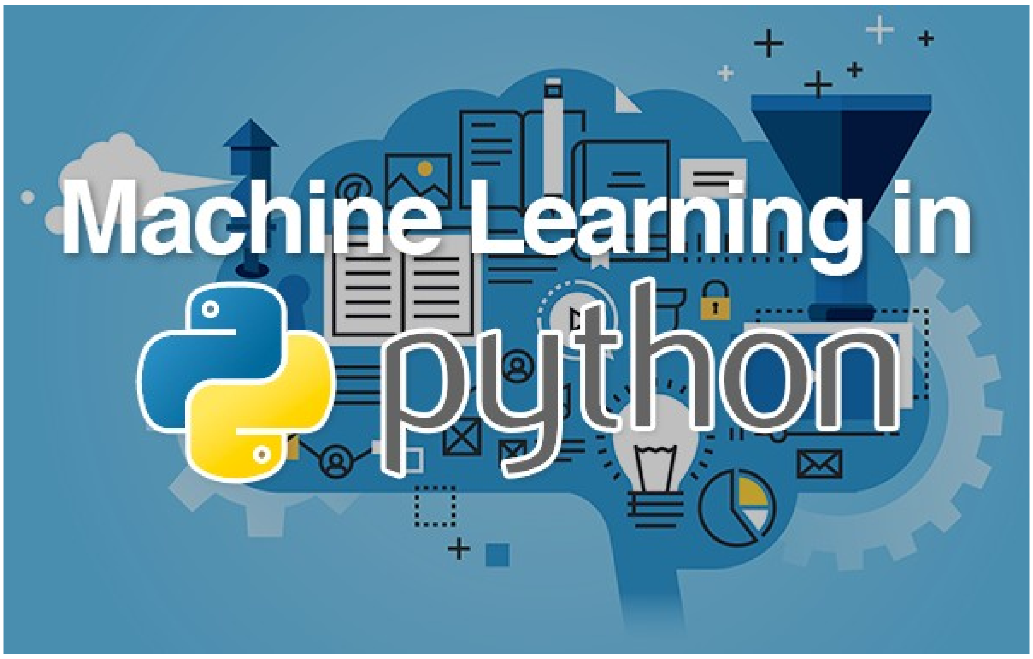
Classification using Machine Learning

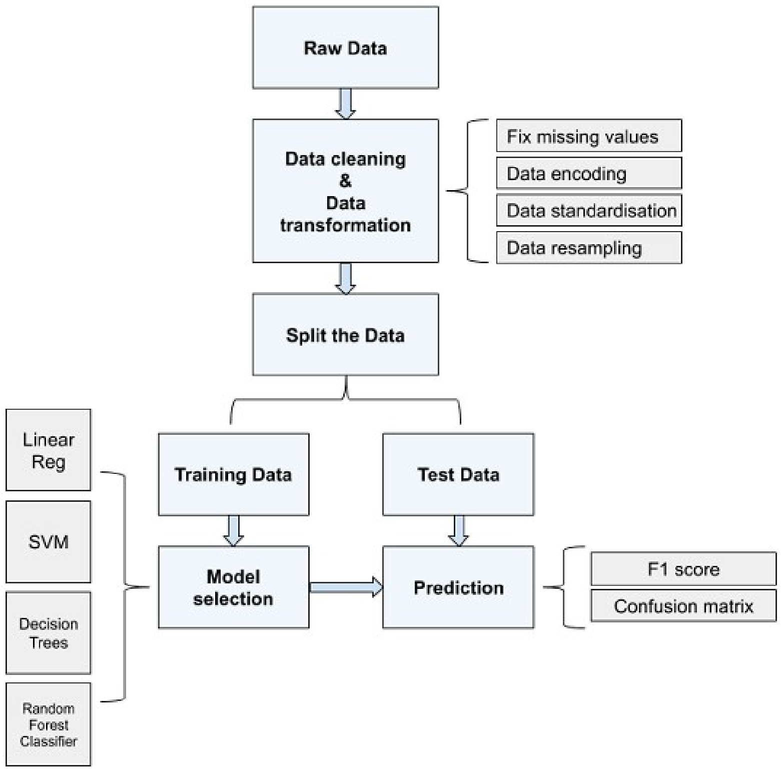


It all starts with operational reports coming out of systems, questionnaires, SQL databases, excel worksheets , a CRM database and then moving up to create metrics-based reporting and eventually advanced data visualizations. Visualization tools such as Tableau, Power BI or QlikView are leaders in the visualization category

As the organizations move towards predictive analytics and other advanced analytics, a variety of tools can be used ranging from open source tools to paid ones. The most popular open source tools are Python and R and there is a thriving community that contributes to this knowledge repository.

How do you make business more time-efficient, slash costs and drive up sales? The question is timeless but not rhetorical. In the next few minutes of your reading time, I will apply a few classification algorithms to demonstrate how the use of the data analytic approach can contribute to that end. Together we'll create a predictive model that will help us customise the client datasets we hand over to the telemarketing team so that they could concentrate resources on more promising clients first.

On course to that, we'll perform a number of actions on the dataset. First, we'll clean, encode, standardize and resample the data. Once the data is ready, we'll try four different classifiers on the training subset, make predictions and visualize them with a confusion matrix, and compute F1 score to elect the best model. These steps have been put together in the schema:



The dataset we'll be using here is not new to the town and you have probably come across it before. The data sample of 41,118 records was collected by a Portuguese bank between 2008 and 2013 and contains the results of a telemarketing campaign including customer's response to the bank's offer of a deposit contract (the binary target variable 'y'). That response is exactly what we are going to predict with the model. The dataset is available at Irvine's Machine Learning Repository.   So let's get started!

First read data into pandas data frame

```
df = pandas.read_csv("bank_updated1.csv")
print(df) # print data
print(df.shape) # check rows & colmns
```

Data Cleaning, Feature Selection, Feature Transformation

With df.isnull().sum() query we make sure there are no missing values in the dataset (if there were any, df.dropna(subset = ['feature_name'], inplace=True) would drop them from the respected column)...or use **fillna** to fill the missing values.

In this example, I used Tableau Prep Builder data cleaning tool to trace and drop outliers, to make sure values in numerical features aren't strings, to rename some columns and to get rid of a few irrelevant ones, i.e. 'contact', 'month', 'day_of_week', 'duration' (these columns describe a telephone call that has happened already, therefore they should not be used in our predictive model).

Before proceeding we run print(df.dtypes)

we get

```
age          int64
job          object
marital      object
education    object
default      object
balance      int64
housing      object
loan         object
contact      object
day          int64
month        object
duration     int64
campaign     int64
pdays        int64
previous     int64
poutcome     object
y            object
```

Machine learning models require to work with numeric int or float... so we need to convert all with objects type to numeric.

Next we shall transform the non-numerical labels of the categorical variables to numerical ones and convert them to integers. We do it like this:

```
# we will work with below colms, we create a subset of the data frame
subset =
df[['age','housing','loan','job','education','marital','balance','default','campaign','pre
vious','y']]
print(subset)
subset['education'].replace({'primary':0, 'secondary':1,'tertiary':2, 'unknown':3},
inplace=True)
subset['marital'].replace({'married':0, 'single':1, 'divorced':2, 'unknown':3},
inplace=True)
subset['default'].replace({'yes':0, 'no':1, 'unknown':2}, inplace=True)
subset['job'].replace({'admin.':0,
              'blue-collar':1,
              'housemaid':2,
              'entrepreneur':3,
              'management':4,
              'retired':5,
              'self-employed':6,
              'student':7,
              'services': 8,
              'technician':9,
              'unemployed':10,
              'unknown':11}, inplace=True)
subset['housing'].replace({'yes':0, 'no':1, 'unknown':2}, inplace=True)
subset['loan'].replace({'yes':0, 'no':1, 'unknown':2}, inplace=True)
```

Alternatively you can use **LabelEncoder**() check
https://medium.com/@contactsunny/label-encoder-vs-one-hot-encoder-in-machine-learning-3fc273365621

Many machine learning algorithms make assumptions about your data. It is often a very good idea to prepare your data in such way to best expose the structure of the problem to the machine learning algorithms that you intend to use. In this chapter you will discover how to prepare your data for machine learning in Python using scikit-learn. Below are used in data preparation
will know how to:
1. Rescale data.
2. Standardize data.
3. Normalize data.
4. Binarize data.

Check machine Learning mastery book , Page 47.

We will apply StandardScaler from sklearn.preprocessing toolbox to **standardize** the numerical values of the other features that we expect to find use of in the model. The method standardizes features by removing the mean and scaling to unit variance:
in this case we standardize balance to be with a scale of -1 to 1...This is step is optional!

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
subset['balance'] = scaler.fit_transform(df[['balance']]).reshape(-1,1)
print(subset['balance'])
```

Feature Selection For Machine Learning

The data features that you use to train your machine learning models have a huge influence on the performance you can achieve. Irrelevant or partially relevant features can negatively impact model performance.

In this lesson you will discover automatic feature selection techniques that you can use to prepare your machine learning data in Python with scikit-learn. After completing this lesson you will know how to use:

1. Univariate Selection.
2. Recursive Feature Elimination. (RFE)
3. Principle Component Analysis.
4. Feature Importance.

Check machine Learning mastery book , Page 52.

Let's now rank the features of the dataset with recursive feature elimination (RFE) method and Random Forest Classifier algorithm as its estimator:

First we split data into X – input variables, y – Outcome/Output/target variables.

```
# split data
array = subset.values   # convert the subset to an array
X = array[:, 0:10] # means 0 - 9    Input variables
y = array[:,10]  # 9th is counted here   : output variable
```

# apply RFE

Below code is optional but gives you top 6 best features that contribute to variable y, that is diabetes outcome.

```
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=40)
rfe = RFE(rfc, 6)
rfe_fit = rfe.fit(X, y)
```

```
print("Num Features: %s" % (rfe_fit.n_features_))
print("Selected Features: %s" % (rfe_fit.support_))
print("Feature Ranking: %s" % (rfe_fit.ranking_))
```

Output:

```
Num Features: 6
Selected Features: [ True False False  True  True False  True False  True  True]
Feature Ranking: [1 3 4 1 1 2 1 5 1 1]
```

At a later stage, when we'll be building a predictive model, we will make use of this feature ranking

Class imbalance

Class imbalance is the problem that often comes along with such classification cases as fraudulent credit card transactions or the results of online campaigns, for instance. After executing **subset['y'].value_counts**() query we see that the two classes of the variable 'y' are not represented equally in our dataset also. After data cleaning there are 4000 records belonging to the class '0' and only 521 records of the class '1' in the target variable 'y'. Prior to splitting the data into the training and testing samples, we should think of oversampling or undersampling the data.

To resample the data, let's apply SMOTE method for oversampling from imblearn.over_sampling toolbox (for this step you may need to install imblearn package with Pip or Conda first):

```
from imblearn.over_sampling import SMOTE
sm=SMOTE(ratio='auto', kind='regular',random_state=42)
```

```
X_sampled,y_sampled=sm.fit_sample(X,y)
```

Check https://towardsdatascience.com/a-deep-dive-into-imbalanced-data-over-sampling-f1167ed74b5

Building a predictive model

Now that the data has been prepared, we are ready to train our model and make predictions. Let's first split the data into the training and testing sets:

```
from sklearn import model_selection
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X_sampled,
y_sampled,test_size= 0.30,
                                                random_state=10)
```

Above we use (test_size=0.30) 70% of data for training and 30% of data for testing our model accuracy , random_state = 10 represent the random ratio used in pick the 70%:30% from the data set.

We will try five classification algorithms, i.e. Logistic Regression, Support Vector Machine, Decision Trees, and Random Forest, and then compute their F1 scores using a user-defined scorer function to choose the classifier with the highest score:

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
```

We create objects using the above models

```
lr = LogisticRegression(C=1, solver='lbfgs')

clf = SVC(kernel='rbf', gamma='auto')
dtree = DecisionTreeClassifier(criterion="entropy", max_depth=4)
rfc = RandomForestClassifier(n_estimators=40,random_state=42)
knn = KNeighborsClassifier()
```

Next we training out data with any of the five...i use knn

```
knn.fit(X_train, Y_train)  # learning process starts here
```

***NB**: we fit only the train data to the model.*

Next, we ask the model to predict the X_test.

```
predictions = knn.predict(X_test)

print(predictions)
```

We now compare the predictions with the known Y_test.

We Find accuarcy score.

```
from sklearn.metrics import accuracy_score

print(accuracy_score(Y_test, predictions))
```

We view the confusion matrix

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(Y_test, predictions))
```

We get the classification report

```
from sklearn.metrics import classification_report
print(classification_report(Y_test, predictions))
```

Output.

```
0.7941666666666667
[[ 763  464]
 [  30 1143]]
             precision    recall  f1-score   support

          no       0.96      0.62      0.76      1227
         yes       0.71      0.97      0.82      1173

    accuracy                           0.79      2400
   macro avg       0.84      0.80      0.79      2400
weighted avg       0.84      0.79      0.79      2400
```

Explanations

The algorithm performed a possible 79% accuracy using KNN.

Trying other algorithms.

Change these lines to use **RandomForestClassifier**

```
rfc.fit(X_train, Y_train)  # learning process
predictions = rfc.predict(X_test)
```

Output:

```
0.9120833333333334
[[1162   65]
 [ 146 1027]]
             precision    recall  f1-score   support

          no       0.89      0.95      0.92      1227
```

```
         yes       0.94      0.88      0.91      1173

    accuracy                           0.91      2400
   macro avg       0.91      0.91      0.91      2400
weighted avg       0.91      0.91      0.91      2400
```

RandomForestClassifier shows better results than KNN, only for this dataset.

Try other algorithms

Check machine Learning mastery book , **Page** 111.

Great! We've cleaned and transformed the data, selected the most relevant features, elected the best model and made a prediction with a decent score. Now we have a model that should help us customize the client databases we hand over to the telemarketing team so that they could center their efforts on those better positioned to react in the affirmative to the campaign first.

Model usage:

```
subset = df[['age','housing','loan','job','education','marital','balance','default','campaign','previous','y']]
```

Above is the subset we used.

We will now prepare data based on this dataset, we test 2 members and find out if they will be interested with the bank offer!

```
Newdata =[[45,1,2,2,2,1,300,1,2,3], [20,1,2,2,2,1,800,1,2,3]] # I use codes since data was
transformed before training

observation = rfc.predict(Newdata)
print(observation)
```

END