Juliana_Talai_LeavesClassification_Big_Data.

March 17, 2017

1 JULIANA TULA TALAI.

1.1 BIG DATA AND MACHINE LEARNING PROJECT.

2 Leaf Classification

There are estimated to be nearly half a million species of plant in the world. Classification of species has been historically problematic and often results in duplicate identifications. Automating plant recognition might have many applications, including:

- Species population tracking and preservation
- Plant-based medicinal research
- Crop and food supply management

The objective of this work is to use binary leaf images and extracted features, including shape, margin & texture, to accurately identify 99 species of plants. Leaves, due to their volume, prevalence, and unique characteristics, are an effective means of differentiating plant species.

3 Preliminaries

```
In [1]: import matplotlib
        import matplotlib.pylab as pylab
        import matplotlib.pyplot as plt
        import numpy as np
        import pandas as pd
        import sklearn as skl
        import sklearn.preprocessing as pr
        import sklearn.ensemble as en
        from sklearn import linear_model as lm
        from sklearn import model_selection
        from sklearn.metrics import accuracy_score
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.metrics import log_loss
        # This ensures plots are shown in the notebook.
        %matplotlib inline
        # Set default plot size
```

```
pylab.rcParams['figure.figsize'] = 16, 12

# Just to switch off pandas warning
pd.options.mode.chained_assignment = None
```

4 Loading data into jupyter notebook

```
In [2]: raw_data = pd.read_csv("train.csv")
```

5 Data exploration

```
In [3]: n_rows = raw_data.count()[1]     #number of rows and features
        n_features = raw_data.shape[1]

In [4]: n_rows

Out[4]: 990

In [5]: n_features

Out[5]: 194

In [6]: raw_data.shape

Out[6]: (990, 194)
```

The dataset is comprised of 99 unique species each with a sample of 10 subspecies.

```
In [7]: len(set(raw_data.species))

Out[7]: 99

In [8]: raw_data.groupby('species').species.count()

Out[8]: species
        Acer_Capillipes                10
        Acer_Circinatum                10
        Acer_Mono                      10
        Acer_Opalus                    10
        Acer_Palmatum                  10
        Acer_Pictum                    10
        Acer_Platanoids                10
        Acer_Rubrum                    10
        Acer_Rufinerve                 10
        Acer_Saccharinum               10
        Alnus_Cordata                  10
        Alnus_Maximowiczii             10
        Alnus_Rubra                    10
        Alnus_Sieboldiana              10
```

```
Alnus_Viridis                   10
Arundinaria_Simonii             10
Betula_Austrosinensis           10
Betula_Pendula                  10
Callicarpa_Bodinieri            10
Castanea_Sativa                 10
Celtis_Koraiensis               10
Cercis_Siliquastrum             10
Cornus_Chinensis                10
Cornus_Controversa              10
Cornus_Macrophylla              10
Cotinus_Coggygria               10
Crataegus_Monogyna              10
Cytisus_Battandieri             10
Eucalyptus_Glaucescens          10
Eucalyptus_Neglecta             10
                                ..
Quercus_Kewensis                10
Quercus_Nigra                   10
Quercus_Palustris               10
Quercus_Phellos                 10
Quercus_Phillyraeoides          10
Quercus_Pontica                 10
Quercus_Pubescens               10
Quercus_Pyrenaica               10
Quercus_Rhysophylla             10
Quercus_Rubra                   10
Quercus_Semecarpifolia          10
Quercus_Shumardii               10
Quercus_Suber                   10
Quercus_Texana                  10
Quercus_Trojana                 10
Quercus_Variabilis              10
Quercus_Vulcanica               10
Quercus_x_Hispanica             10
Quercus_x_Turneri               10
Rhododendron_x_Russellianum     10
Salix_Fragilis                  10
Salix_Intergra                  10
Sorbus_Aria                     10
Tilia_Oliveri                   10
Tilia_Platyphyllos              10
Tilia_Tomentosa                 10
Ulmus_Bergmanniana              10
Viburnum_Tinus                  10
Viburnum_x_Rhytidophylloides    10
Zelkova_Serrata                 10
Name: species, dtype: int64
```

6 To check for missing data

In [9]:
```
A = []
for i in range(n_features):
    if raw_data.isnull().sum()[i]:
        A.append(raw_data.isnull().sum()[i])
A
```

Out[9]:
```
[]
```

In [10]:
```
raw_data.isnull().sum()
```

Out[10]:
```
id            0
species       0
margin1       0
margin2       0
margin3       0
margin4       0
margin5       0
margin6       0
margin7       0
margin8       0
margin9       0
margin10      0
margin11      0
margin12      0
margin13      0
margin14      0
margin15      0
margin16      0
margin17      0
margin18      0
margin19      0
margin20      0
margin21      0
margin22      0
margin23      0
margin24      0
margin25      0
margin26      0
margin27      0
margin28      0
             ..
texture35     0
texture36     0
texture37     0
texture38     0
texture39     0
texture40     0
```

```
texture41    0
texture42    0
texture43    0
texture44    0
texture45    0
texture46    0
texture47    0
texture48    0
texture49    0
texture50    0
texture51    0
texture52    0
texture53    0
texture54    0
texture55    0
texture56    0
texture57    0
texture58    0
texture59    0
texture60    0
texture61    0
texture62    0
texture63    0
texture64    0
dtype: int64
```

There is no missing data in the dataset

7 Structure of the data

```
In [97]: raw_data.dtypes  #The feature values are floats
```

```
Out[97]: id                  int64
         species            object
         margin1           float64
         margin2           float64
         margin3           float64
         margin4           float64
         margin5           float64
         margin6           float64
         margin7           float64
         margin8           float64
         margin9           float64
         margin10          float64
         margin11          float64
         margin12          float64
         margin13          float64
         margin14          float64
```

```
margin15         float64
margin16         float64
margin17         float64
margin18         float64
margin19         float64
margin20         float64
margin21         float64
margin22         float64
margin23         float64
margin24         float64
margin25         float64
margin26         float64
margin27         float64
margin28         float64
                   ...
texture36        float64
texture37        float64
texture38        float64
texture39        float64
texture40        float64
texture41        float64
texture42        float64
texture43        float64
texture44        float64
texture45        float64
texture46        float64
texture47        float64
texture48        float64
texture49        float64
texture50        float64
texture51        float64
texture52        float64
texture53        float64
texture54        float64
texture55        float64
texture56        float64
texture57        float64
texture58        float64
texture59        float64
texture60        float64
texture61        float64
texture62        float64
texture63        float64
texture64        float64
class_species      int64
dtype: object
```

8 Assign numerical values to string(species) response variable

Label encoder transforms the species labels such that we have values between 0 and n_classes-1, that is, (0 and 98) classes which are our species classes.

```
In [12]: le = pr.LabelEncoder()
         le.fit(raw_data.species)

Out[12]: LabelEncoder()

In [98]: raw_data.loc[:,'class_species'] = le.transform(raw_data.species)#Transforming the speci

In [14]: raw_data.sort_values(by = 'species').head(5)

Out[14]:       id          species   margin1   margin2   margin3   margin4   margin5  \
         111   201  Acer_Capillipes  0.001953  0.000000  0.017578  0.001953  0.054688
         951  1525  Acer_Capillipes  0.000000  0.000000  0.013672  0.015625  0.035156
         370   610  Acer_Capillipes  0.001953  0.001953  0.025391  0.017578  0.029297
         126   227  Acer_Capillipes  0.001953  0.000000  0.017578  0.013672  0.027344
         859  1377  Acer_Capillipes  0.001953  0.000000  0.011719  0.029297  0.033203

               margin6   margin7  margin8       ...         texture56  texture57  \
         111  0.001953  0.019531      0.0       ...               0.0   0.011719
         951  0.000000  0.023438      0.0       ...               0.0   0.008789
         370  0.005859  0.041016      0.0       ...               0.0   0.002930
         126  0.000000  0.009766      0.0       ...               0.0   0.009766
         859  0.000000  0.017578      0.0       ...               0.0   0.005859

              texture58  texture59  texture60  texture61  texture62  texture63  \
         111        0.0   0.019531        0.0        0.0        0.0   0.029297
         951        0.0   0.011719        0.0        0.0        0.0   0.021484
         370        0.0   0.018555        0.0        0.0        0.0   0.036133
         126        0.0   0.019531        0.0        0.0        0.0   0.012695
         859        0.0   0.020508        0.0        0.0        0.0   0.020508

              texture64  class_species
         111   0.025391              0
         951   0.000977              0
         370   0.020508              0
         126   0.000000              0
         859   0.000000              0

         [5 rows x 195 columns]
```

9 Splitting the Data into training and testing data

The data is split into train(60%) and test(40%) with the random number generator used for random sampling as 100

```
In [15]: train_raw, test_raw=model_selection.train_test_split(raw_data,test_size=0.4, random_sta
```

```
In [16]: len(train_raw)
```

```
Out[16]: 594
```

```
In [17]: len(test_raw)
```

```
Out[17]: 396
```

```
In [18]: train_raw.head()
```

```
Out[18]:       id                 species   margin1   margin2   margin3   margin4  \
         616  976          Salix_Fragilis  0.000000  0.000000  0.035156  0.052734
         151  263      Quercus_Imbricaria  0.046875  0.046875  0.021484  0.013672
         341  561  Populus_Grandidentata  0.007812  0.011719  0.126950  0.007812
         396  651          Acer_Rufinerve  0.000000  0.000000  0.015625  0.003906
         271  450       Cornus_Chinensis  0.039062  0.080078  0.019531  0.015625

               margin5   margin6   margin7   margin8      ...        texture56  \
         616  0.083984  0.000000  0.001953  0.000000      ...         0.000000
         151  0.001953  0.080078  0.013672  0.000000      ...         0.008789
         341  0.005859  0.048828  0.007812  0.000000      ...         0.000000
         396  0.041016  0.000000  0.011719  0.000000      ...         0.016602
         271  0.001953  0.070312  0.013672  0.003906      ...         0.000000

              texture57  texture58  texture59  texture60  texture61  texture62  \
         616   0.004883   0.092773   0.045898   0.047852   0.000000   0.077148
         151   0.000000   0.000977   0.011719   0.000000   0.046875   0.036133
         341   0.017578   0.000000   0.002930   0.000000   0.000000   0.011719
         396   0.006836   0.002930   0.020508   0.000000   0.000000   0.022461
         271   0.006836   0.000000   0.041016   0.000000   0.000000   0.006836

              texture63  texture64  class_species
         616   0.000000   0.002930             89
         151   0.003906   0.037109             67
         341   0.000977   0.064453             45
         396   0.002930   0.007812              8
         271   0.024414   0.044922             22

         [5 rows x 195 columns]
```

10 Preparing the data for modelling, we drop the features 'Id' and 'species' as it doesn't carry much information for modelling classification

```
In [19]: t1=train_raw.drop('id',axis=1)     #Train_raw dataset
         t2= test_raw.drop('id',axis=1)      #Test_raw dataset
```

```
t1=t1.drop('species',axis=1)
t2=t2.drop('species',axis=1)
t1=t1.drop('class_species',axis=1)
t2=t2.drop('class_species',axis=1)
```

```
In [20]: ob=list(t1.columns)
```

11 Assigning response and explanatory variables to numpy array

```
In [21]: def choose_columns(data):
             ret_X= np.array(data.loc[:,ob])    #Explanatory variables
             ret_Y= data.class_species.values   #Response variable
             return ret_X, ret_Y
```

```
In [22]: train_X, train_Y=choose_columns(train_raw)
```

```
In [23]: test_X, test_Y=choose_columns(test_raw)
```

12 Multinomial logistic regression (without Normalizing the Data)

```
In [24]: lr1 = lm.LogisticRegressionCV(Cs= 3, fit_intercept=True, cv=5, dual=False, penalty='l2'
                                       scoring=None, solver='lbfgs', tol=0.0001, max_iter=100, class_w
                                       n_jobs=1, verbose=0, refit=True, intercept_scaling=1.0, multi_c
                                       random_state=None)
```

Altering the values of cv and Cs, i realize that the score is better with smaller values of cs and cv=5.Therefore, smaller values bring out stronger regularization.The log loss value also decreases.

```
In [25]: lr1.fit(train_X,train_Y)
```

```
/home/juliana/.local/lib/python3.4/site-packages/sklearn/model_selection/_split.py:581: Warning:
  % (min_groups, self.n_splits)), Warning)
```

```
Out[25]: LogisticRegressionCV(Cs=3, class_weight=None, cv=5, dual=False,
                    fit_intercept=True, intercept_scaling=1.0, max_iter=100,
                    multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
                    refit=True, scoring=None, solver='lbfgs', tol=0.0001, verbose=0)
```

```
In [26]: lr1.score(train_X, train_Y)
```

```
Out[26]: 0.9747474747474747
```

```
In [27]: lr1.score(test_X, test_Y)
```

```
Out[27]: 0.86111111111111116
```

```
In [28]: #This binarizes the class_species to 1 if the species is predicted and 0 if the species
         lb = pr.LabelBinarizer()
         lb.fit(train_raw.class_species)
```

```
Out[28]: LabelBinarizer(neg_label=0, pos_label=1, sparse_output=False)
```

```
In [29]: Y_predicted = lr1.predict(train_X)
```

K and K1 are the label indicator matrices of 1,s if the species is predicted and 0's if the species is not predicted. Pr_Y and Pr_Yt are the predicted probabilities, as returned by the logistic predict_proba method for train_raw and test_raw respectively.

```
In [30]: K =lb.transform(train_raw.class_species)
         K1 =lb.transform(test_raw.class_species)
         Pr_Y = lr1.predict_proba(train_X)
         Pr_Yt = lr1.predict_proba(test_X)
```

```
/usr/lib/python3/dist-packages/scipy/sparse/compressed.py:130: VisibleDeprecationWarning: `rank`
  if np.rank(self.data) != 1 or np.rank(self.indices) != 1 or np.rank(self.indptr) != 1:
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:200: VisibleDeprecationWarning: `rank` is dep
  if np.rank(self.data) != 1 or np.rank(self.row) != 1 or np.rank(self.col) != 1:
```

12.1 Log loss value

```
In [31]: #log loss of train
         log_loss(K, Pr_Y,eps=1e-15, normalize=True, sample_weight=None, labels=None)
```

```
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:182: VisibleDeprecationWarning: `rank` is dep
  if np.rank(M) != 2:
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:200: VisibleDeprecationWarning: `rank` is dep
  if np.rank(self.data) != 1 or np.rank(self.row) != 1 or np.rank(self.col) != 1:
/usr/lib/python3/dist-packages/scipy/sparse/compressed.py:130: VisibleDeprecationWarning: `rank`
  if np.rank(self.data) != 1 or np.rank(self.indices) != 1 or np.rank(self.indptr) != 1:
```

```
Out[31]: 0.15845817583874483
```

```
In [32]: #log loss of test
         log_loss(K1, Pr_Yt,eps=1e-15, normalize=True, sample_weight=None, labels=None)
```

```
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:182: VisibleDeprecationWarning: `rank` is dep
  if np.rank(M) != 2:
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:200: VisibleDeprecationWarning: `rank` is dep
  if np.rank(self.data) != 1 or np.rank(self.row) != 1 or np.rank(self.col) != 1:
/usr/lib/python3/dist-packages/scipy/sparse/compressed.py:130: VisibleDeprecationWarning: `rank`
  if np.rank(self.data) != 1 or np.rank(self.indices) != 1 or np.rank(self.indptr) != 1:
```

```
Out[32]: 0.69510681232960647
```

13 K NearestNeighbours (Not normalized data)

Adjusting the value of n_neighbors,it is realized that the best score is achieved when the n_neighbors is 7.The logloss function is quite large,so we normalized the data.

```
In [33]: Nn = KNeighborsClassifier(n_neighbors = 7)
         Nn.fit(train_X,train_Y)

Out[33]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=7, p=2,
                    weights='uniform')

In [34]: Nn.score(train_X,train_Y)

Out[34]: 0.84680134680134678

In [35]: Nn.score(test_X,test_Y)

Out[35]: 0.72727272727272729

In [36]: Prn_Y = Nn.predict_proba(train_X)
         Prn_Yt =Nn.predict_proba(test_X)
```

13.1 Logloss value

```
In [37]: ##log loss of train on knn
         log_loss(K,Prn_Y ,eps=1e-15, normalize=True, sample_weight=None, labels=None)

/usr/lib/python3/dist-packages/scipy/sparse/coo.py:182: VisibleDeprecationWarning: `rank` is dep
  if np.rank(M) != 2:
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:200: VisibleDeprecationWarning: `rank` is dep
  if np.rank(self.data) != 1 or np.rank(self.row) != 1 or np.rank(self.col) != 1:
/usr/lib/python3/dist-packages/scipy/sparse/compressed.py:130: VisibleDeprecationWarning: `rank`
  if np.rank(self.data) != 1 or np.rank(self.indices) != 1 or np.rank(self.indptr) != 1:


Out[37]: 0.5878013793343011

In [38]: #log loss of test
         log_loss(K1,Prn_Yt ,eps=1e-15, normalize=True, sample_weight=None, labels=None)

/usr/lib/python3/dist-packages/scipy/sparse/coo.py:182: VisibleDeprecationWarning: `rank` is dep
  if np.rank(M) != 2:
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:200: VisibleDeprecationWarning: `rank` is dep
  if np.rank(self.data) != 1 or np.rank(self.row) != 1 or np.rank(self.col) != 1:
/usr/lib/python3/dist-packages/scipy/sparse/compressed.py:130: VisibleDeprecationWarning: `rank`
  if np.rank(self.data) != 1 or np.rank(self.indices) != 1 or np.rank(self.indptr) != 1:


Out[38]: 2.5558525593293444
```

14 Random Forest (Non Normalized data)

The best score is found when n_estimators=120 but the logloss value is greater than 1.We therefore normalized the data.

```
In [39]: Rf = en.RandomForestClassifier(n_estimators= 120, criterion='gini', max_depth=None,
                                        min_samples_split=2, min_samples_leaf=1, min_weight_fraction_
                                        max_features='auto', max_leaf_nodes=None, min_impurity_split=
                                        bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
                                        warm_start=False, class_weight=None)
```

```
In [40]: Rf.fit(train_X,train_Y)
```

```
Out[40]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_impurity_split=1e-07, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     n_estimators=120, n_jobs=1, oob_score=False, random_state=None,
                     verbose=0, warm_start=False)
```

```
In [41]: Rf.score(train_X,train_Y)
```

```
Out[41]: 1.0
```

```
In [42]: Rf.score(test_X,test_Y)
```

```
Out[42]: 0.93686868686868685
```

```
In [43]: Pro_Y = Rf.predict_proba(train_X)
         Pro_Yt = Rf.predict_proba(test_X)
```

14.1 Logloss value.

```
In [44]: #log loss of train in random forest
         log_loss(K,Pro_Y ,eps=1e-15, normalize=True, sample_weight=None, labels=None)
```

```
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:182: VisibleDeprecationWarning: `rank` is dep
  if np.rank(M) != 2:
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:200: VisibleDeprecationWarning: `rank` is dep
  if np.rank(self.data) != 1 or np.rank(self.row) != 1 or np.rank(self.col) != 1:
/usr/lib/python3/dist-packages/scipy/sparse/compressed.py:130: VisibleDeprecationWarning: `rank`
  if np.rank(self.data) != 1 or np.rank(self.indices) != 1 or np.rank(self.indptr) != 1:
```

```
Out[44]: 0.25849691062348223
```

```
In [45]: #log loss of test in random forest
         log_loss(K1,Pro_Yt ,eps=1e-15, normalize=True, sample_weight=None, labels=None)
```

```
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:182: VisibleDeprecationWarning: `rank` is dep
  if np.rank(M) != 2:
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:200: VisibleDeprecationWarning: `rank` is dep
  if np.rank(self.data) != 1 or np.rank(self.row) != 1 or np.rank(self.col) != 1:
/usr/lib/python3/dist-packages/scipy/sparse/compressed.py:130: VisibleDeprecationWarning: `rank`
  if np.rank(self.data) != 1 or np.rank(self.indices) != 1 or np.rank(self.indptr) != 1:
```

```
Out[45]: 1.0715188404352942
```

15 - Normalized Data -

```
In [46]: train_norm = skl.preprocessing.scale(t1) #t1 and t2 are our data frames where we droppe
         test_norm = skl.preprocessing.scale(t2)
```

```
In [47]: train_X1 = train_norm
         #Pro_Yt = Rf.predict_proba(test_X)
         train_Y1 = np.array(train_raw.class_species)
         test_X1 = test_norm
```

16 Multinomial Logistic Regression

Varying the values of Cs in the range (1e-4 and 1e4),the best model is found when Cs is 1e1 and cv=5.The model score improves with standardization.The logloss value also decreases.

```
In [48]: lr = lm.LogisticRegressionCV(Cs=[1e1], fit_intercept=True, cv=5, dual=False, penalty='l
                                      scoring=None, solver='lbfgs', tol=0.0001, max_iter=100, class_w
                                      n_jobs=1, verbose=0, refit=True, intercept_scaling=1.0, multi_c
                                      random_state=None)
```

```
In [49]: lr.fit(train_X1,train_Y1)
```

```
/home/juliana/.local/lib/python3.4/site-packages/sklearn/model_selection/_split.py:581: Warning:
  % (min_groups, self.n_splits)), Warning)
```

```
Out[49]: LogisticRegressionCV(Cs=[10.0], class_weight=None, cv=5, dual=False,
                   fit_intercept=True, intercept_scaling=1.0, max_iter=100,
                   multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
                   refit=True, scoring=None, solver='lbfgs', tol=0.0001, verbose=0)
```

```
In [50]: lr.score(train_X1, train_Y1)
```

```
Out[50]: 1.0
```

```
In [51]: lr.score(test_X1, test_Y)
```

```
Out[51]: 0.96717171717171713
```

```
In [52]: Pr1_Y = lr.predict_proba(train_X1)
         Pr1_Yt = lr.predict_proba(test_X1)
```

16.1 Logloss value.

```
In [53]: #log loss of train normalized data
         log_loss(K,Pr1_Y ,eps=1e-15, normalize=True, sample_weight=None, labels=None)

/usr/lib/python3/dist-packages/scipy/sparse/coo.py:182: VisibleDeprecationWarning: `rank` is dep
  if np.rank(M) != 2:
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:200: VisibleDeprecationWarning: `rank` is dep
  if np.rank(self.data) != 1 or np.rank(self.row) != 1 or np.rank(self.col) != 1:
/usr/lib/python3/dist-packages/scipy/sparse/compressed.py:130: VisibleDeprecationWarning: `rank`
  if np.rank(self.data) != 1 or np.rank(self.indices) != 1 or np.rank(self.indptr) != 1:


Out[53]: 0.0083327547554483232
```

```
In [54]: #log loss of test normalized data
         log_loss(K1,Pr1_Yt ,eps=1e-15, normalize=True, sample_weight=None, labels=None)

/usr/lib/python3/dist-packages/scipy/sparse/coo.py:182: VisibleDeprecationWarning: `rank` is dep
  if np.rank(M) != 2:
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:200: VisibleDeprecationWarning: `rank` is dep
  if np.rank(self.data) != 1 or np.rank(self.row) != 1 or np.rank(self.col) != 1:
/usr/lib/python3/dist-packages/scipy/sparse/compressed.py:130: VisibleDeprecationWarning: `rank`
  if np.rank(self.data) != 1 or np.rank(self.indices) != 1 or np.rank(self.indptr) != 1:


Out[54]: 0.15444530907544521
```

17 K NearestNeighbours

Normalizing the data and fitting model when n_neighbors=7,improves the scores and decreases the logloss value.

```
In [55]: Nn.fit(train_X1,train_Y1)

Out[55]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=7, p=2,
                    weights='uniform')

In [56]: Nn.score(train_X1,train_Y1)

Out[56]: 0.92929292929292928

In [57]: Nn.score(test_X1,test_Y)

Out[57]: 0.84090909090909094

In [58]: Prn1_Y = Nn.predict_proba(train_X1)
         Prn1_Yt =Nn.predict_proba(test_X1)
```

17.1 Logloss value.

```
In [59]: #log loss of train normalized data
         log_loss(K,Prn1_Y ,eps=1e-10, normalize=True, sample_weight=None, labels=None)
```

```
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:182: VisibleDeprecationWarning: `rank` is dep
  if np.rank(M) != 2:
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:200: VisibleDeprecationWarning: `rank` is dep
  if np.rank(self.data) != 1 or np.rank(self.row) != 1 or np.rank(self.col) != 1:
/usr/lib/python3/dist-packages/scipy/sparse/compressed.py:130: VisibleDeprecationWarning: `rank`
  if np.rank(self.data) != 1 or np.rank(self.indices) != 1 or np.rank(self.indptr) != 1:
```

```
Out[59]: 0.32522168275726843
```

```
In [60]: #log loss of test normalized data
         log_loss(K1,Prn1_Yt ,eps=1e-10, normalize=True, sample_weight=None, labels=None)
```

```
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:182: VisibleDeprecationWarning: `rank` is dep
  if np.rank(M) != 2:
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:200: VisibleDeprecationWarning: `rank` is dep
  if np.rank(self.data) != 1 or np.rank(self.row) != 1 or np.rank(self.col) != 1:
/usr/lib/python3/dist-packages/scipy/sparse/compressed.py:130: VisibleDeprecationWarning: `rank`
  if np.rank(self.data) != 1 or np.rank(self.indices) != 1 or np.rank(self.indptr) != 1:
```

```
Out[60]: 0.72054540366619058
```

18 Random Forest

Normalizing the data and using this classifier does not really improve the model and the logloss values.

```
In [61]: Rf.fit(train_X1,train_Y1)
```

```
Out[61]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_impurity_split=1e-07, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     n_estimators=120, n_jobs=1, oob_score=False, random_state=None,
                     verbose=0, warm_start=False)
```

```
In [62]: Rf.score(train_X1,train_Y1)
```

```
Out[62]: 1.0
```

```
In [63]: Rf.score(test_X1,test_Y)
```

```
Out[63]: 0.90404040404040409
```

```
In [64]: test_pred = Rf.predict(test_X1)
```

```
In [65]: Pro1_Y = Rf.predict_proba(train_X1)
         Pro1_Yt = Rf.predict_proba(test_X1)
```

18.1 Logloss value.

```
In [66]: #log loss of train normalized data
         log_loss(K,Pro1_Y ,eps=1e-10, normalize=True, sample_weight=None, labels=None)

/usr/lib/python3/dist-packages/scipy/sparse/coo.py:182: VisibleDeprecationWarning: `rank` is dep
  if np.rank(M) != 2:
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:200: VisibleDeprecationWarning: `rank` is dep
  if np.rank(self.data) != 1 or np.rank(self.row) != 1 or np.rank(self.col) != 1:
/usr/lib/python3/dist-packages/scipy/sparse/compressed.py:130: VisibleDeprecationWarning: `rank`
  if np.rank(self.data) != 1 or np.rank(self.indices) != 1 or np.rank(self.indptr) != 1:


Out[66]: 0.25621457838401568
```

```
In [67]: #log loss of test normalized data
         log_loss(K1,Pro1_Yt ,eps=1e-10, normalize=True, sample_weight=None, labels=None)

/usr/lib/python3/dist-packages/scipy/sparse/coo.py:182: VisibleDeprecationWarning: `rank` is dep
  if np.rank(M) != 2:
/usr/lib/python3/dist-packages/scipy/sparse/coo.py:200: VisibleDeprecationWarning: `rank` is dep
  if np.rank(self.data) != 1 or np.rank(self.row) != 1 or np.rank(self.col) != 1:
/usr/lib/python3/dist-packages/scipy/sparse/compressed.py:130: VisibleDeprecationWarning: `rank`
  if np.rank(self.data) != 1 or np.rank(self.indices) != 1 or np.rank(self.indptr) != 1:


Out[67]: 1.1106418156036588
```

19 Checking the Accuracy, Recall and Precision

```
In [68]: import sklearn.metrics as m
```

20 Logistic

```
In [69]: test_predicted = lr.predict(test_X1)
```

```
In [70]: def score_test_set(test_pred, test_Y):
             print("accuracy:", m.accuracy_score(test_Y, test_pred))
             print("precision:", m.precision_score(test_Y, test_pred, average = 'weighted'))
             print("recall:", m.recall_score(test_Y, test_pred, average = 'weighted'))
         score_test_set(test_predicted, test_Y)

accuracy: 0.967171717172
precision: 0.979882154882
recall: 0.967171717172
```

21 KNearest Neighbors

```
In [71]: test_predicted1 = Nn.predict(test_X1)

In [72]: score_test_set(test_predicted1, test_Y)
```

```
accuracy: 0.840909090909
precision: 0.880808080808
recall: 0.840909090909


/home/juliana/.local/lib/python3.4/site-packages/sklearn/metrics/classification.py:1113: Undefin
  'precision', 'predicted', average, warn_for)
```

22 Random forest

```
In [73]: test_predicted2 = Rf.predict(test_X1)

In [74]: score_test_set(test_predicted2, test_Y)
```

```
accuracy: 0.90404040404
precision: 0.938251563252
recall: 0.90404040404
```

Accuracy:We are 96.7%,84% and 91.4% accurate respectively that the set of predicted labels for the sample matches the corresponding set of labels in Y_true.

Precision:We have 97.99%,88%,93% precision rate respectively meaning that the we have predicted 97.99%,88%,93% of the data as true positives.

Recall: The classifiers are returning low false negative values.96.7%,84.1%,90.4% of the data is therefore positively predicted.

23 Kaggle submission

```
In [95]: test2 = pd.read_csv("test.csv") #Reading the test data from kaggle

In [76]: t3 = test2.drop('id',axis=1)

In [96]: test_norm1 = skl.preprocessing.scale(t3) #Normalizing the data
```

24 For logistic

```
In [78]: sp=lr.predict(test_norm1)

In [79]: Probabilities = lr.predict_proba(test_norm1)

In [92]: species = le.inverse_transform(sp) #Transforming numerical labels to non-numerical labe
         species
```

```
Out[92]: array(['Quercus_Agrifolia', 'Quercus_Afares', 'Acer_Circinatum',
       'Castanea_Sativa', 'Alnus_Viridis', 'Acer_Opalus', 'Acer_Opalus',
       'Eucalyptus_Glaucescens', 'Quercus_Variabilis', 'Acer_Rufinerve',
       'Phildelphus', 'Quercus_Pontica', 'Quercus_Pubescens',
       'Alnus_Cordata', 'Quercus_Alnifolia', 'Populus_Nigra',
       'Populus_Grandidentata', 'Quercus_Phillyraeoides',
       'Alnus_Sieboldiana', 'Quercus_Palustris', 'Quercus_Crassipes',
       'Quercus_Infectoria_sub', 'Quercus_Chrysolepis',
       'Quercus_Rhysophylla', 'Acer_Circinatum', 'Quercus_Nigra',
       'Eucalyptus_Glaucescens', 'Arundinaria_Simonii',
       'Liquidambar_Styraciflua', 'Quercus_Nigra', 'Quercus_Brantii',
       'Quercus_Pontica', 'Prunus_Avium', 'Quercus_Afares',
       'Acer_Palmatum', 'Liriodendron_Tulipifera', 'Alnus_Viridis',
       'Quercus_Castaneifolia', 'Liriodendron_Tulipifera',
       'Tilia_Platyphyllos', 'Acer_Rufinerve', 'Ginkgo_Biloba',
       'Acer_Rufinerve', 'Acer_Saccharinum', 'Quercus_Palustris',
       'Quercus_Nigra', 'Lithocarpus_Edulis', 'Cornus_Controversa',
       'Tilia_Tomentosa', 'Callicarpa_Bodinieri', 'Betula_Pendula',
       'Acer_Pictum', 'Quercus_Castaneifolia', 'Tilia_Tomentosa',
       'Alnus_Viridis', 'Quercus_x_Hispanica', 'Quercus_Dolicholepis',
       'Ilex_Aquifolium', 'Quercus_Agrifolia', 'Zelkova_Serrata',
       'Rhododendron_x_Russellianum', 'Quercus_Cerris',
       'Cercis_Siliquastrum', 'Quercus_Coccinea', 'Quercus_Hartwissiana',
       'Alnus_Maximowiczii', 'Prunus_X_Shmittii', 'Acer_Pictum',
       'Alnus_Sieboldiana', 'Acer_Palmatum', 'Quercus_Canariensis',
       'Quercus_Chrysolepis', 'Eucalyptus_Neglecta', 'Acer_Rubrum',
       'Fagus_Sylvatica', 'Zelkova_Serrata', 'Tilia_Oliveri',
       'Quercus_Variabilis', 'Cotinus_Coggygria', 'Alnus_Cordata',
       'Quercus_Crassipes', 'Phildelphus', 'Quercus_Vulcanica',
       'Cornus_Macrophylla', 'Acer_Circinatum', 'Acer_Mono',
       'Viburnum_Tinus', 'Quercus_Trojana', 'Magnolia_Salicifolia',
       'Cornus_Chinensis', 'Prunus_X_Shmittii', 'Salix_Intergra',
       'Cotinus_Coggygria', 'Cercis_Siliquastrum',
       'Lithocarpus_Cleistocarpus', 'Quercus_Cerris', 'Morus_Nigra',
       'Ulmus_Bergmanniana', 'Acer_Rubrum', 'Salix_Fragilis',
       'Zelkova_Serrata', 'Quercus_Rhysophylla', 'Acer_Opalus',
       'Alnus_Rubra', 'Fagus_Sylvatica', 'Quercus_Variabilis',
       'Quercus_Brantii', 'Viburnum_Tinus', 'Quercus_Greggii',
       'Quercus_Phellos', 'Tilia_Platyphyllos', 'Tilia_Platyphyllos',
       'Quercus_Imbricaria', 'Eucalyptus_Urnigera', 'Acer_Rufinerve',
       'Rhododendron_x_Russellianum', 'Acer_Opalus', 'Quercus_x_Turneri',
       'Acer_Platanoids', 'Quercus_Chrysolepis', 'Ilex_Cornuta',
       'Salix_Intergra', 'Quercus_Crassifolia', 'Betula_Pendula',
       'Quercus_Pubescens', 'Cytisus_Battandieri', 'Quercus_Agrifolia',
       'Acer_Rubrum', 'Magnolia_Heptapeta', 'Cornus_Controversa',
       'Cornus_Macrophylla', 'Acer_Mono', 'Morus_Nigra',
       'Quercus_Crassipes', 'Cornus_Macrophylla',
       'Viburnum_x_Rhytidophylloides', 'Eucalyptus_Neglecta',
```

```
'Eucalyptus_Glaucescens', 'Quercus_Infectoria_sub', 'Quercus_Suber',
'Olea_Europaea', 'Quercus_Agrifolia', 'Quercus_x_Hispanica',
'Quercus_Dolicholepis', 'Quercus_Crassifolia', 'Quercus_Alnifolia',
'Ulmus_Bergmanniana', 'Quercus_Greggii', 'Olea_Europaea',
'Viburnum_Tinus', 'Ulmus_Bergmanniana', 'Celtis_Koraiensis',
'Quercus_Coccinea', 'Liquidambar_Styraciflua',
'Quercus_x_Hispanica', 'Acer_Circinatum', 'Crataegus_Monogyna',
'Lithocarpus_Edulis', 'Phildelphus', 'Quercus_Pubescens',
'Celtis_Koraiensis', 'Quercus_Dolicholepis', 'Populus_Nigra',
'Quercus_Semecarpifolia', 'Cornus_Chinensis',
'Quercus_Semecarpifolia', 'Quercus_Kewensis', 'Quercus_x_Turneri',
'Quercus_Hartwissiana', 'Viburnum_x_Rhytidophylloides',
'Quercus_Pubescens', 'Cercis_Siliquastrum', 'Eucalyptus_Neglecta',
'Cercis_Siliquastrum', 'Alnus_Maximowiczii', 'Alnus_Cordata',
'Quercus_Coccifera', 'Tilia_Tomentosa', 'Cytisus_Battandieri',
'Cornus_Chinensis', 'Arundinaria_Simonii', 'Populus_Grandidentata',
'Salix_Fragilis', 'Quercus_Canariensis', 'Phildelphus',
'Acer_Pictum', 'Cornus_Controversa', 'Tilia_Tomentosa',
'Magnolia_Salicifolia', 'Pterocarya_Stenoptera', 'Salix_Fragilis',
'Quercus_Phellos', 'Liriodendron_Tulipifera', 'Tilia_Platyphyllos',
'Quercus_Suber', 'Ulmus_Bergmanniana', 'Callicarpa_Bodinieri',
'Sorbus_Aria', 'Eucalyptus_Neglecta', 'Quercus_Greggii',
'Quercus_Shumardii', 'Acer_Platanoids', 'Quercus_Rubra',
'Populus_Grandidentata', 'Eucalyptus_Glaucescens',
'Acer_Saccharinum', 'Quercus_Rubra', 'Salix_Intergra',
'Populus_Adenopoda', 'Salix_Fragilis', 'Tilia_Oliveri',
'Alnus_Sieboldiana', 'Acer_Mono', 'Quercus_Coccinea',
'Acer_Capillipes', 'Viburnum_Tinus', 'Quercus_Nigra',
'Ginkgo_Biloba', 'Eucalyptus_Neglecta', 'Quercus_Rubra',
'Sorbus_Aria', 'Quercus_Castaneifolia', 'Populus_Adenopoda',
'Lithocarpus_Edulis', 'Acer_Pictum', 'Quercus_Dolicholepis',
'Pterocarya_Stenoptera', 'Quercus_Coccifera', 'Tilia_Tomentosa',
'Quercus_Imbricaria', 'Tilia_Oliveri',
'Rhododendron_x_Russellianum', 'Salix_Intergra',
'Quercus_Semecarpifolia', 'Cotinus_Coggygria',
'Lithocarpus_Cleistocarpus', 'Callicarpa_Bodinieri',
'Acer_Capillipes', 'Quercus_Pyrenaica', 'Cytisus_Battandieri',
'Quercus_Nigra', 'Quercus_Palustris', 'Populus_Adenopoda',
'Quercus_Nigra', 'Ginkgo_Biloba', 'Salix_Intergra',
'Eucalyptus_Urnigera', 'Quercus_Texana', 'Ilex_Cornuta',
'Eucalyptus_Urnigera', 'Quercus_Texana', 'Viburnum_Tinus',
'Prunus_X_Shmittii', 'Quercus_Hartwissiana', 'Quercus_Chrysolepis',
'Quercus_Greggii', 'Crataegus_Monogyna', 'Quercus_Brantii',
'Quercus_Castaneifolia', 'Quercus_Phillyraeoides',
'Acer_Saccharinum', 'Acer_Opalus', 'Quercus_Trojana',
'Crataegus_Monogyna', 'Eucalyptus_Urnigera',
'Quercus_Ellipsoidalis', 'Betula_Pendula', 'Cornus_Chinensis',
'Quercus_Semecarpifolia', 'Quercus_Ellipsoidalis', 'Alnus_Rubra',
```

```
'Quercus_Rubra', 'Liriodendron_Tulipifera', 'Alnus_Viridis',
'Cytisus_Battandieri', 'Cotinus_Coggygria', 'Quercus_Imbricaria',
'Lithocarpus_Edulis', 'Celtis_Koraiensis', 'Quercus_Canariensis',
'Quercus_Pyrenaica', 'Quercus_Kewensis', 'Quercus_Imbricaria',
'Viburnum_x_Rhytidophylloides', 'Quercus_Shumardii',
'Populus_Adenopoda', 'Tilia_Oliveri', 'Quercus_Kewensis',
'Cornus_Controversa', 'Cercis_Siliquastrum',
'Betula_Austrosinensis', 'Quercus_Agrifolia', 'Ilex_Aquifolium',
'Quercus_Rhysophylla', 'Betula_Austrosinensis',
'Alnus_Maximowiczii', 'Viburnum_x_Rhytidophylloides',
'Acer_Circinatum', 'Quercus_Alnifolia', 'Magnolia_Heptapeta',
'Cornus_Macrophylla', 'Quercus_Alnifolia', 'Olea_Europaea',
'Quercus_Alnifolia', 'Acer_Mono', 'Quercus_Phillyraeoides',
'Viburnum_Tinus', 'Quercus_Trojana', 'Rhododendron_x_Russellianum',
'Acer_Saccharinum', 'Quercus_Ellipsoidalis', 'Quercus_Afares',
'Betula_Austrosinensis', 'Lithocarpus_Edulis', 'Quercus_x_Turneri',
'Ulmus_Bergmanniana', 'Acer_Opalus', 'Liquidambar_Styraciflua',
'Quercus_Trojana', 'Quercus_Crassifolia', 'Quercus_Coccinea',
'Quercus_Coccifera', 'Acer_Capillipes', 'Quercus_Afares',
'Quercus_Dolicholepis', 'Lithocarpus_Edulis', 'Tilia_Platyphyllos',
'Quercus_Infectoria_sub', 'Quercus_Castaneifolia', 'Populus_Nigra',
'Castanea_Sativa', 'Populus_Nigra', 'Tilia_Tomentosa',
'Callicarpa_Bodinieri', 'Acer_Capillipes', 'Ilex_Aquifolium',
'Salix_Fragilis', 'Magnolia_Salicifolia', 'Quercus_Dolicholepis',
'Prunus_X_Shmittii', 'Olea_Europaea', 'Acer_Platanoids',
'Fagus_Sylvatica', 'Sorbus_Aria', 'Quercus_Phillyraeoides',
'Quercus_Suber', 'Alnus_Rubra', 'Quercus_Vulcanica', 'Prunus_Avium',
'Acer_Platanoids', 'Quercus_Semecarpifolia', 'Acer_Mono',
'Cornus_Chinensis', 'Liquidambar_Styraciflua', 'Phildelphus',
'Acer_Platanoids', 'Quercus_Shumardii', 'Quercus_Rubra',
'Quercus_Texana', 'Acer_Pictum', 'Quercus_Crassipes',
'Lithocarpus_Cleistocarpus', 'Phildelphus', 'Ilex_Aquifolium',
'Quercus_Kewensis', 'Quercus_Cerris', 'Quercus_Palustris',
'Populus_Grandidentata', 'Quercus_Coccinea', 'Olea_Europaea',
'Quercus_Ilex', 'Quercus_x_Hispanica', 'Zelkova_Serrata',
'Quercus_Trojana', 'Salix_Intergra', 'Quercus_Greggii',
'Quercus_Texana', 'Alnus_Maximowiczii', 'Quercus_Semecarpifolia',
'Quercus_Cerris', 'Acer_Opalus', 'Pterocarya_Stenoptera',
'Prunus_X_Shmittii', 'Celtis_Koraiensis', 'Quercus_Pontica',
'Arundinaria_Simonii', 'Ilex_Aquifolium', 'Pterocarya_Stenoptera',
'Salix_Fragilis', 'Populus_Adenopoda', 'Acer_Rubrum',
'Liquidambar_Styraciflua', 'Alnus_Viridis', 'Quercus_Castaneifolia',
'Quercus_Nigra', 'Ilex_Cornuta', 'Populus_Adenopoda',
'Ginkgo_Biloba', 'Eucalyptus_Urnigera', 'Liriodendron_Tulipifera',
'Acer_Saccharinum', 'Quercus_Phellos', 'Fagus_Sylvatica',
'Quercus_Crassipes', 'Quercus_Afares', 'Quercus_Texana',
'Ilex_Cornuta', 'Eucalyptus_Glaucescens', 'Cornus_Chinensis',
'Tilia_Oliveri', 'Quercus_Phellos', 'Alnus_Maximowiczii',
```

```
'Castanea_Sativa', 'Acer_Palmatum', 'Quercus_x_Turneri',
'Quercus_Agrifolia', 'Quercus_Palustris', 'Magnolia_Heptapeta',
'Quercus_Variabilis', 'Ginkgo_Biloba', 'Quercus_Ilex',
'Liriodendron_Tulipifera', 'Morus_Nigra', 'Fagus_Sylvatica',
'Quercus_Shumardii', 'Acer_Palmatum', 'Crataegus_Monogyna',
'Quercus_Infectoria_sub', 'Viburnum_Tinus', 'Celtis_Koraiensis',
'Magnolia_Salicifolia', 'Ilex_Cornuta', 'Magnolia_Heptapeta',
'Quercus_Cerris', 'Quercus_Phillyraeoides', 'Quercus_Pyrenaica',
'Quercus_Variabilis', 'Acer_Rubrum', 'Quercus_Imbricaria',
'Lithocarpus_Cleistocarpus', 'Acer_Rufinerve', 'Ulmus_Bergmanniana',
'Quercus_Vulcanica', 'Quercus_Dolicholepis', 'Alnus_Cordata',
'Quercus_Hartwissiana', 'Morus_Nigra', 'Acer_Mono',
'Quercus_Alnifolia', 'Quercus_x_Hispanica', 'Morus_Nigra',
'Quercus_Alnifolia', 'Acer_Opalus', 'Pterocarya_Stenoptera',
'Prunus_Avium', 'Quercus_Pyrenaica', 'Quercus_Alnifolia',
'Betula_Austrosinensis', 'Crataegus_Monogyna',
'Rhododendron_x_Russellianum', 'Quercus_Ellipsoidalis',
'Populus_Grandidentata', 'Magnolia_Heptapeta', 'Quercus_Ilex',
'Quercus_x_Turneri', 'Quercus_Pubescens', 'Quercus_Pontica',
'Acer_Rubrum', 'Quercus_Greggii', 'Quercus_Hartwissiana',
'Quercus_Rubra', 'Quercus_Ellipsoidalis', 'Quercus_Cerris',
'Cercis_Siliquastrum', 'Quercus_Crassipes',
'Rhododendron_x_Russellianum', 'Quercus_Dolicholepis',
'Sorbus_Aria', 'Quercus_Coccinea', 'Alnus_Rubra', 'Quercus_Pontica',
'Arundinaria_Simonii', 'Quercus_Vulcanica', 'Acer_Rufinerve',
'Quercus_Ilex', 'Quercus_Chrysolepis', 'Quercus_Trojana',
'Quercus_Texana', 'Quercus_Phellos', 'Quercus_Coccifera',
'Eucalyptus_Glaucescens', 'Acer_Rubrum', 'Quercus_Imbricaria',
'Quercus_Ilex', 'Quercus_Chrysolepis', 'Quercus_Ilex',
'Quercus_Pubescens', 'Liquidambar_Styraciflua',
'Callicarpa_Bodinieri', 'Acer_Saccharinum', 'Quercus_Canariensis',
'Sorbus_Aria', 'Quercus_Hartwissiana', 'Castanea_Sativa',
'Arundinaria_Simonii', 'Alnus_Cordata', 'Cornus_Macrophylla',
'Quercus_Palustris', 'Quercus_Kewensis', 'Prunus_X_Shmittii',
'Magnolia_Heptapeta', 'Zelkova_Serrata', 'Betula_Austrosinensis',
'Castanea_Sativa', 'Populus_Grandidentata', 'Quercus_Pontica',
'Acer_Platanoids', 'Quercus_Suber', 'Olea_Europaea', 'Ilex_Cornuta',
'Quercus_Shumardii', 'Prunus_Avium', 'Quercus_Vulcanica',
'Eucalyptus_Glaucescens', 'Quercus_Vulcanica', 'Prunus_Avium',
'Cotinus_Coggygria', 'Cytisus_Battandieri', 'Quercus_Coccifera',
'Quercus_Infectoria_sub', 'Quercus_Variabilis',
'Viburnum_x_Rhytidophylloides', 'Quercus_Ellipsoidalis',
'Viburnum_x_Rhytidophylloides', 'Quercus_Pyrenaica',
'Quercus_Suber', 'Quercus_x_Turneri', 'Quercus_Rhysophylla',
'Alnus_Sieboldiana', 'Acer_Capillipes', 'Eucalyptus_Glaucescens',
'Morus_Nigra', 'Alnus_Viridis', 'Alnus_Rubra', 'Ilex_Aquifolium',
'Quercus_x_Hispanica', 'Populus_Nigra', 'Acer_Palmatum',
'Acer_Palmatum', 'Prunus_Avium', 'Eucalyptus_Urnigera',
```

```
        'Castanea_Sativa', 'Quercus_Coccifera', 'Alnus_Sieboldiana',
        'Quercus_Rhysophylla', 'Zelkova_Serrata', 'Acer_Pictum',
        'Pterocarya_Stenoptera', 'Quercus_Phellos', 'Quercus_Brantii',
        'Ginkgo_Biloba', 'Quercus_Rhysophylla', 'Magnolia_Salicifolia',
        'Quercus_Infectoria_sub', 'Crataegus_Monogyna', 'Tilia_Oliveri',
        'Betula_Austrosinensis', 'Quercus_Suber',
        'Lithocarpus_Cleistocarpus', 'Alnus_Viridis', 'Tilia_Platyphyllos',
        'Quercus_Shumardii', 'Quercus_Brantii', 'Populus_Nigra',
        'Cotinus_Coggygria', 'Quercus_Afares', 'Acer_Rubrum',
        'Lithocarpus_Cleistocarpus', 'Tilia_Platyphyllos',
        'Acer_Capillipes', 'Celtis_Koraiensis', 'Quercus_Canariensis',
        'Alnus_Cordata', 'Sorbus_Aria', 'Magnolia_Salicifolia',
        'Quercus_Suber', 'Quercus_Brantii', 'Callicarpa_Bodinieri',
        'Cytisus_Battandieri', 'Acer_Circinatum', 'Alnus_Rubra',
        'Quercus_Canariensis', 'Quercus_Phillyraeoides',
        'Arundinaria_Simonii'], dtype=object)
```

```
In [102]: le.fit(species)
          sp1 = le.classes_  #Transforming the non-numerical labels(species) to the 99 unique cl
          sp1
```

```
Out[102]: array(['Acer_Capillipes', 'Acer_Circinatum', 'Acer_Mono', 'Acer_Opalus',
                 'Acer_Palmatum', 'Acer_Pictum', 'Acer_Platanoids', 'Acer_Rubrum',
                 'Acer_Rufinerve', 'Acer_Saccharinum', 'Alnus_Cordata',
                 'Alnus_Maximowiczii', 'Alnus_Rubra', 'Alnus_Sieboldiana',
                 'Alnus_Viridis', 'Arundinaria_Simonii', 'Betula_Austrosinensis',
                 'Betula_Pendula', 'Callicarpa_Bodinieri', 'Castanea_Sativa',
                 'Celtis_Koraiensis', 'Cercis_Siliquastrum', 'Cornus_Chinensis',
                 'Cornus_Controversa', 'Cornus_Macrophylla', 'Cotinus_Coggygria',
                 'Crataegus_Monogyna', 'Cytisus_Battandieri',
                 'Eucalyptus_Glaucescens', 'Eucalyptus_Neglecta',
                 'Eucalyptus_Urnigera', 'Fagus_Sylvatica', 'Ginkgo_Biloba',
                 'Ilex_Aquifolium', 'Ilex_Cornuta', 'Liquidambar_Styraciflua',
                 'Liriodendron_Tulipifera', 'Lithocarpus_Cleistocarpus',
                 'Lithocarpus_Edulis', 'Magnolia_Heptapeta', 'Magnolia_Salicifolia',
                 'Morus_Nigra', 'Olea_Europaea', 'Phildelphus', 'Populus_Adenopoda',
                 'Populus_Grandidentata', 'Populus_Nigra', 'Prunus_Avium',
                 'Prunus_X_Shmittii', 'Pterocarya_Stenoptera', 'Quercus_Afares',
                 'Quercus_Agrifolia', 'Quercus_Alnifolia', 'Quercus_Brantii',
                 'Quercus_Canariensis', 'Quercus_Castaneifolia', 'Quercus_Cerris',
                 'Quercus_Chrysolepis', 'Quercus_Coccifera', 'Quercus_Coccinea',
                 'Quercus_Crassifolia', 'Quercus_Crassipes', 'Quercus_Dolicholepis',
                 'Quercus_Ellipsoidalis', 'Quercus_Greggii', 'Quercus_Hartwissiana',
                 'Quercus_Ilex', 'Quercus_Imbricaria', 'Quercus_Infectoria_sub',
                 'Quercus_Kewensis', 'Quercus_Nigra', 'Quercus_Palustris',
                 'Quercus_Phellos', 'Quercus_Phillyraeoides', 'Quercus_Pontica',
                 'Quercus_Pubescens', 'Quercus_Pyrenaica', 'Quercus_Rhysophylla',
                 'Quercus_Rubra', 'Quercus_Semecarpifolia', 'Quercus_Shumardii',
```

```
                 'Quercus_Suber', 'Quercus_Texana', 'Quercus_Trojana',
                 'Quercus_Variabilis', 'Quercus_Vulcanica', 'Quercus_x_Hispanica',
                 'Quercus_x_Turneri', 'Rhododendron_x_Russellianum',
                 'Salix_Fragilis', 'Salix_Intergra', 'Sorbus_Aria', 'Tilia_Oliveri',
                 'Tilia_Platyphyllos', 'Tilia_Tomentosa', 'Ulmus_Bergmanniana',
                 'Viburnum_Tinus', 'Viburnum_x_Rhytidophylloides', 'Zelkova_Serrata'], dtype=obj
```

```
In [94]: k = test2['id']
         k = np.array(k)   #converting the column id to numpy array.

In [105]: p = np.vstack((sp1, Probabilities)) #vertically stacking the.

In [99]: pp=np.matrix(p) #creating a matrix of the  99 unique species to the probabilities.

In [85]: species.shape

Out[85]: (594,)

In [117]: dd=pd.DataFrame(pp) #converting the matrix to a dataframe
          dd2 = dd.rename(columns=dd.loc[0,:]).loc[1:,:] #removes the row with indexes and retai

In [118]: dd2.loc[:,'id'] = k #Adds the column id to the dataframe

In [119]: columns = dd2.columns.tolist()

In [120]: #columns = columns[-1:] + columns[:-1]

In [121]: dd2 =dd2[columns]

In [91]: dd2.to_csv('Solutionsleave.csv', index = False)
```

The score from kaggle for the best model 0.10739

The objective is to minimize the logloss value.The lower the logloss value,the higher the accuracy level of the predicted values.