In [1]:

```
#importar
import pandas
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn import preprocessing

import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
from statistics import mean
from numpy.linalg import norm
from matplotlib.colors import ListedColormap
import random
import time
import timeit
```

In [2]:

```
#importação da base de dados
#os dados estão disponíveis abertamente em http://archive.ics.uci.edu/ml/datasets/Cover
type

file = "covtype.data"

#nome dos atributos
#estamos descartando os atributos que descrevem o tipo do solo
names = ['Elevation', 'Aspect', 'Slope', 'Horizontal_Distance_To_Hydrology', 'Vertical_
Distance_To_Hydrology', 'Horizontal_Distance_To_Roadways', 'Hillshade_9am', 'Hillshade_
Noon', 'Hillshade_3pm', 'Horizontal_Distance_To_Fire_Points', 'Wilderness_Area_Rawah',
'Wilderness_Area_Neota', 'Wilderness_Area_Comanche', 'Wilderness_Area_Cache', 'Cover_Ty
pe']

#de 0 a 13 são os atributos listados acima, e 54 é a classificação
usecols = list(range(0, 14)) + [54]

#especifico o tipo de alguns parametros(os que não são simplesmente numéricos)
dtype = {'Cover_Type': 'category', 'Wilderness_Area_Rawah' : bool, 'Wilderness_Area_Neo
ta' : bool, 'Wilderness_Area_Comanche' : bool, 'Wilderness_Area_Cache' : bool}

#lê o arquivo num pandas.dataframe
dataset = pandas.read_csv(file, header = None, usecols = usecols, names = names, dtype
= dtype)

#adicionando uma coluna adicional para sintetizar os 4 boleanos que representam a Wilde
rness_area.
#para uma única instância, somente um dos 4 booleanos pode ser verdadeiro, logo eles, e
m realidade, funcionam como uma categorização
new_column = pandas.Series([1 if dataset['Wilderness_Area_Rawah'][i] else
                            2 if dataset['Wilderness_Area_Neota'][i] else
                            3 if dataset['Wilderness_Area_Comanche'][i] else
                            4 for i in range(len(dataset.index)) ], dtype="category")
#elimina as colunas reduzidas
dataset = dataset.drop(columns=['Wilderness_Area_Rawah', 'Wilderness_Area_Neota', 'Wild
erness_Area_Comanche', 'Wilderness_Area_Cache'])
#insere nova coluna na posição 10
dataset.insert(loc = 10, column = 'Wilderness_Area', value = new_column)

#atualiza names para refletir a mudança acima
names = ['Elevation', 'Aspect', 'Slope', 'Horizontal_Distance_To_Hydrology', 'Vertical_
Distance_To_Hydrology', 'Horizontal_Distance_To_Roadways', 'Hillshade_9am', 'Hillshade_
Noon', 'Hillshade_3pm', 'Horizontal_Distance_To_Fire_Points', 'Wilderness_Area', 'Cover
_Type']

print("Dataframe com %d exemplares e %d atributos importado com sucesso" % (dataset.val
ues.shape[0], dataset.values.shape[1]))
```

```
Dataframe com 581012 exemplares e 12 atributos importado com sucesso
```

In [3]:

```
#test options and evaluation metric
seed = 7
scoring = 'accuracy'
#os atributos para treino em X e a classificação correta em Y
```

In [4]:

```
array = dataset.values
X = array[:, 0:11]
Y = array[:, 11]


#preprocessa dataset
#estava preprocessando, mas não fazia diferença
#X_scale_temp = preprocessing.scale(X[:, 0:10])
#X_scaled = np.append(X_scale_temp, X[:, 10].reshape(-1, 1), axis = 1)

#separamos um pedaço do dataset para usarmos de varificação depois
validation_size = 0.2
seed = 7
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y, t
est_size = validation_size, random_state = seed)

print("Set de treinamento: %d amostras" % len(X_train))
print("Set de validação: %d amostras" % len(X_validation))
```

```
Set de treinamento: 464809 amostras
Set de validação: 116203 amostras
```

Full Decision Tree

In [5]:

```
#verificamos a acurácia do método aplicando-o sobre o set de validação que separamos an
teriormente
dtc = DecisionTreeClassifier(random_state = seed)
t0 = time.process_time()
dtc.fit(X_train, Y_train)
tf = time.process_time()
print("Training time in seconds, for %d samples: %f" % (len(X_train), tf - t0))
predictions = dtc.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))

#numero de amostras aleatórias
N = 1000
#numero de avaliações dessa amostra pelo timeit
N2 = 10

#lista de N aleatórios a serem avaliados, pra medir o tempo médio de avaliação
l = random.sample(range(0, len(X_validation)), N)
time_list = []
for i in l:
    reshape = X_validation[i].reshape(1, -1)
    time_list.append((timeit.timeit("dtc.predict(reshape)", "from __main__ import dtc,
 reshape", number = N2) / N2))


print("Tempo médio para uma predição: %f segundos" % mean(time_list))
```

```
Training time in seconds, for 464809 samples: 15.234375
0.9265681608908548
[[39422  2838     4     0    37     9   214]
 [ 2771 52935   215     1   285   145    34]
 [    7   199  6553    79    13   334     0]
 [    0     1    95   477     0    27     0]
 [   46   303    20     0  1524    11     1]
 [   12   150   324    35    12  2923     0]
 [  268    42     0     0     1     0  3836]]
              precision    recall  f1-score   support

           1       0.93      0.93      0.93     42524
           2       0.94      0.94      0.94     56386
           3       0.91      0.91      0.91      7185
           4       0.81      0.80      0.80       600
           5       0.81      0.80      0.81      1905
           6       0.85      0.85      0.85      3456
           7       0.94      0.93      0.93      4147

   micro avg       0.93      0.93      0.93    116203
   macro avg       0.88      0.88      0.88    116203
weighted avg       0.93      0.93      0.93    116203

Tempo médio para uma predição: 0.000256 segundos
```

KNN

In [6]:

```
knn = KNeighborsClassifier(n_neighbors = 3)
t0 = time.process_time()
knn.fit(X_train, Y_train)
tf = time.process_time()
print("Training time in seconds, for %d samples: %f" % (len(X_train), tf - t0))
predictions = knn.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))

#numero de amostras aleatórias
N = 1000
#numero de avaliações dessa amostra pelo timeit
N2 = 10

#lista de N aleatórios a serem avaliados, pra medir o tempo médio de avaliação
l = random.sample(range(0, len(X_validation)), N)
time_list = []
for i in l:
    reshape = X_validation[i].reshape(1, -1)
    time_list.append((timeit.timeit("knn.predict(reshape)", "from __main__ import knn,
 reshape", number = N2) / N2))


print("Tempo médio para uma predição: %f segundos" % mean(time_list))
```

```
Training time in seconds, for 464809 samples: 5.015625
0.9696221267953495
[[41232  1183     1     0    10     1    97]
 [ 1114 55002    58     0   143    53    16]
 [    0    87  6956    25    13   104     0]
 [    0     0    79   487     0    34     0]
 [   18   146    15     0  1717     8     1]
 [    4    71   112    15     5  3249     0]
 [   95    22     0     0     0     0  4030]]
              precision    recall  f1-score   support

           1       0.97      0.97      0.97     42524
           2       0.97      0.98      0.97     56386
           3       0.96      0.97      0.97      7185
           4       0.92      0.81      0.86       600
           5       0.91      0.90      0.91      1905
           6       0.94      0.94      0.94      3456
           7       0.97      0.97      0.97      4147

   micro avg       0.97      0.97      0.97    116203
   macro avg       0.95      0.93      0.94    116203
weighted avg       0.97      0.97      0.97    116203

Tempo médio para uma predição: 0.001259 segundos
```