University of Ferhat Abbas 1

Faculty of sciences

Computing science department

Machine learning project:

Forest Cover Type Prediction

Submitted by:

BERRIMI Mohamed

Supervised by :

Pr. Moussaoui Abdelouaheb

Abstract

Machine learning techniques are now used in many fields, whether in classification tasks or regressions . in this project we are interested to apply some Machine learning methods to predict and solve the ***Forest Cover Type Prediction*** *challenge*.

Problem Description :

In this competition you are asked to predict the forest cover type (the predominant kind of tree cover) from strictly cartographic variables (as opposed to remotely sensed data). The actual forest cover type for a given 30 x 30 meter cell was determined from US Forest Service (USFS) Region 2 Resource Information System data. Independent variables were then derived from data obtained from the US Geological Survey and USFS. The data is in raw form (not scaled) and contains binary columns of data for qualitative independent variables such as wilderness areas and soil type.

This study area includes four wilderness areas located in the Roosevelt National Forest of northern Colorado. These areas represent forests with minimal human-caused disturbances, so that existing forest cover types are more a result of ecological processes rather than forest management practices.[2]

Dataset description :

The complete dataset is composed by 581,012 instances, where each observation (instance) corresponds to a 30 x 30 meters cell. For each observation, 12 measures (attributes) are given. However, two categorical properties (wilderness area and soil type) were binarized, in order to have a dataset composed only by numerical attributes. So, the final dataset is composed by 10 quantitative variables, 4 binary wilderness areas and 40

binary soil type variables, given a total of 54 columns of data. Below, some description about these attributes is given:
• Elevation: Elevation in meters.
• Aspect: Aspect in degrees azimuth.
• Slope: Slope in degrees. •
Horizontal distance to hydrology: Horizontal distance to nearest surface water features.
• Vertical distance to hydrology: Vertical distance to nearest surface water features.
• Horizontal distance to roadways: Horizontal distance to nearest roadway.
• Hillshade 9am: Hillshade index at 9am, summer solstice (0 to 255).
• Hillshade Noon: Hillshade index at noon, summer soltice (0 to 255).
• Hillshade 3pm: Hillshade index at 3pm, summer solstice (0 to 255).
• Horizontal distance to fire points: Horizontal distance to nearest wildfire ignition points.
• Wilderness area: Wilderness area designation. 4 binary columns, 0 (absence) or 1 (presence).
• Soil type: Soil type designation. 40 binary columns, 0 (absence) or 1 (presence).

• Cover Type: Forest cover type designation. This is the attribute to be predicted (1 to 7).

Demarche :

The dataset used in this project is available from

As it’s mentioned in the description , the problem to solve here is a classification task , because we are going to predict the type of the forest

I have used *Google Colab* Free cloud platform to solve this challenge , because of two main objectives :

- Get to know how to work with Cloud Machine learning platforms.
- Accelerate the processing time. Accelerate the processing time.

1. *Programming language*

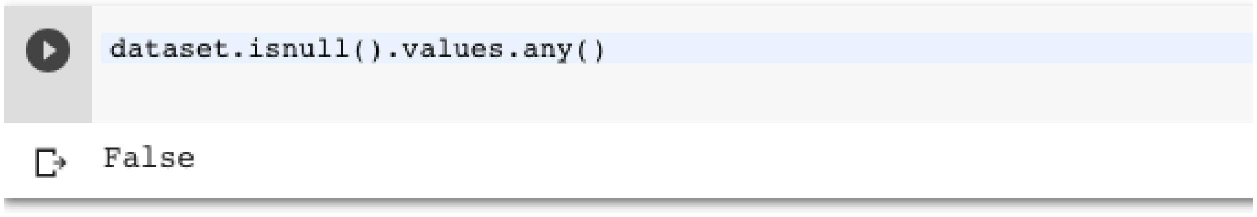
The programming language chosen is python , this language is chosen here to get more familiar with the language because we are up to a final project where we are going to use Python as programming language .

2. Pre-processing :

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format.

In the pre processing first step , we are interested in choosing the variables that we are going to work with .

i. Test if there are NA's:



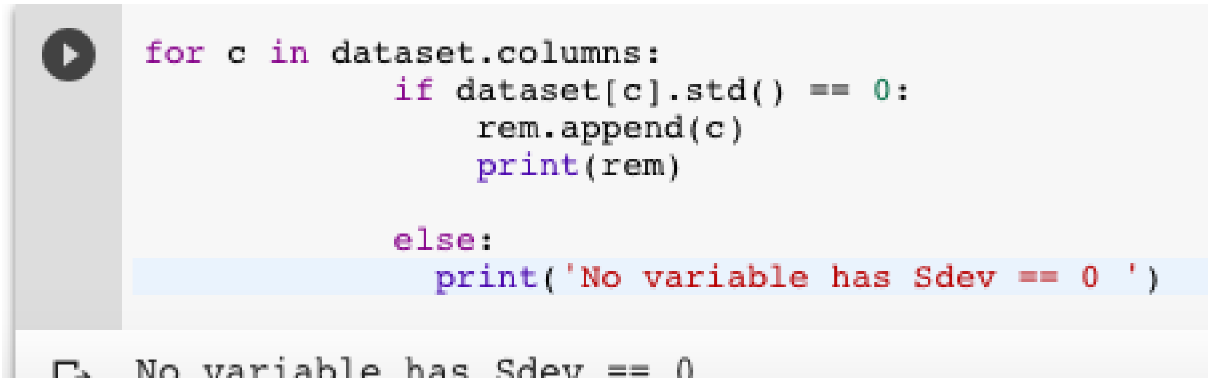
```
dataset.isnull().values.any()
```

```
False
```

ii. *Test if the are variables that doesn't effect the outcome of the class :*

One way to test this is to look if the Sdev of any variable is equal to 0 ,

if so, the values of this variable are identical , this variable doesn't have any influence on the outcome , it need to be omitted .

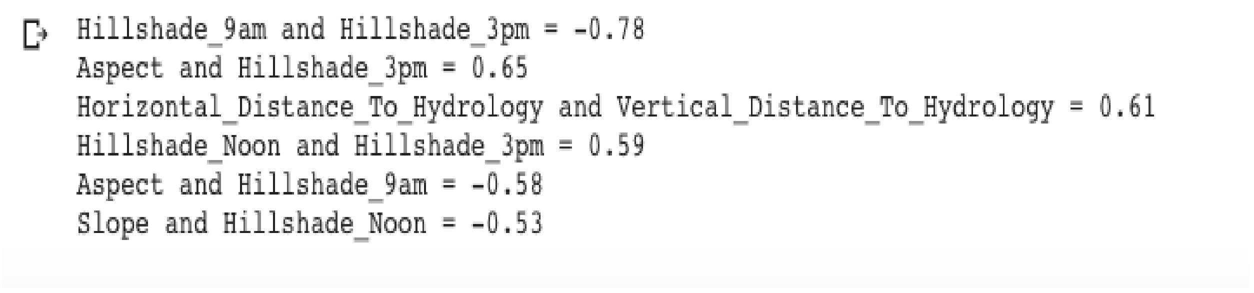


```
for c in dataset.columns:
            if dataset[c].std() == 0:
                rem.append(c)
                print(rem)

            else:
              print('No variable has Sdev == 0 ')
```

```
No variable has Sdev == 0
```

The is no variable in our dataset that have an sdev equal to 0.

3. Get the most correlated variables:

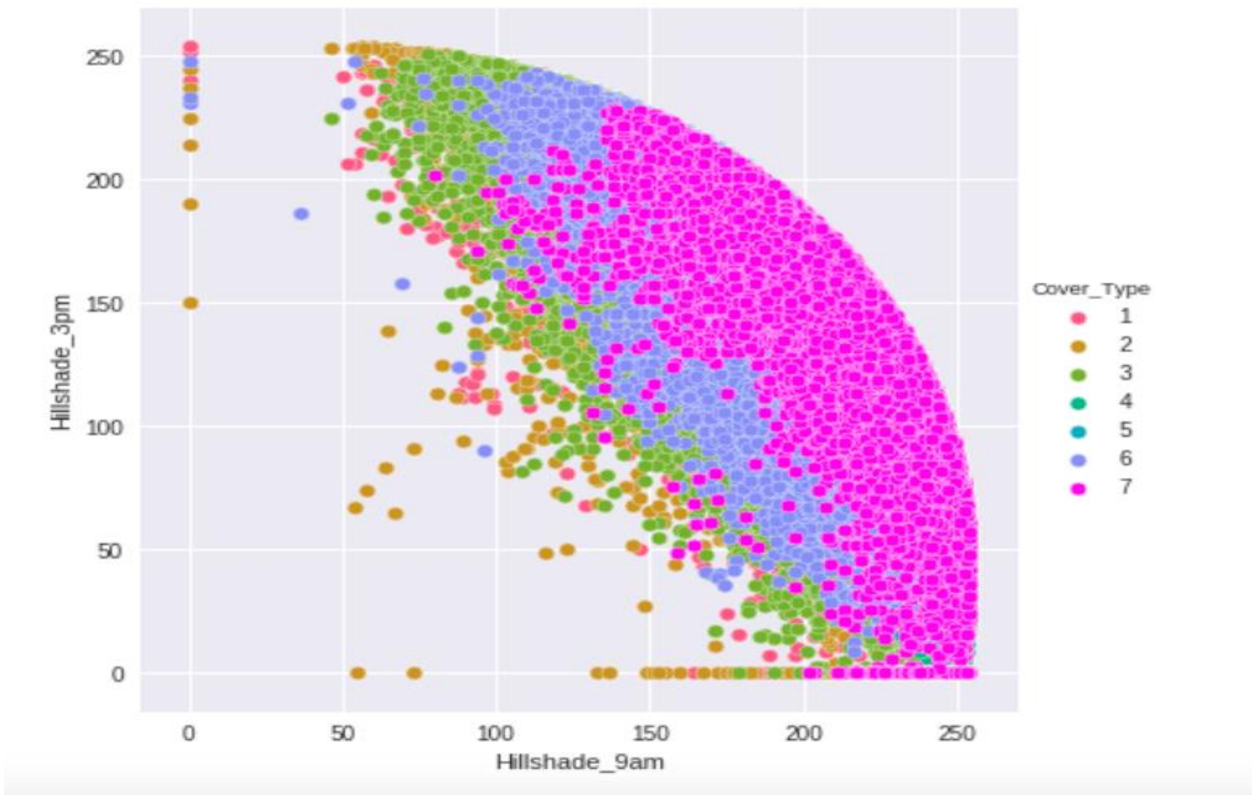


```
Hillshade_9am and Hillshade_3pm = -0.78
Aspect and Hillshade_3pm = 0.65
Horizontal_Distance_To_Hydrology and Vertical_Distance_To_Hydrology = 0.61
Hillshade_Noon and Hillshade_3pm = 0.59
Aspect and Hillshade_9am = -0.58
Slope and Hillshade_Noon = -0.53
```

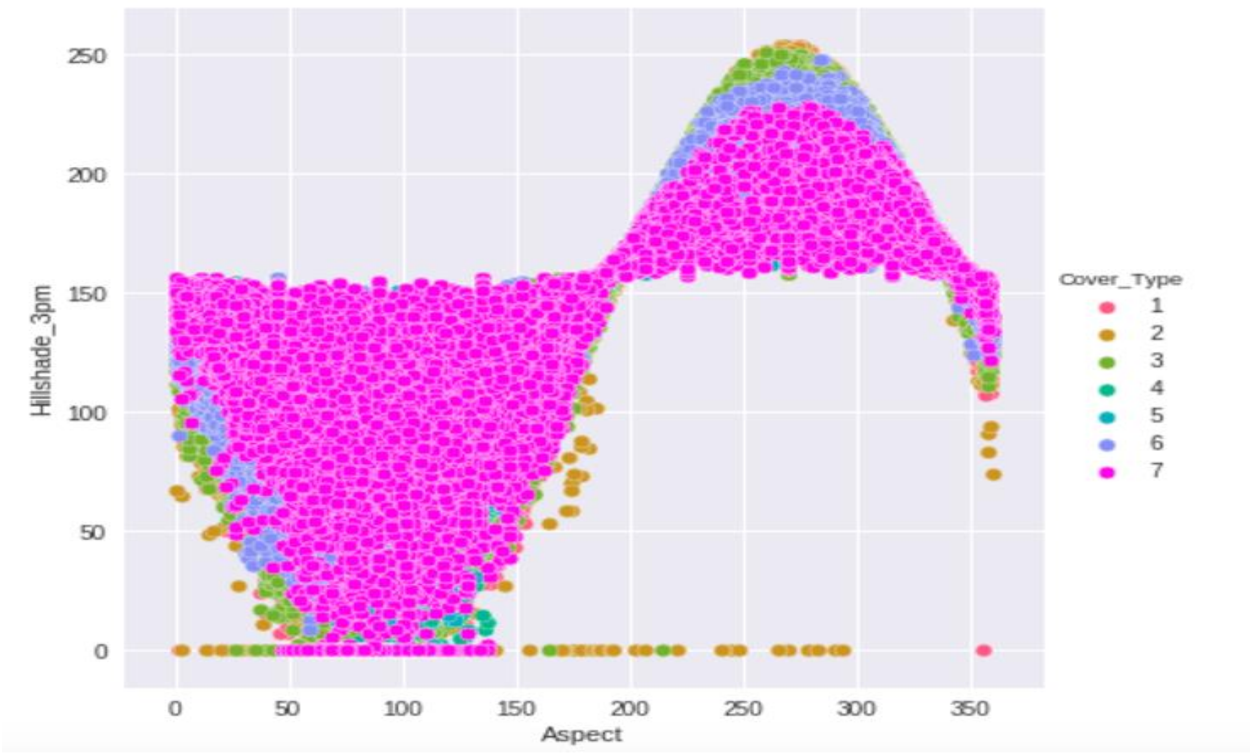
4. Plot the correlation :

Using Matplotlib we obtained 6 plots explaining the correlation between the different correlated variables .

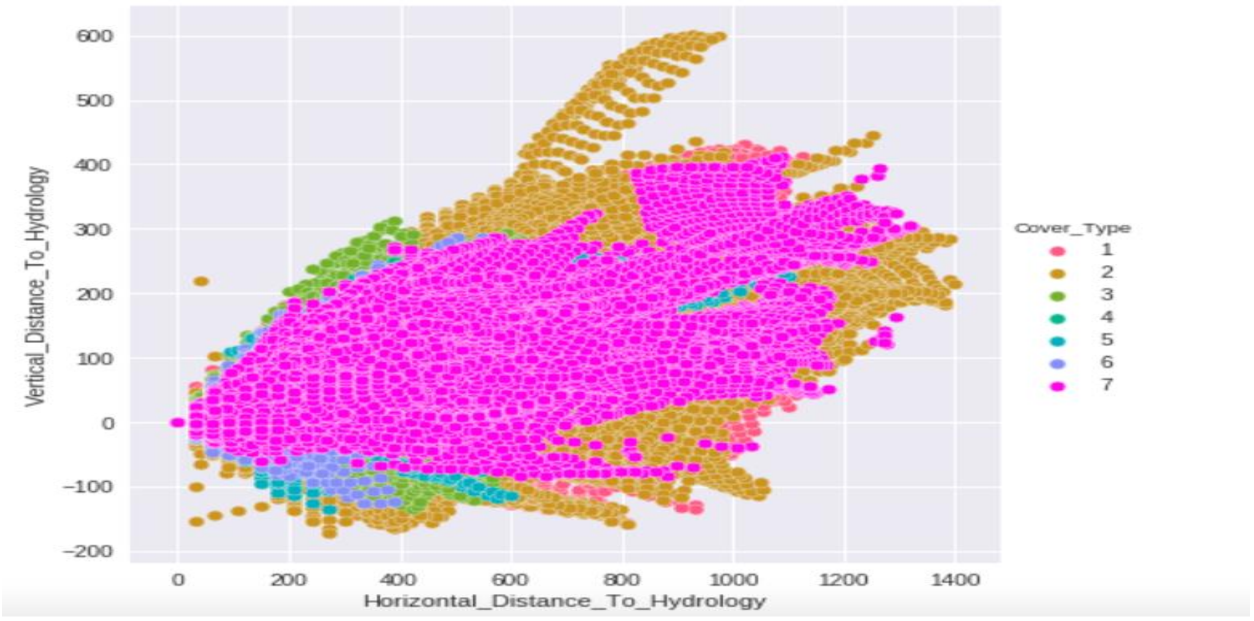
The plots show to which class does a point belong to. The class distribution overlaps in the plots.



Hillshade patterns give a nice ellipsoid patterns with each other



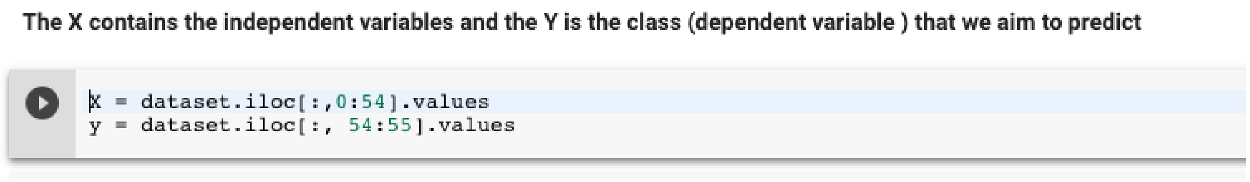
Aspect and Hillshades attributes form a sigmoid pattern



Horizontal and vertical distance to hydrology give an almost linear pattern.

5. **Split the data**:

We first select the dependent variable (the class) and independent variables from our dataset :

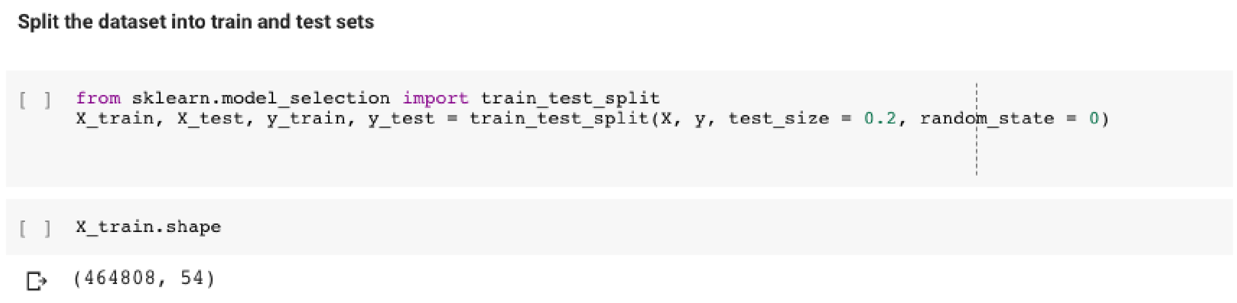

The X contains the independent variables and the Y is the class (dependent variable) that we aim to predict

```
X = dataset.iloc[:,0:54].values
y = dataset.iloc[:, 54:55].values
```

If the model is not tested and is made such that it just perform good on training data then parameters will be such that they are only good enough to predict the value for data which was in training set. That is not general. This is called overfitting.

So we don't land making a useless model which is only good for the training set and not general enough.

Thus test set and training set is important to make Machine Learning model better.


Split the dataset into train and test sets

```
[ ] 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_state = 0)
```

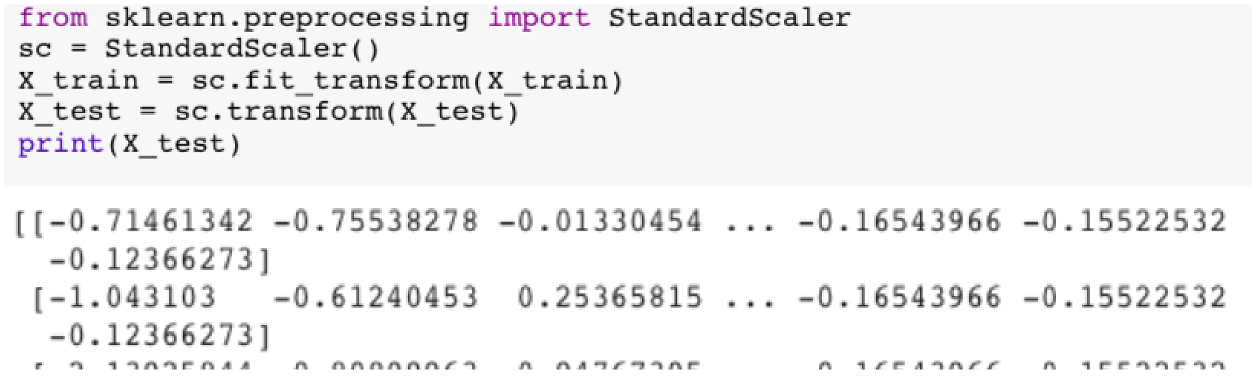
```
[ ] X_train.shape
```

```
(464808, 54)
```

6. Feature Scaling

Feature scaling is a method used to standardize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing step.We also normalize the data because that gradient descent converges much faster with feature scaling than without it.

Python code for feature scaling :



```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_test)
```

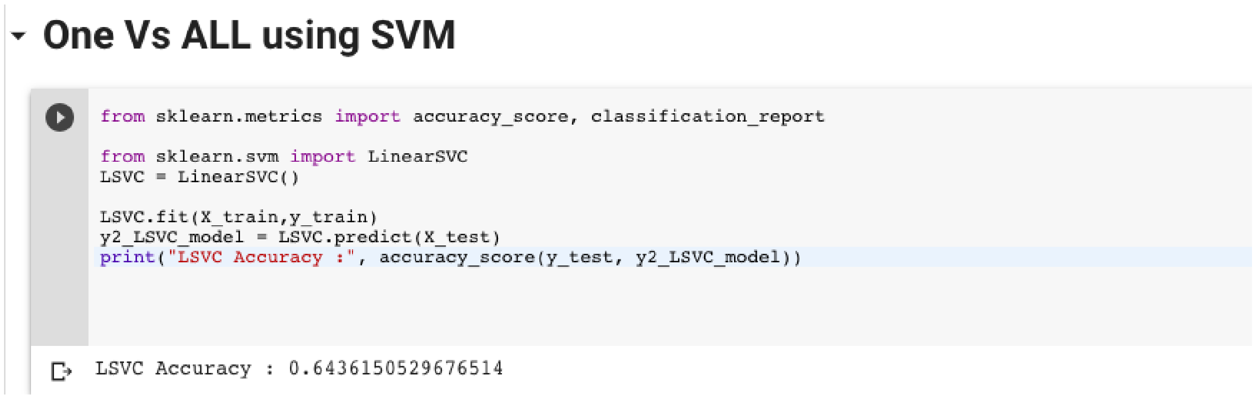
```
[[-0.71461342 -0.75538278 -0.01330454 ... -0.16543966 -0.15522532
  -0.12366273]
 [-1.043103   -0.61240453  0.25365815 ... -0.16543966 -0.15522532
  -0.12366273]
```

I. Machine Learning Algorithms :

In this section we will focus on applying and training our data on some famous machine learning methods and try to compare their performance based on accuracy obtained by the modal and the running time .

1) Train our data on SVM classifier:

I have chosen to work with LinearSVC package of Sickilearn [1] , that works on the strategy of One Vs ALL , how ever , this classifier is linear , unfortunately I didn’t notice until the results of this classifier came out and I tried to add a kernel because the data isn’t linearly separable the second classifier took too much time (more than 3 hours !) , at the end I decided to stop the compiling .



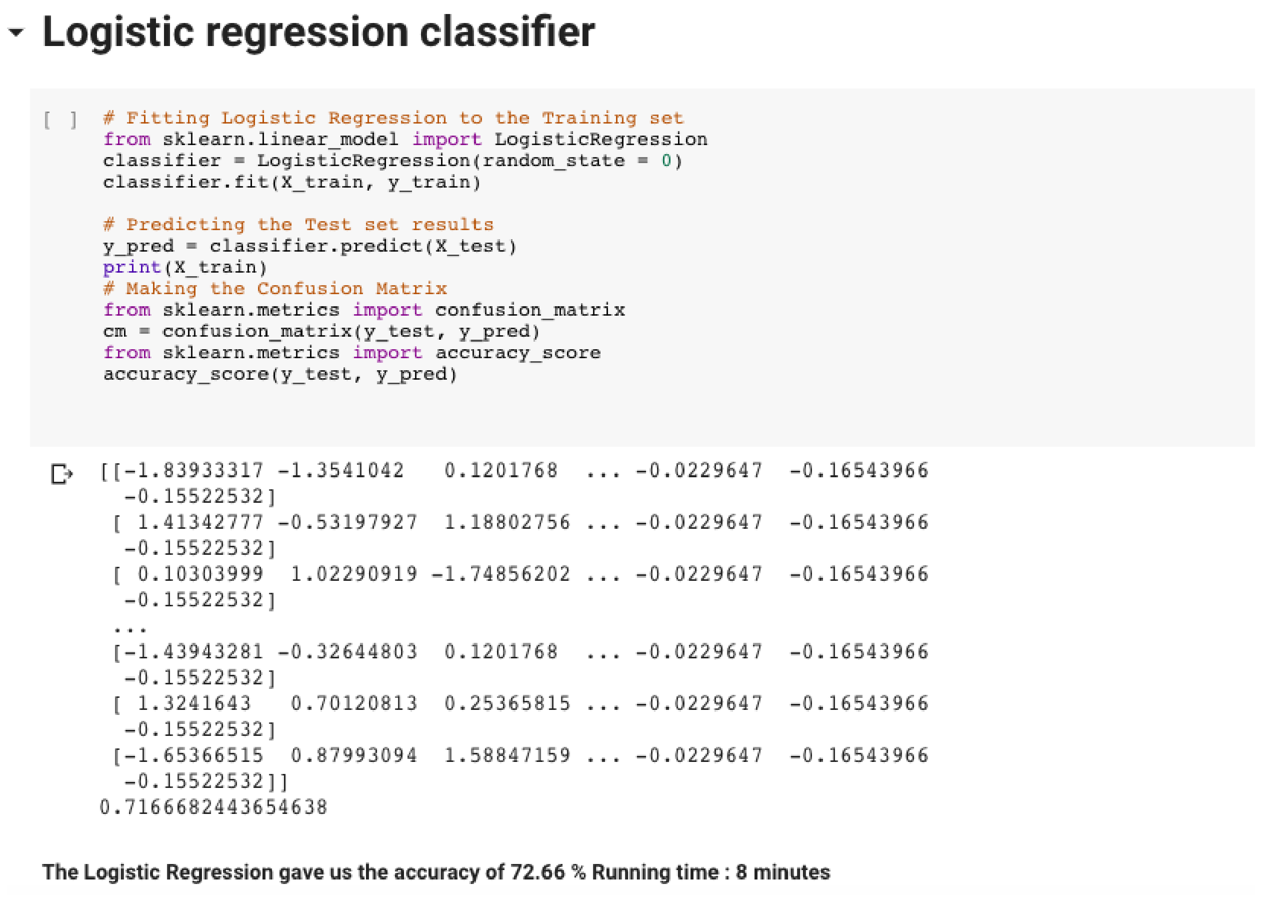
```
from sklearn.metrics import accuracy_score, classification_report

from sklearn.svm import LinearSVC
LSVC = LinearSVC()

LSVC.fit(X_train,y_train)
y2_LSVC_model = LSVC.predict(X_test)
print("LSVC Accuracy :", accuracy_score(y_test, y2_LSVC_model))
```

```
LSVC Accuracy : 0.6436150529676514
```

One Vs ALL SVM gave us the accuracy of 64.3 % !! Execution time: 47 minutes

2) Logistic Regression classifier:

Logistic regression is generally used where the dependent variable is Binary or Dichotomous ,but our data contains 7 classes that we aim to predict . Logistic regression classifier uses the strategy of One Vs ALL , which mean that , it classes one class and consider the other classes as if it is one class , and vice versa with all other classes .



Logistic regression classifier

```
# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(X_train)
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

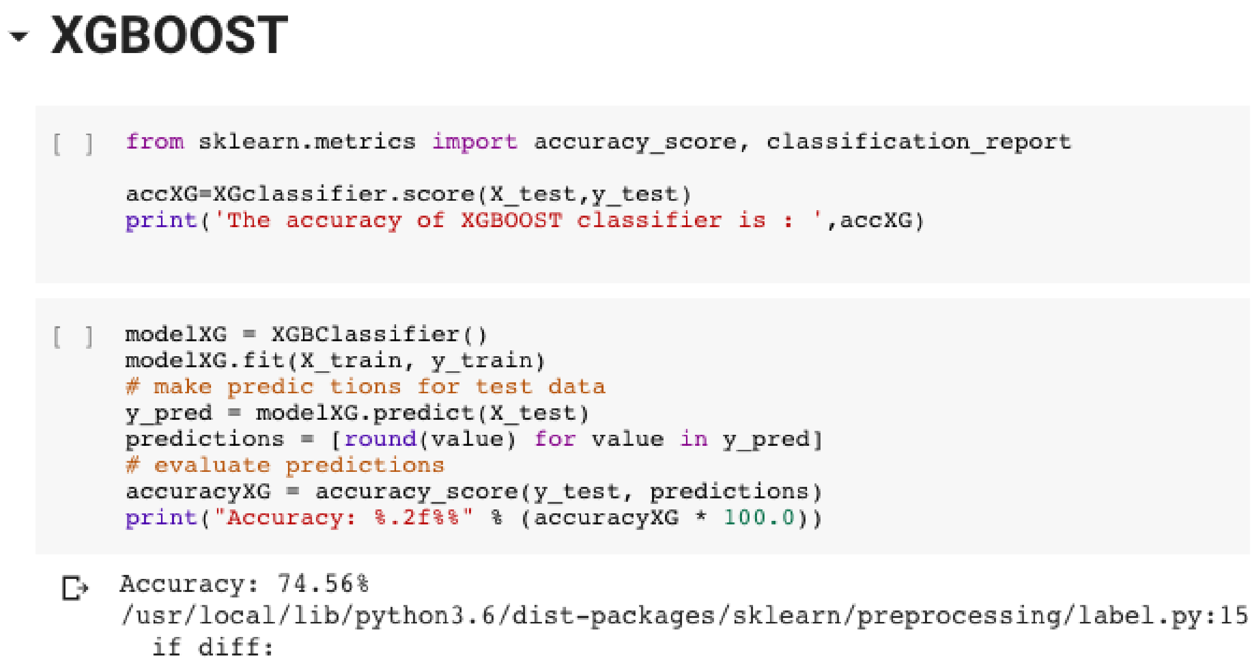
```
[[-1.83933317 -1.3541042   0.1201768  ... -0.0229647  -0.16543966
  -0.15522532]
 [ 1.41342777 -0.53197927  1.18802756 ... -0.0229647  -0.16543966
  -0.15522532]
 [ 0.10303999  1.02290919 -1.74856202 ... -0.0229647  -0.16543966
  -0.15522532]
 ...
 [-1.43943281 -0.32644803  0.1201768  ... -0.0229647  -0.16543966
  -0.15522532]
 [ 1.3241643   0.70120813  0.25365815 ... -0.0229647  -0.16543966
  -0.15522532]
 [-1.65366515  0.87993094  1.58847159 ... -0.0229647  -0.16543966
  -0.15522532]]
0.7166682443654638
```

The Logistic Regression gave us the accuracy of 72.66 % Running time : 8 minutes

3) Ensemble learning

i. BOOSTING:

I have picked **XGBoost** as a Boosting method to train my data with . It is short for eXtreme gradient boosting. It is a library designed and optimized for boosted tree algorithms. Gradient Boosting algorithm is a machine learning technique used for building predictive tree-based models. Boosting is an ensemble technique in which new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made.



XGBOOST

```
[ ]  from sklearn.metrics import accuracy_score, classification_report

     accXG=XGclassifier.score(X_test,y_test)
     print('The accuracy of XGBOOST classifier is : ',accXG)
```

```
[ ]  modelXG = XGBClassifier()
     modelXG.fit(X_train, y_train)
     # make predic tions for test data
     y_pred = modelXG.predict(X_test)
     predictions = [round(value) for value in y_pred]
     # evaluate predictions
     accuracyXG = accuracy_score(y_test, predictions)
     print("Accuracy: %.2f%%" % (accuracyXG * 100.0))
```

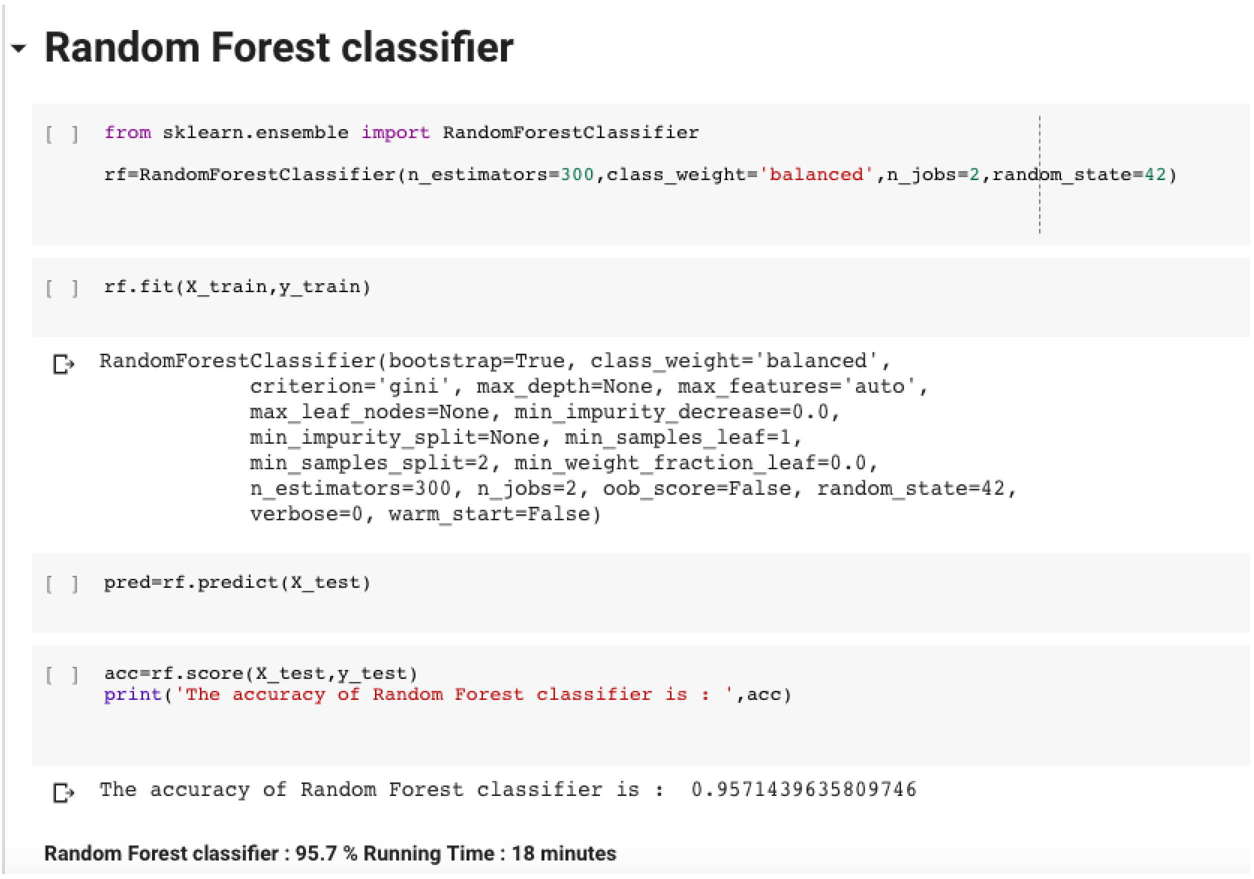
```
Accuracy: 74.56%
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:15
  if diff:
```

XGBOOST Accuracy : 74.56% Execution time : 21 minutes

ii. BAGGING:

Random forests

Random Forest is a supervised learning algorithm , it creates a forest and makes it somehow random. The „forest“ it builds, is an ensemble of Decision Trees, most of the time trained with the “bagging” method.it bases on the concept of that we can make a strong classifier based on ensemble of weak classifiers**.**



Random Forest classifier

```
[ ] from sklearn.ensemble import RandomForestClassifier

    rf=RandomForestClassifier(n_estimators=300,class_weight='balanced',n_jobs=2,random_state=42)
```

```
[ ] rf.fit(X_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=300, n_jobs=2, oob_score=False, random_state=42,
            verbose=0, warm_start=False)
```

```
[ ] pred=rf.predict(X_test)
```

```
[ ] acc=rf.score(X_test,y_test)
    print('The accuracy of Random Forest classifier is : ',acc)
```

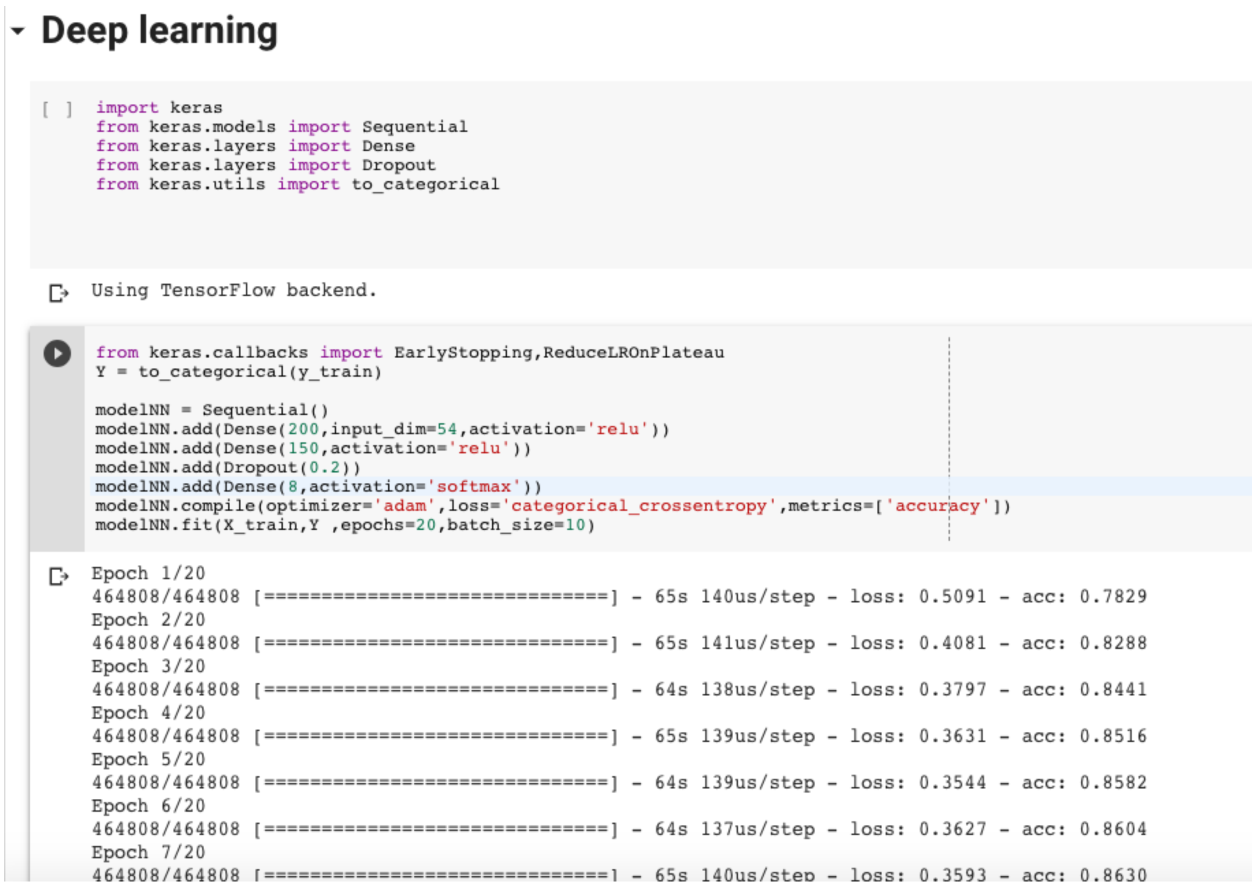
```
The accuracy of Random Forest classifier is :  0.9571439635809746
```

Random Forest classifier : 95.7 % Running Time : 18 minutes

Random forest classifier : 95.7 % on training data
Test Score: 0.84

II. DEEP LEARNING

I have used keras (tensorflow backend) to build the Deep architecture .
Input layer dimension : 54 .
2 hidden layers .
Dropout for regularization in the last layer .
Output layer dimension : 7
20 epochs , Backpropagation each 10 individuals .



Deep learning

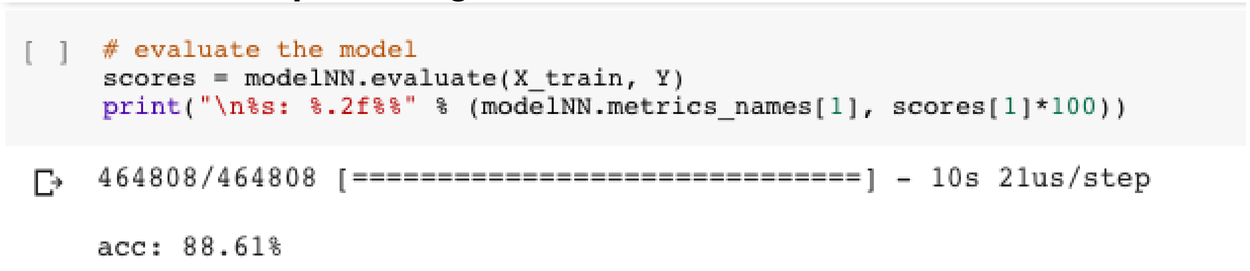
```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.utils import to_categorical
```

```
Using TensorFlow backend.
```

```
from keras.callbacks import EarlyStopping,ReduceLROnPlateau
Y = to_categorical(y_train)

modelNN = Sequential()
modelNN.add(Dense(200,input_dim=54,activation='relu'))
modelNN.add(Dense(150,activation='relu'))
modelNN.add(Dropout(0.2))
modelNN.add(Dense(8,activation='softmax'))
modelNN.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
modelNN.fit(X_train,Y ,epochs=20,batch_size=10)
```

```
Epoch 1/20
464808/464808 [==============================] - 65s 140us/step - loss: 0.5091 - acc: 0.7829
Epoch 2/20
464808/464808 [==============================] - 65s 141us/step - loss: 0.4081 - acc: 0.8288
Epoch 3/20
464808/464808 [==============================] - 64s 138us/step - loss: 0.3797 - acc: 0.8441
Epoch 4/20
464808/464808 [==============================] - 65s 139us/step - loss: 0.3631 - acc: 0.8516
Epoch 5/20
464808/464808 [==============================] - 64s 139us/step - loss: 0.3544 - acc: 0.8582
Epoch 6/20
464808/464808 [==============================] - 64s 137us/step - loss: 0.3627 - acc: 0.8604
Epoch 7/20
464808/464808 [==============================] - 65s 140us/step - loss: 0.3593 - acc: 0.8630
```

Evaluate the deep learning modal :



```
[ ]  # evaluate the model
     scores = modelNN.evaluate(X_train, Y)
     print("\n%s: %.2f%%" % (modelNN.metrics_names[1], scores[1]*100))

     464808/464808 [==============================] - 10s 21us/step

     acc: 88.61%
```

Running time : 21 minutes

Conclusion:

Machine learning and data science is applicable now in many fields.
In this project the random forest gave the best accuracy 95.7 % on training set ,
We are obliged to perform a deep learning modal , sometimes , machine learning algorithm is sufficient .

References:

[1] UCI Machine Learning Repository: Covertype Data Set , https://archive.ics.uci.edu/ml/datasets/covertype , accessed on , 4,12,2019.

[2] https://www.kaggle.com/c/forest-cover-type-prediction
[2] some visualizations from : https://shankarmsy.github.io/posts/forest-cover-types.html