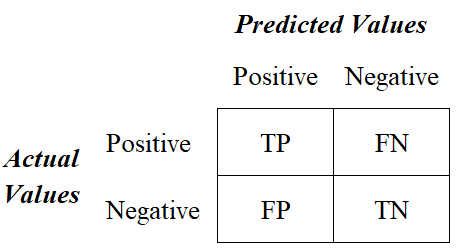
**Performance Evaluation Metrics**

Classification models must be evaluated to determine their degree of effectiveness for performing a specific task. While good classification models are useful for prediction purposes, poor classification models lead to unreliable outcomes, and thus, are not useful for the user.

Performance evaluation metrics are based on the total number of the following variables:

* **True Positives**: outcome correctly predicted as positive class
* **True Negatives**: outcome correctly predicted as negative class
* **False Positives**: outcome incorrectly predicted as positive class
* **False Negatives**: outcome incorrectly predicted as negative class

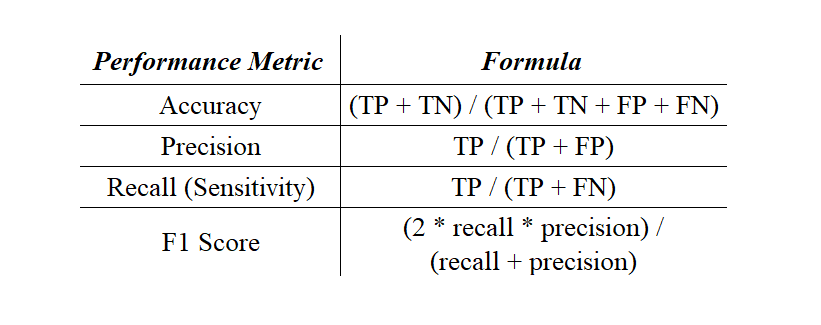
which are visually represented in a matrix (i.e. confusion matrix) where one of its axis is the label that the machine learning model predicted, and the other the actual label:



Confusion Matrix for the Binary Classification

There are four main performance metrics used to evaluate the effectiveness of classification models:

* **Accuracy**: test’s ability to correctly predict both classes
* **Precision**: test’s ability to correctly detect positive classes from all predicted positive classes
* **Recall (Sensitivity):**test’s ability to correctly detect positive classes from all actual positive classes
* **F1 Score**: harmonic mean of precision and recall



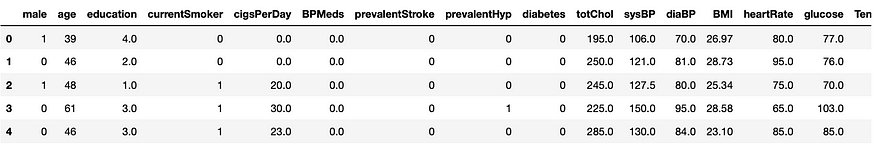
Performance Metrics Formulas

For the following example, let’s evaluate the performance of five different classification models (i.e. logistic regression, support vector classifier, decision tree, random forest and Gaussian Naïve Bayes classifier) on the [Framingham Heart Study data set](https://github.com/rsalaza4/Machine-Learning-Classifiers-Comparison/blob/master/Data%20Sets/framingham.csv) (a study conducted to identify the common factors that contribute to cardiovascular diseases) to determine the one that leads to the most reliable outcomes.

The following Python code will be divided into five major steps. Lines of comments are included to provide a brief explanation and guide you through the coding process.

|  |  |
| --- | --- |
|  | # Import required libraries |
|  | import numpy as np |
|  | import pandas as pd |
|  | import matplotlib.pyplot as plt |
|  | import seaborn as sns |
|  | %matplotlib inline |
|  |  |
|  | # Set random seed |
|  | np.random.seed(42) |
|  |  |
|  | # Load csv file |
|  | df = pd.read\_csv('framingham.csv') |
|  |  |
|  | # View top 5 rows |
|  | df.head() |

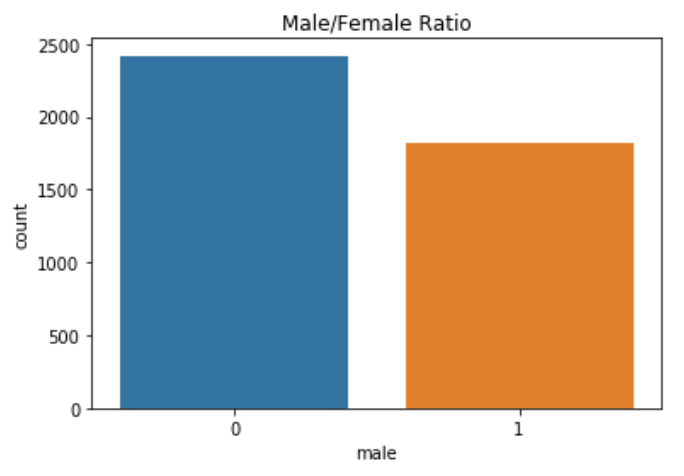
**Step #1: Data Loading**



Framingham Heart Study Top Rows

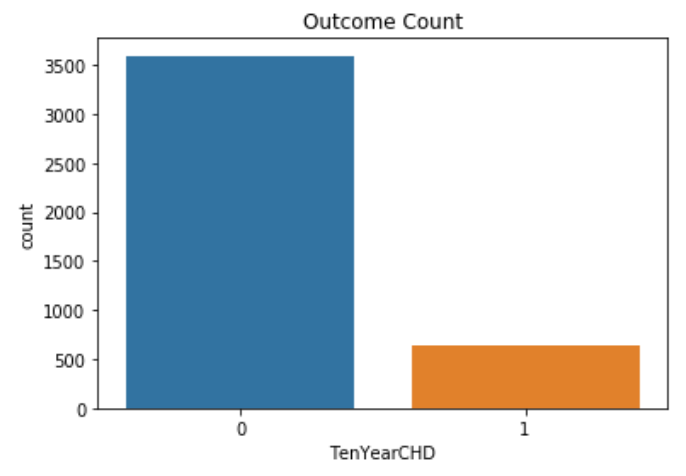
**Step #2: Exploratory Data Analysis**

|  |  |
| --- | --- |
|  | # Visualize male/female ratio |
|  | sns.countplot(x=df["male"]).set\_title("Male/Female Ratio") |
|  |  |
|  | # Visualize the classes distributions |
|  | sns.countplot(x=df["TenYearCHD"]).set\_title("Outcome Count") |
|  |  |
|  | # Visualize the classes distributions by gender |
|  | sns.countplot(x="TenYearCHD", hue="male", data=df).set\_title('Outcome Count by Gender') |



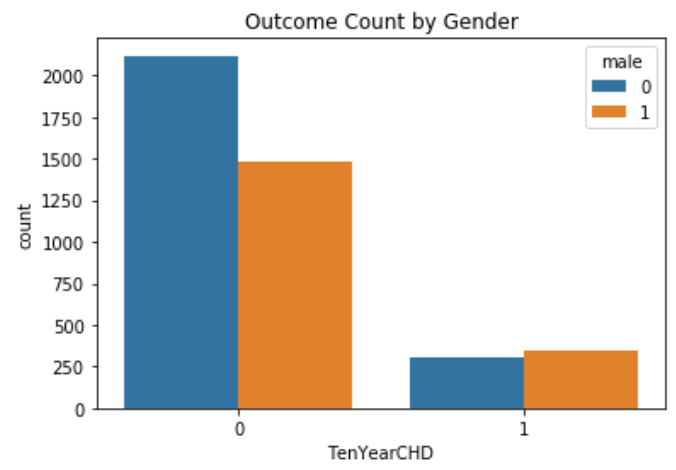
Male/Female Ratio

According to the plot above, the Framingham Heart Study contains more data points corresponding to women than to men.



Outcome Count

The plot above reveals that the Framingham Heart Study is a heavily unbalanced data set. Most of the data points correspond to a negative class (i.e. low risk of developing a cardiovascular disease in ten years). Further data balancing will be necessary to address this issue.



Outcome Count by Gender

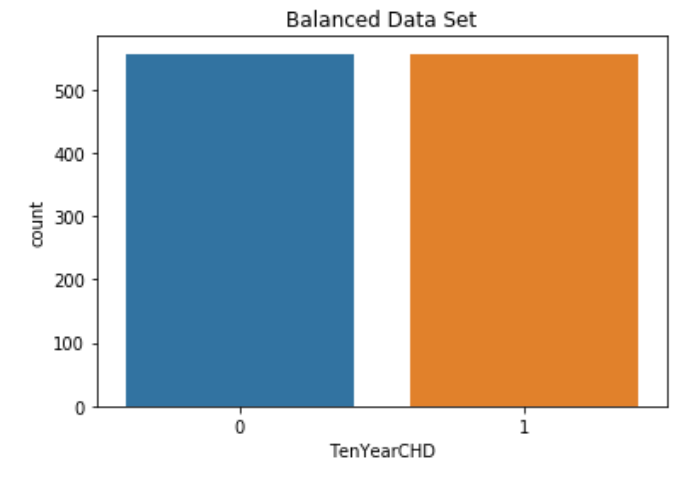
Interesting. Even though the total number of data points corresponding to men was lower, the plot above suggests that the risk of developing a cardiovascular disease on men is higher than on women.

**Step #3: Data Cleaning**

|  |  |
| --- | --- |
|  | # Check if there are any null values |
|  | df.isnull().values.any() |
|  |  |
|  | # Remove null values |
|  | df = df.dropna() |
|  |  |
|  | # Check if there are any null values |
|  | df.isnull().values.any() |

**Step #4: Data Balancing**

|  |  |
| --- | --- |
|  | # Specify features columns |
|  | X = df.drop(columns="TenYearCHD", axis=0) |
|  |  |
|  | # Specify target column |
|  | y = df["TenYearCHD"] |
|  |  |
|  | # Import required library for resampling |
|  | from imblearn.under\_sampling import RandomUnderSampler |
|  |  |
|  | # Instantiate Random Under Sampler |
|  | rus = RandomUnderSampler(random\_state=42) |
|  |  |
|  | # Perform random under sampling |
|  | df\_data, df\_target = rus.fit\_resample(X, y) |
|  |  |
|  | # Visualize new classes distributions |
|  | sns.countplot(df\_target).set\_title('Balanced Data Set') |

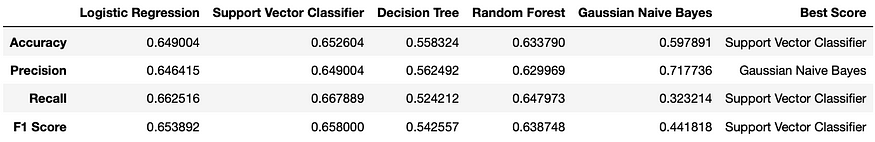


The plot above shows an equal number of classes is equal after having used the Random Under Sampling technique to balance the data set.

**Step #5: Models Building and Performance Evaluation**

|  |
| --- |
| # Import required libraries for performance metrics |
|  | from sklearn.metrics import make\_scorer |
|  | from sklearn.metrics import accuracy\_score |
|  | from sklearn.metrics import precision\_score |
|  | from sklearn.metrics import recall\_score |
|  | from sklearn.metrics import f1\_score |
|  | from sklearn.model\_selection import cross\_validate |
|  |  |
|  | # Define dictionary with performance metrics |
|  | scoring = {'accuracy':make\_scorer(accuracy\_score), |
|  | 'precision':make\_scorer(precision\_score), |
|  | 'recall':make\_scorer(recall\_score), |
|  | 'f1\_score':make\_scorer(f1\_score)} |
|  |  |
|  | # Import required libraries for machine learning classifiers |
|  | from sklearn.linear\_model import LogisticRegression |
|  | from sklearn.svm import LinearSVC |
|  | from sklearn.tree import DecisionTreeClassifier |
|  | from sklearn.ensemble import RandomForestClassifier |
|  | from sklearn.naive\_bayes import GaussianNB |
|  |  |
|  | # Instantiate the machine learning classifiers |
|  | log\_model = LogisticRegression(max\_iter=10000) |
|  | svc\_model = LinearSVC(dual=False) |
|  | dtr\_model = DecisionTreeClassifier() |
|  | rfc\_model = RandomForestClassifier() |
|  | gnb\_model = GaussianNB() |
|  |  |
|  | # Define the models evaluation function |
|  | def models\_evaluation(X, y, folds): |
|  |  |
|  | ''' |
|  | X : data set features |
|  | y : data set target |
|  | folds : number of cross-validation folds |
|  |  |
|  | ''' |
|  |  |
|  | # Perform cross-validation to each machine learning classifier |
|  | log = cross\_validate(log\_model, X, y, cv=folds, scoring=scoring) |
|  | svc = cross\_validate(svc\_model, X, y, cv=folds, scoring=scoring) |
|  | dtr = cross\_validate(dtr\_model, X, y, cv=folds, scoring=scoring) |
|  | rfc = cross\_validate(rfc\_model, X, y, cv=folds, scoring=scoring) |
|  | gnb = cross\_validate(gnb\_model, X, y, cv=folds, scoring=scoring) |
|  |  |
|  | # Create a data frame with the models perfoamnce metrics scores |
|  | models\_scores\_table = pd.DataFrame({'Logistic Regression':[log['test\_accuracy'].mean(), |
|  | log['test\_precision'].mean(), |
|  | log['test\_recall'].mean(), |
|  | log['test\_f1\_score'].mean()], |
|  |  |
|  | 'Support Vector Classifier':[svc['test\_accuracy'].mean(), |
|  | svc['test\_precision'].mean(), |
|  | svc['test\_recall'].mean(), |
|  | svc['test\_f1\_score'].mean()], |
|  |  |
|  | 'Decision Tree':[dtr['test\_accuracy'].mean(), |
|  | dtr['test\_precision'].mean(), |
|  | dtr['test\_recall'].mean(), |
|  | dtr['test\_f1\_score'].mean()], |
|  |  |
|  | 'Random Forest':[rfc['test\_accuracy'].mean(), |
|  | rfc['test\_precision'].mean(), |
|  | rfc['test\_recall'].mean(), |
|  | rfc['test\_f1\_score'].mean()], |
|  |  |
|  | 'Gaussian Naive Bayes':[gnb['test\_accuracy'].mean(), |
|  | gnb['test\_precision'].mean(), |
|  | gnb['test\_recall'].mean(), |
|  | gnb['test\_f1\_score'].mean()]}, |
|  |  |
|  | index=['Accuracy', 'Precision', 'Recall', 'F1 Score']) |
|  |  |
|  | # Add 'Best Score' column |
|  | models\_scores\_table['Best Score'] = models\_scores\_table.idxmax(axis=1) |
|  |  |
|  | # Return models performance metrics scores data frame |
|  | return(models\_scores\_table) |
|  |  |
|  | # Run models\_evaluation function |
|  | models\_evaluation(df\_data, df\_target, 5) |

**Final Outcome**



Models’ Performance Measures Scores Table