feature_selection

June 7, 2017

1 Part 3 -- Feature Selection

The libraries that we are going to use:

```
In [281]: import pandas as pd
          import math
          from math import log
          from sklearn.naive_bayes import MultinomialNB, BernoulliNB
          from sklearn.feature_extraction.text import TfidfTransformer
          from sklearn.feature_extraction.text import CountVectorizer
          import numpy as np
          from sklearn import svm
          from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
          from sklearn.metrics import classification_report, accuracy_score, auc
          from sklearn.model_selection import KFold
          from sklearn.decomposition import TruncatedSVD
          from sklearn.linear_model import SGDClassifier
          from sklearn import metrics
          import matplotlib.pyplot as plt1
          import matplotlib.pyplot as plt2

          from wordcloud import STOPWORDS
          from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
          import csv
          import random
          import math
          import operator
          from operator import itemgetter
          from collections import Counter
```

We read our training and testing data:

```
In [282]: Train_data = pd.read_csv(sep='\t',filepath_or_buffer='train.tsv')
          Test_data = pd.read_csv(sep='\t',filepath_or_buffer='test.tsv')
          target = Train_data['Label']
```

1.0.1 Our Cross-Validation function

```
In [283]: cross_val_instance = 0

          def cross_validate(clf,train_data,target_data):
              global cross_val_instance    # Needed to modify global copy of a global variable

              kf = KFold(n_splits=10)
              average_accuracy =0
              fold = 0
              for train_index, test_index in kf.split(train_data):
                  cross_val_instance += 1

                  test = train_data.loc[test_index, train_data.columns]
                  train = train_data.loc[train_index, train_data.columns]
                  target = target_data[train_index]
                  clf_cv = clf.fit(train, target)
                  yPred = clf_cv.predict(test)
                  fold += 1
                  print ("Fold " + str(fold)+"\n\n")
                  target = target_data[test_index]
                  accuracy = accuracy_score(target, yPred)
                  # mylist['Accuracy'].append(accuracy)
                  print("Accuracy: ", accuracy)
                  average_accuracy+= accuracy
              average_accuracy = average_accuracy/10
              print("Average accuracy = ",average_accuracy)
              return average_accuracy
```

```
In [284]: Train_data.head()
```

```
Out[284]:   Attribute1  Attribute2 Attribute3 Attribute4  Attribute5 Attribute6  \
          0        A11           6        A34        A43        1169        A65
          1        A12          48        A32        A43        5951        A61
          2        A14          12        A34        A46        2096        A61
          3        A11          42        A32        A42        7882        A61
          4        A11          24        A33        A40        4870        A61

            Attribute7  Attribute8 Attribute9 Attribute10  ...    Attribute13  \
          0        A75           4        A93        A101  ...             67
          1        A73           2        A92        A101  ...             22
          2        A74           2        A93        A101  ...             49
          3        A74           2        A93        A103  ...             45
          4        A73           3        A93        A101  ...             53

            Attribute14  Attribute15 Attribute16 Attribute17  Attribute18 Attribute19  \
          0        A143         A152           2        A173            1        A192
          1        A143         A152           1        A173            1        A191
```

```
2         A143         A152           1        A172           2        A191
3         A143         A153           1        A173           2        A191
4         A143         A153           2        A173           2        A191

   Attribute20 Label     Id
0         A201     1  10101
1         A201     2  10102
2         A201     1  10103
3         A201     1  10104
4         A201     2  10105

[5 rows x 22 columns]
```

1.1 Data preprocessing

In [285]:
```
categories = ["Attribute1","Attribute3","Attribute4","Attribute6","Attribute7","Attrib

proccessedData_train = Train_data.copy()
proccessedData_test = Test_data.copy()
for x in categories:
    converted = pd.Categorical(Train_data[x])
    proccessedData_train[x] = converted.codes
    converted_test = pd.Categorical(Test_data[x])
    proccessedData_test[x] = converted_test.codes

print(proccessedData_train)
```

```
     Attribute1  Attribute2  Attribute3  Attribute4  Attribute5  Attribute6  \
0             0           6           4           4        1169           4
1             1          48           2           4        5951           0
2             3          12           4           7        2096           0
3             0          42           2           3        7882           0
4             0          24           3           0        4870           0
5             3          36           2           7        9055           4
6             3          24           2           3        2835           2
7             1          36           2           1        6948           0
8             3          12           2           4        3059           3
9             1          30           4           0        5234           0
10            1          12           2           0        1295           0
11            0          48           2           9        4308           0
12            1          12           2           4        1567           0
13            0          24           4           0        1199           0
14            0          15           2           0        1403           0
15            0          24           2           4        1282           1
16            3          24           4           4        2424           4
17            0          30           0           9        8072           4
18            1          24           2           1       12579           0
19            3          24           2           4        3430           2
```

20	3	9	4	0	2134	0
21	0	6	2	4	2647	2
22	0	10	4	0	2241	0
23	1	12	4	1	1804	1
24	3	10	4	3	2069	4
25	0	6	2	3	1374	0
26	3	6	0	4	426	0
27	2	12	1	4	409	3
28	1	7	2	4	2415	0
29	0	60	3	9	6836	0
..	...	...	...	...	...	...
770	0	24	2	1	2812	4
771	0	36	4	7	8065	0
772	3	21	4	1	3275	0
773	3	24	4	4	2223	1
774	2	12	4	0	1480	2
775	0	24	2	0	1371	4
776	3	36	4	0	3535	0
777	0	18	2	4	3509	0
778	3	36	4	1	5711	3
779	1	18	2	6	3872	0
780	1	39	4	4	4933	0
781	3	24	4	0	1940	3
782	1	12	0	8	1410	0
783	1	12	2	0	836	1
784	1	20	2	1	6468	4
785	1	18	2	9	1941	3
786	3	22	2	4	2675	2
787	3	48	4	1	2751	4
788	1	48	3	7	6224	0
789	0	40	4	7	5998	0
790	1	21	2	9	1188	0
791	3	24	2	1	6313	4
792	3	6	4	3	1221	4
793	2	24	2	3	2892	0
794	3	24	2	3	3062	2
795	3	9	2	3	2301	1
796	0	18	2	1	7511	4
797	3	12	4	3	1258	0
798	3	24	3	0	717	4
799	1	9	2	0	1549	4

	Attribute7	Attribute8	Attribute9	Attribute10	...	Attribute13	\
0	4	4	2	0	...	67	
1	2	2	1	0	...	22	
2	3	2	2	0	...	49	
3	3	2	2	2	...	45	
4	2	3	2	0	...	53	

5	2	2	2	0	...	35
6	4	3	2	0	...	53
7	2	2	2	0	...	35
8	3	2	0	0	...	61
9	0	4	3	0	...	28
10	1	3	1	0	...	25
11	1	3	1	0	...	24
12	2	1	1	0	...	22
13	4	4	2	0	...	60
14	2	2	1	0	...	28
15	2	4	1	0	...	32
16	4	4	2	0	...	53
17	1	2	2	0	...	25
18	4	4	1	0	...	44
19	4	3	2	0	...	31
20	2	4	2	0	...	48
21	2	2	2	0	...	44
22	1	1	2	0	...	48
23	1	3	2	0	...	44
24	2	2	3	0	...	26
25	2	1	2	0	...	36
26	4	4	3	0	...	39
27	2	3	1	0	...	42
28	2	3	2	2	...	34
29	4	3	2	0	...	63
..	...	...	...	...	...	...
770	4	2	1	0	...	26
771	2	3	1	0	...	25
772	4	1	2	0	...	36
773	4	4	2	0	...	52
774	0	2	2	0	...	66
775	2	4	1	0	...	25
776	3	4	2	0	...	37
777	3	4	1	2	...	25
778	4	4	2	0	...	38
779	0	2	1	0	...	67
780	3	2	2	2	...	25
781	4	4	2	0	...	60
782	2	2	2	0	...	31
783	1	4	1	0	...	23
784	0	1	0	0	...	60
785	2	4	2	0	...	35
786	4	3	2	0	...	40
787	4	4	2	0	...	38
788	4	4	2	0	...	50
789	2	4	2	0	...	27
790	4	2	1	0	...	39
791	4	3	2	0	...	41

792	2	1	3	0	...	27
793	4	3	0	0	...	51
794	4	4	2	0	...	32
795	1	2	1	0	...	22
796	4	1	2	0	...	51
797	1	2	1	0	...	22
798	4	4	3	0	...	54
799	1	4	2	0	...	35

	Attribute14	Attribute15	Attribute16	Attribute17	Attribute18	\
0	2	1	2	2	1	
1	2	1	1	2	1	
2	2	1	1	1	2	
3	2	2	1	2	2	
4	2	2	2	2	2	
5	2	2	1	1	2	
6	2	1	1	2	1	
7	2	0	1	3	1	
8	2	1	1	1	1	
9	2	1	2	3	1	
10	2	0	1	2	1	
11	2	0	1	2	1	
12	2	1	1	2	1	
13	2	1	2	1	1	
14	2	0	1	2	1	
15	2	1	1	1	1	
16	2	1	2	2	1	
17	0	1	3	2	1	
18	2	2	1	3	1	
19	2	1	1	2	2	
20	2	1	3	2	1	
21	2	0	1	2	2	
22	2	0	2	1	2	
23	2	1	1	2	1	
24	2	1	2	2	1	
25	0	1	1	1	1	
26	2	1	1	1	1	
27	2	0	2	2	1	
28	2	1	1	2	1	
29	2	1	2	2	1	
..	...	...	...	...	...	
770	2	0	1	2	1	
771	2	1	2	3	1	
772	2	1	1	3	1	
773	0	1	2	2	1	
774	0	2	3	0	1	
775	2	0	1	2	1	
776	2	1	2	2	1	

777	2	1	1	2	1
778	2	1	2	3	1
779	2	1	1	2	1
780	2	1	2	2	1
781	2	1	1	2	1
782	2	1	1	1	1
783	0	1	1	1	1
784	2	1	1	3	1
785	2	1	1	1	1
786	2	1	1	2	1
787	2	1	2	2	2
788	2	2	1	2	1
789	0	1	1	2	1
790	2	1	1	2	2
791	2	1	1	3	2
792	2	1	2	2	1
793	2	2	1	2	1
794	2	0	1	2	1
795	2	0	1	2	1
796	2	2	1	2	2
797	2	0	2	1	1
798	2	1	2	2	1
799	2	1	1	0	1

	Attribute19	Attribute20	Label	Id
0	1	0	1	10101
1	0	0	2	10102
2	0	0	1	10103
3	0	0	1	10104
4	0	0	2	10105
5	1	0	1	10106
6	0	0	1	10107
7	1	0	1	10108
8	0	0	1	10109
9	0	0	2	10110
10	0	0	2	10111
11	0	0	2	10112
12	1	0	1	10113
13	0	0	2	10114
14	0	0	1	10115
15	0	0	2	10116
16	0	0	1	10117
17	0	0	1	10118
18	1	0	2	10119
19	1	0	1	10120
20	1	0	1	10121
21	0	0	1	10122
22	0	1	1	10123

```
23              0            0      1  10124
24              0            1      1  10125
25              1            0      1  10126
26              0            0      1  10127
27              0            0      1  10128
28              0            0      1  10129
29              1            0      2  10130
..            ...          ...    ...    ...
770             0            0      1  10871
771             1            0      2  10872
772             1            0      1  10873
773             0            0      1  10874
774             0            0      1  10875
775             0            0      2  10876
776             1            0      1  10877
777             0            0      1  10878
778             1            0      1  10879
779             1            0      1  10880
780             0            0      2  10881
781             1            0      1  10882
782             1            0      1  10883
783             0            0      2  10884
784             1            0      1  10885
785             1            0      1  10886
786             0            0      1  10887
787             1            0      1  10888
788             0            0      2  10889
789             1            0      2  10890
790             0            0      2  10891
791             1            0      1  10892
792             0            0      1  10893
793             0            0      1  10894
794             1            0      1  10895
795             0            0      1  10896
796             1            0      2  10897
797             0            0      1  10898
798             1            0      1  10899
799             0            0      1  10900

[800 rows x 22 columns]
```

1.1.1 Our gain calculation

```
In [286]: def get_number_of_Good(data):
              return  len(data[data["Label"]==1])
          def get_number_of_Bad(data):
              return  len(data[data["Label"]==2])
```

1.1.2 Our entropy calculation function

```
In [287]: def entropy(data):

              if(data.shape[0]==0): #se periptwsh pou to dataset einai adeio
                  return 0

              good_percent = get_number_of_Good(data) /data.shape[0]
              bad_percent = get_number_of_Bad(data)/data.shape[0]
              good_entropy =0.0
              if(get_number_of_Good(data)>0):
                  good_entropy = -(good_percent*log(good_percent, 2))
              bad_entropy =0.0
              if(get_number_of_Bad(data)>0):
                  bad_entropy = -(bad_percent*log(bad_percent, 2))
              entropy = good_entropy+bad_entropy
              return entropy

In [288]: print(entropy(Train_data)) #entropy for the hole dataset
```

```
0.87975753726356
```

```
In [289]: def get_attribute_values(data):#synarthsh pou epistrefei ta values enos attribute
              values = []
              for x in data:
                  #print(x)
                  if(x not in values ):
                      values.append(x)
              return values

In [290]: features = list(Train_data.columns.values)
          features.remove('Id')
          features.remove('Label')

          #kanw ta attribute ayta categorical ['Attribute2', 'Attribute5', 'Attribute13']
          #den kanoume kai ta alla (numerical)attribute dioti exoun to poly 4 diaforetikes times

          Train_data['Attribute2']= pd.qcut(Train_data['Attribute2'], 5, labels=["Attribute2_A",
          Train_data['Attribute5']= pd.qcut(Train_data['Attribute5'], 5, labels=["Attribute5_A",
          Train_data['Attribute13']= pd.qcut(Train_data['Attribute13'], 5, labels=["Attribute13_

          InformationGain_list = []
          for Attribute in features:
              Attribute_values = get_attribute_values(Train_data[Attribute])
              attr_entropy = 0.0
              for value in Attribute_values:#for every value in Attribute values
                  value_set= Train_data[Train_data[Attribute]==value]
                  attr_entropy += (len(value_set)/(len(Train_data)))*entropy(value_set)  #ypolog
```

```
            attr_information_gain = entropy(Train_data)-attr_entropy
            InformationGain_list.append((Attribute,attr_information_gain))
        #print("Infomation Gain List(of all attributes ):\n\n",InformationGain_list)
        InformationGain_list
```

Out[290]:
```
[('Attribute1', 0.09382796302345509),
 ('Attribute2', 0.031782332193863394),
 ('Attribute3', 0.03788940622151615),
 ('Attribute4', 0.02689745203308369),
 ('Attribute5', 0.015294038701320956),
 ('Attribute6', 0.02219896605243432),
 ('Attribute7', 0.014547865230223445),
 ('Attribute8', 0.007330500076830004),
 ('Attribute9', 0.012746841156174304),
 ('Attribute10', 0.005674399790160045),
 ('Attribute11', 0.00022057134927411237),
 ('Attribute12', 0.014905530877295403),
 ('Attribute13', 0.0117447128999153),
 ('Attribute14', 0.007041506325139002),
 ('Attribute15', 0.011618886823694607),
 ('Attribute16', 0.002395770112591733),
 ('Attribute17', 0.0029403166312881313),
 ('Attribute18', 0.0001296657019278502),
 ('Attribute19', 0.0012028625910776025),
 ('Attribute20', 0.007704386546436126)]
```

In [291]:
```
print(sorted(InformationGain_list, key=lambda tup: tup[1]) )
InformationGain_list = sorted(InformationGain_list, key=lambda tup: tup[1])
InformationGain_list
```

```
[('Attribute18', 0.0001296657019278502), ('Attribute11', 0.00022057134927411237), ('Attribute19'
```

Out[291]:
```
[('Attribute18', 0.0001296657019278502),
 ('Attribute11', 0.00022057134927411237),
 ('Attribute19', 0.0012028625910776025),
 ('Attribute16', 0.002395770112591733),
 ('Attribute17', 0.0029403166312881313),
 ('Attribute10', 0.005674399790160045),
 ('Attribute14', 0.007041506325139002),
 ('Attribute8', 0.007330500076830004),
 ('Attribute20', 0.007704386546436126),
 ('Attribute15', 0.011618886823694607),
 ('Attribute13', 0.0117447128999153),
 ('Attribute9', 0.012746841156174304),
 ('Attribute7', 0.014547865230223445),
 ('Attribute12', 0.014905530877295403),
 ('Attribute5', 0.015294038701320956),
 ('Attribute6', 0.02219896605243432),
```

```
 ('Attribute4', 0.02689745203308369),
 ('Attribute2', 0.031782332193863394),
 ('Attribute3', 0.03788940622151615),
 ('Attribute1', 0.09382796302345509)]
```

In [292]:
```
RANDOM_STATE = 123

rndf = RandomForestClassifier(warm_start=False, oob_score=False, max_features="sqrt",
```

In [293]:
```
count = len(InformationGain_list)
average_accuracy_list=[]
average_accuracy_list_tuples = []
average_accuracy_list_accuracy =[]
average_accuracy_list_number_attributes = []
my_df = proccessedData_train
exclude = ['Id','Label']

for attr in InformationGain_list:#gia kathe stoixeio sthn InformationGain_list afairw
    print("count == ",count)

    excl = my_df.columns.difference(exclude)

    new_df_to_use  = my_df[excl]
    #efarmozoume cross validation gia kathe attribute pou kanoume exclude
    average_accuracy = cross_validate(rndf,new_df_to_use,proccessedData_train['Label']
    average_accuracy_list.append((average_accuracy,count) )#to meiwn 2 gt sthn lista e
    average_accuracy_list_accuracy.append(average_accuracy)
    average_accuracy_list_number_attributes.append(count)
    exclude.append(str(attr[0]))

    count -= 1

    if(count == -1):
        break;
```

```
count ==  20
Fold 1


Accuracy:  0.8
Fold 2


Accuracy:  0.7375
Fold 3


Accuracy:  0.7
```

```
Fold 4


Accuracy:  0.8
Fold 5


Accuracy:  0.825
Fold 6


Accuracy:  0.7375
Fold 7


Accuracy:  0.725
Fold 8


Accuracy:  0.625
Fold 9


Accuracy:  0.8125
Fold 10


Accuracy:  0.7375
Average accuracy =  0.75
count ==  19
Fold 1


Accuracy:  0.85
Fold 2


Accuracy:  0.725
Fold 3


Accuracy:  0.6625
Fold 4


Accuracy:  0.7625
Fold 5
```

```
Accuracy:  0.85
Fold 6


Accuracy:  0.7375
Fold 7


Accuracy:  0.7375
Fold 8


Accuracy:  0.625
Fold 9


Accuracy:  0.725
Fold 10


Accuracy:  0.7125
Average accuracy =  0.73875
count ==  18
Fold 1


Accuracy:  0.7625
Fold 2


Accuracy:  0.775
Fold 3


Accuracy:  0.6625
Fold 4


Accuracy:  0.8
Fold 5


Accuracy:  0.75
Fold 6


Accuracy:  0.725
```

```
Fold 7


Accuracy:  0.7125
Fold 8


Accuracy:  0.6875
Fold 9


Accuracy:  0.725
Fold 10


Accuracy:  0.6375
Average accuracy =  0.72375
count ==  17
Fold 1


Accuracy:  0.8125
Fold 2


Accuracy:  0.7625
Fold 3


Accuracy:  0.6625
Fold 4


Accuracy:  0.8125
Fold 5


Accuracy:  0.75
Fold 6


Accuracy:  0.7375
Fold 7


Accuracy:  0.7125
Fold 8
```

```
Accuracy:  0.6625
Fold 9


Accuracy:  0.7375
Fold 10


Accuracy:  0.725
Average accuracy =  0.7375
count ==  16
Fold 1


Accuracy:  0.8125
Fold 2


Accuracy:  0.7125
Fold 3


Accuracy:  0.65
Fold 4


Accuracy:  0.85
Fold 5


Accuracy:  0.7875
Fold 6


Accuracy:  0.7
Fold 7


Accuracy:  0.75
Fold 8


Accuracy:  0.675
Fold 9


Accuracy:  0.7625
```

```
Fold 10


Accuracy:  0.7375
Average accuracy =  0.74375
count ==  15
Fold 1


Accuracy:  0.8125
Fold 2


Accuracy:  0.675
Fold 3


Accuracy:  0.675
Fold 4


Accuracy:  0.7875
Fold 5


Accuracy:  0.825
Fold 6


Accuracy:  0.7625
Fold 7


Accuracy:  0.775
Fold 8


Accuracy:  0.6375
Fold 9


Accuracy:  0.7125
Fold 10


Accuracy:  0.7125
Average accuracy =  0.7375
count ==  14
```

```
Fold 1


Accuracy:  0.7625
Fold 2


Accuracy:  0.7
Fold 3


Accuracy:  0.65
Fold 4


Accuracy:  0.7875
Fold 5


Accuracy:  0.7625
Fold 6


Accuracy:  0.7625
Fold 7


Accuracy:  0.7625
Fold 8


Accuracy:  0.7
Fold 9


Accuracy:  0.75
Fold 10


Accuracy:  0.75
Average accuracy =  0.73875
count ==  13
Fold 1


Accuracy:  0.7375
Fold 2
```

```
Accuracy:  0.775
Fold 3


Accuracy:  0.65
Fold 4


Accuracy:  0.7875
Fold 5


Accuracy:  0.8375
Fold 6


Accuracy:  0.7
Fold 7


Accuracy:  0.7625
Fold 8


Accuracy:  0.6875
Fold 9


Accuracy:  0.7375
Fold 10


Accuracy:  0.75
Average accuracy =  0.7425
count ==  12
Fold 1


Accuracy:  0.7875
Fold 2


Accuracy:  0.725
Fold 3


Accuracy:  0.7125
```

```
Fold 4


Accuracy:  0.7625
Fold 5


Accuracy:  0.7625
Fold 6


Accuracy:  0.675
Fold 7


Accuracy:  0.7125
Fold 8


Accuracy:  0.65
Fold 9


Accuracy:  0.7375
Fold 10


Accuracy:  0.7375
Average accuracy =  0.72625
count ==  11
Fold 1


Accuracy:  0.8
Fold 2


Accuracy:  0.775
Fold 3


Accuracy:  0.6625
Fold 4


Accuracy:  0.7875
Fold 5
```

```
Accuracy:  0.7875
Fold 6


Accuracy:  0.7125
Fold 7


Accuracy:  0.7375
Fold 8


Accuracy:  0.6875
Fold 9


Accuracy:  0.7125
Fold 10


Accuracy:  0.7375
Average accuracy =  0.74
count ==  10
Fold 1


Accuracy:  0.7625
Fold 2


Accuracy:  0.725
Fold 3


Accuracy:  0.7125
Fold 4


Accuracy:  0.775
Fold 5


Accuracy:  0.85
Fold 6


Accuracy:  0.75
```

```
Fold 7


Accuracy:  0.75
Fold 8


Accuracy:  0.6625
Fold 9


Accuracy:  0.775
Fold 10


Accuracy:  0.75
Average accuracy =  0.75125
count ==  9
Fold 1


Accuracy:  0.7875
Fold 2


Accuracy:  0.725
Fold 3


Accuracy:  0.6875
Fold 4


Accuracy:  0.85
Fold 5


Accuracy:  0.8
Fold 6


Accuracy:  0.725
Fold 7


Accuracy:  0.75
Fold 8
```

```
Accuracy:  0.7125
Fold 9


Accuracy:  0.7625
Fold 10


Accuracy:  0.725
Average accuracy =  0.7525
count ==  8
Fold 1


Accuracy:  0.775
Fold 2


Accuracy:  0.7375
Fold 3


Accuracy:  0.6625
Fold 4


Accuracy:  0.775
Fold 5


Accuracy:  0.7875
Fold 6


Accuracy:  0.8
Fold 7


Accuracy:  0.725
Fold 8


Accuracy:  0.6875
Fold 9


Accuracy:  0.7375
```

```
Fold 10


Accuracy:  0.675
Average accuracy =  0.73625
count ==  7
Fold 1


Accuracy:  0.8
Fold 2


Accuracy:  0.725
Fold 3


Accuracy:  0.7125
Fold 4


Accuracy:  0.7875
Fold 5


Accuracy:  0.7375
Fold 6


Accuracy:  0.725
Fold 7


Accuracy:  0.75
Fold 8


Accuracy:  0.6625
Fold 9


Accuracy:  0.675
Fold 10


Accuracy:  0.7375
Average accuracy =  0.73125
count ==  6
```

```
Fold 1


Accuracy:  0.8375
Fold 2


Accuracy:  0.725
Fold 3


Accuracy:  0.6625
Fold 4


Accuracy:  0.7625
Fold 5


Accuracy:  0.825
Fold 6


Accuracy:  0.7125
Fold 7


Accuracy:  0.725
Fold 8


Accuracy:  0.6625
Fold 9


Accuracy:  0.7125
Fold 10


Accuracy:  0.75
Average accuracy =  0.7375
count ==  5
Fold 1


Accuracy:  0.775
Fold 2
```

```
Accuracy:  0.7375
Fold 3


Accuracy:  0.675
Fold 4


Accuracy:  0.775
Fold 5


Accuracy:  0.75
Fold 6


Accuracy:  0.725
Fold 7


Accuracy:  0.7125
Fold 8


Accuracy:  0.7375
Fold 9


Accuracy:  0.7
Fold 10


Accuracy:  0.725
Average accuracy =  0.73125
count ==  4
Fold 1


Accuracy:  0.775
Fold 2


Accuracy:  0.6875
Fold 3


Accuracy:  0.65
```

```
Fold 4


Accuracy:  0.725
Fold 5


Accuracy:  0.6875
Fold 6


Accuracy:  0.675
Fold 7


Accuracy:  0.75
Fold 8


Accuracy:  0.6375
Fold 9


Accuracy:  0.6625
Fold 10


Accuracy:  0.7
Average accuracy =  0.695
count ==  3
Fold 1


Accuracy:  0.8125
Fold 2


Accuracy:  0.65
Fold 3


Accuracy:  0.675
Fold 4


Accuracy:  0.7125
Fold 5
```

```
Accuracy:  0.7375
Fold 6


Accuracy:  0.675
Fold 7


Accuracy:  0.6625
Fold 8


Accuracy:  0.6875
Fold 9


Accuracy:  0.6875
Fold 10


Accuracy:  0.6625
Average accuracy =  0.69625
count ==  2
Fold 1


Accuracy:  0.775
Fold 2


Accuracy:  0.7875
Fold 3


Accuracy:  0.6875
Fold 4


Accuracy:  0.7875
Fold 5


Accuracy:  0.75
Fold 6


Accuracy:  0.7125
```

```
Fold 7


Accuracy:  0.7375
Fold 8


Accuracy:  0.65
Fold 9


Accuracy:  0.725
Fold 10


Accuracy:  0.65
Average accuracy =  0.72625
count ==  1
Fold 1


Accuracy:  0.75
Fold 2


Accuracy:  0.775
Fold 3


Accuracy:  0.6375
Fold 4


Accuracy:  0.75
Fold 5


Accuracy:  0.7375
Fold 6


Accuracy:  0.7
Fold 7


Accuracy:  0.65
Fold 8
```

```
Accuracy:  0.5875
Fold 9


Accuracy:  0.75
Fold 10


Accuracy:  0.675
Average accuracy =  0.70125
```

```
In [294]: print(average_accuracy_list)
```

```
[(0.74999999999999989, 20), (0.73875000000000002, 19), (0.72375, 18), (0.73749999999999993, 17),
```

```
In [295]: print(average_accuracy_list_number_attributes)
```

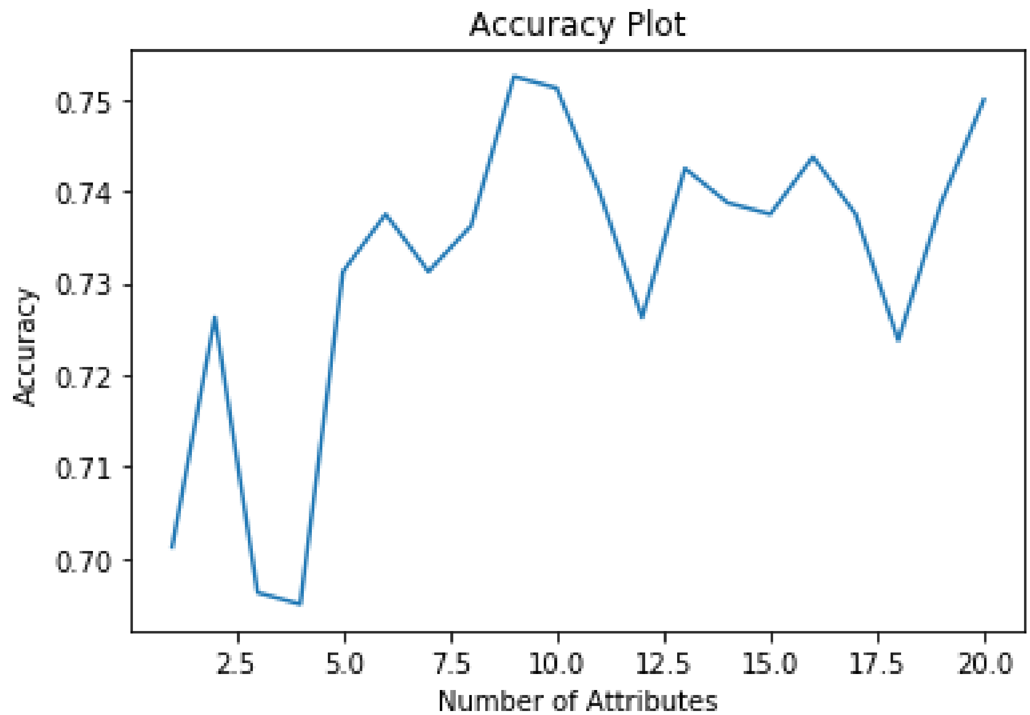
```
[20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
In [296]: print(average_accuracy_list_accuracy)
```

```
[0.74999999999999989, 0.73875000000000002, 0.72375, 0.73749999999999993, 0.74375000000000002, 0.
```

```
In [297]: plt1.title('Accuracy Plot')

          plt1.plot(average_accuracy_list_number_attributes, average_accuracy_list_accuracy)
          plt1.ylabel('Accuracy')
          plt1.xlabel('Number of Attributes')
          plt1.show()
```



1.2 The matrix of the features that we chose to remove in each repetition

```
In [298]: for x in InformationGain_list:
              print(x[0],"\t",x[1])
```

```
Attribute18          0.0001296657019278502
Attribute11          0.00022057134927411237
Attribute19          0.0012028625910776025
Attribute16          0.002395770112591733
Attribute17          0.0029403166312881313
Attribute10          0.005674399790160045
Attribute14          0.007041506325139002
Attribute8          0.007330500076830004
Attribute20          0.007704386546436126
Attribute15          0.011618886823694607
Attribute13          0.0117447128999153
Attribute9          0.012746841156174304
Attribute7          0.014547865230223445
Attribute12          0.014905530877295403
Attribute5          0.015294038701320956
Attribute6          0.02219896605243432
Attribute4          0.02689745203308369
Attribute2          0.031782332193863394
Attribute3          0.03788940622151615
```

```
Attribute1          0.09382796302345509
```

2 We find the max accuracy

```
In [299]: max_acc= average_accuracy_list[0][0] #h megisth accuracy
          max_features = average_accuracy_list[0][1] #metablhth pou kratame to posa features hta
          for x in average_accuracy_list:
              if(x[0] >max_acc ):
                  max_features= x[1]
                  max_acc = x[0]
          print(max_acc,max_features)
```

```
0.7525 9
```

2.0.1 The testSet_Predictions.csv implementation

We base our implementation on the best classifier and the best number of features.

```
In [300]: count = len(InformationGain_list)
          my_df = proccessedData_train
          exclude = ['Id','Label']
          for attr in InformationGain_list:
              print("count == ",count)
              exclude.append(str(attr[0]))
              count-=1
              if(count == max_features):
                  print(excl)
                  break;

          excl = my_df.columns.difference(exclude)
          print(excl)
          new_train_data  = my_df[excl] #tous afairw ta attributes pou xreiazetai
          new_test_data = my_df[excl]
```

```
count ==  20
count ==  19
count ==  18
count ==  17
count ==  16
count ==  15
count ==  14
count ==  13
count ==  12
count ==  11
count ==  10
Index(['Attribute1'], dtype='object')
```

```
Index(['Attribute1', 'Attribute12', 'Attribute2', 'Attribute3', 'Attribute4',
       'Attribute5', 'Attribute6', 'Attribute7', 'Attribute9'],
      dtype='object')
```

2.1 Random Forest (RF) Classification

```
In [301]: RANDOM_STATE = 123
          rndf= RandomForestClassifier()
          clf_cv = rndf.fit(new_train_data, target)
          predicted = clf_cv.predict(new_test_data)
```

```
In [302]: print(predicted)
```

```
[1 2 1 1 2 1 1 1 1 2 2 2 1 2 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1
 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 2 1 1 2 1 1 2 2 1 1 1 1 2 1 1 1 1 1
 2 1 2 1 1 1 2 1 1 1 1 1 1 2 1 2 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1
 1 1 2 1 1 2 1 1 1 2 1 1 1 2 1 1 2 1 2 1 2 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1
 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 2 1 2 2 1 1 1 1 2 2 2 1 2
 1 2 1 2 1 1 2 2 1 2 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1
 1 1 1 1 2 2 2 1 2 1 1 1 1 2 2 2 1 1 2 1 2 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 1
 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 2
 1 1 1 1 1 2 1 1 2 1 1 2 2 1 1 1 1 2 1 2 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 2 2
 2 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 1 2 1 2 1 2 1 1 1 1 2 1 1 1 2 1
 1 1 1 1 2 2 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2 1
 1 1 2 1 1 2 1 2 1 2 1 1 2 1 1 1 1 2 1 1 1 1 2 1 2 1 1 1 2 1 1 1 2 1 1 1 2
 2 1 2 1 1 2 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 2 2 2 1 2 2 1 1 1 1 1
 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 2 1 1 1 2 1 1 2 2 2 1 2 1 1 2 1 1 1 1 1 1 2
 1 1 1 2 2 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 2 1 2 1 2 2 1 2 1 1 2 1 1 1 2 1 1
 2 2 2 2 1 1 2 1 2 1 1 2 1 1 2 2 1 1 1 1 1 1 1 2 1 2 1 1 2 1 2 1 1 2 2 1 1
 1 2 2 2 2 2 2 1 1 2 2 2 1 1 1 2 1 1 2 2 1 1 2 1 1 1 2 1 1 1 2 1 2 1 1 2 1
 1 1 2 1 2 2 1 1 1 1 2 2 1 2 1 1 2 1 1 2 2 1 2 2 2 1 1 2 1 1 1 1 2 1 1 1 1
 1 1 2 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1
 1 1 1 2 2 1 1 1 2 1 1 2 1 1 1 1 1 2 2 2 1 2 1 1 2 2 1 1 2 1 1 1 1 2 1 1 2
 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2 2 1 2 1 2 1 2 1 2 1 1 2 1 1 1 1 2 1 1 1 2 1
 1 1 1 2 1 1 2 1 1 1 1 2 2 2 1 1 1 1 1 2 1 1 1]
```

```
In [303]: testSet = {'Client_ID':[],'Predicted_Label':[]}
          label_dict = {'1':"Good" , '2':"Bad"}
          counter = 1
          for x in predicted:
              testSet['Client_ID'].append(counter)
              counter+=1
              testSet['Predicted_Label'].append(str(label_dict[str(x)]))
```

```
In [304]: testSetpd = pd.DataFrame(data=testSet)
          testSetcsv=testSetpd.ix[::, ['Client_ID', 'Predicted_Label']]

          testSetpd
```

Out[304]:

	Client_ID	Predicted_Label
0	1	Good
1	2	Bad
2	3	Good
3	4	Good
4	5	Bad
5	6	Good
6	7	Good
7	8	Good
8	9	Good
9	10	Bad
10	11	Bad
11	12	Bad
12	13	Good
13	14	Bad
14	15	Good
15	16	Bad
16	17	Good
17	18	Good
18	19	Bad
19	20	Good
20	21	Good
21	22	Good
22	23	Good
23	24	Good
24	25	Good
25	26	Good
26	27	Good
27	28	Good
28	29	Good
29	30	Bad
..	...	...
770	771	Good
771	772	Bad
772	773	Good
773	774	Good
774	775	Good
775	776	Bad
776	777	Good
777	778	Good
778	779	Good
779	780	Good
780	781	Bad
781	782	Good
782	783	Good
783	784	Bad
784	785	Good
785	786	Good

```
786         787             Good
787         788             Good
788         789              Bad
789         790              Bad
790         791              Bad
791         792             Good
792         793             Good
793         794             Good
794         795             Good
795         796             Good
796         797              Bad
797         798             Good
798         799             Good
799         800             Good

[800 rows x 2 columns]
```

```
In [305]: testSetcsv.to_csv(path_or_buf='testSet_Predictions.csv', sep = '\t')

In [ ]:
```