UNIVERSITY OF LEICESTER

Computational Task 2

For Data Mining and Neural Networks MA4022/ MA7022

Gleb Vorobchuk

Leicester

March 2018

Task №1.
Decision trees predictions (20 marks).

The first step was to replace "y” and “n” answers with 0 and 1 in original dataset.

Number	Year	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12
P	1864	0	0	0	0	1	0	0	1	1	0	0	0
P	1868	1	1	0	0	0	0	1	1	1	0	1	0
P	1872	1	1	0	0	1	0	1	0	0	0	1	0
P	1880	1	0	0	1	0	0	1	1	0	0	0	0
P	1888	0	0	0	0	1	0	0	0	0	0	0	0
P	1900	0	1	0	0	1	0	1	0	0	0	0	1
P	1904	1	1	0	0	1	0	0	0	0	0	1	0
P	1908	1	1	0	0	0	0	0	1	0	0	0	1
P	1916	0	0	0	0	1	0	0	1	0	0	0	0
P	1924	0	1	1	0	1	0	1	1	0	1	0	0
P	1928	1	1	0	0	0	0	1	0	0	0	0	0
P	1936	0	1	0	0	1	1	1	1	0	0	1	0
P	1940	1	1	0	0	1	1	1	1	0	0	1	0
P	1944	1	1	0	0	1	0	1	1	0	0	1	0
P	1948	1	1	1	0	1	0	0	1	0	0	0	0
P	1956	0	1	0	0	1	0	0	0	0	0	1	0
P	1964	0	0	0	0	1	0	1	0	0	0	0	0
P	1972	0	0	0	0	1	0	1	1	1	0	0	0
O	1860	1	0	1	1	0	0	1	0	1	0	0	0
O	1876	1	1	0	1	0	1	0	0	0	1	0	0
O	1884	1	0	0	1	0	0	1	0	1	0	1	0
O	1892	0	0	1	0	1	0	0	1	1	0	0	1
O	1896	0	0	0	1	0	1	0	1	1	0	1	0
O	1912	1	1	1	1	1	0	1	0	0	0	0	0
O	1920	1	0	0	1	0	0	0	1	1	0	0	0
O	1932	1	1	0	0	1	1	0	0	1	0	0	1
O	1952	1	0	0	1	0	0	1	0	0	1	0	1
O	1960	1	1	0	0	0	1	0	0	0	0	0	1
O	1968	1	1	1	1	0	0	1	1	1	0	0	0
O	1976	1	1	0	1	1	0	0	0	0	1	0	0
O	1980	0	0	1	1	1	1	0	0	0	1	0	1

Since we need to use 7 examples, I prepared 7 randomly sampled datasets with 3 P victories and 2 O victories that we will use further to create decision trees.

Here is the example of sampling:

	Number	Year	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12
24	O	1920	1	0	0	1	0	0	0	1	1	0	0	0
25	O	1932	1	1	0	0	1	1	0	0	1	0	0	1
	Number	Year	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12
12	P	1940	1	1	0	0	1	1	1	1	0	0	1	0
13	P	1944	1	1	0	0	1	0	1	1	0	0	1	0
6	P	1904	1	1	0	0	1	0	0	0	0	0	1	0

Table 2. Example of sampling.

After that I've used Decision Tree Classifier from scikit-learn package for python. I created 21 trees: 7 for m=2, 7 for m=3 and 7 for m=4. Default m =2.

Here are the trees structures:

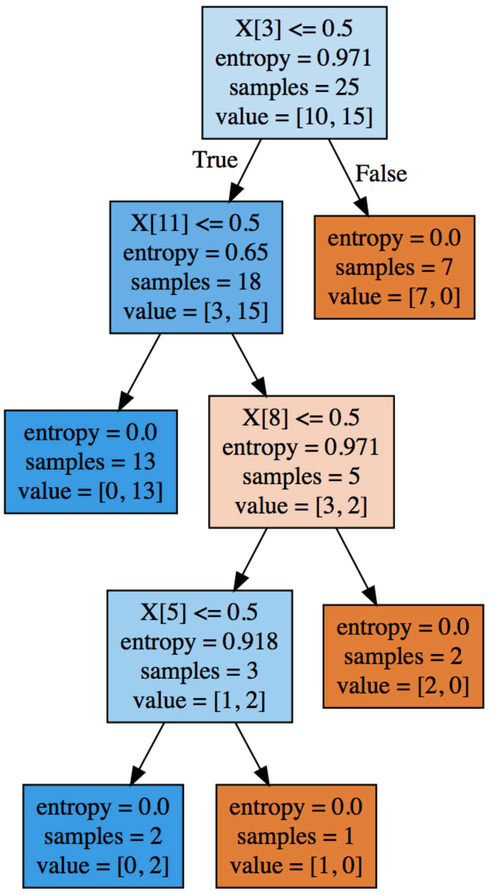


Figure 1. Tree A(Example 1,m=2).

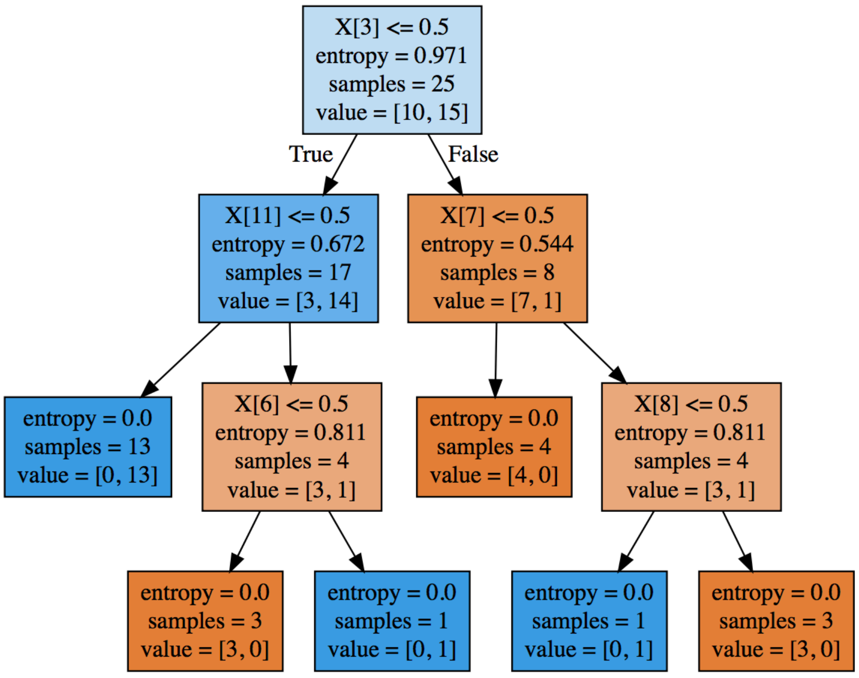


Figure 1.1. Tree B(Example 2,m=2).

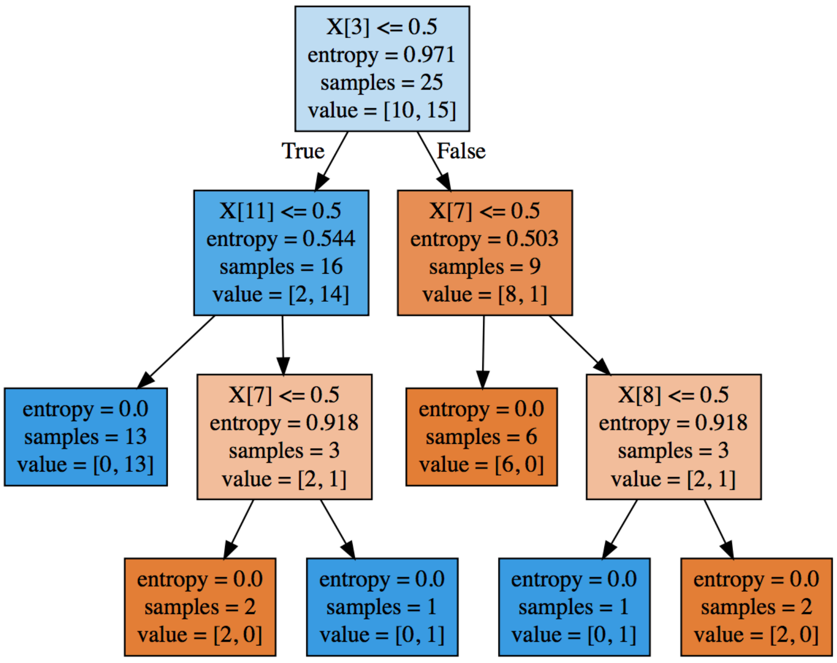


Figure 1.3. Tree C(Example 3,m=2).

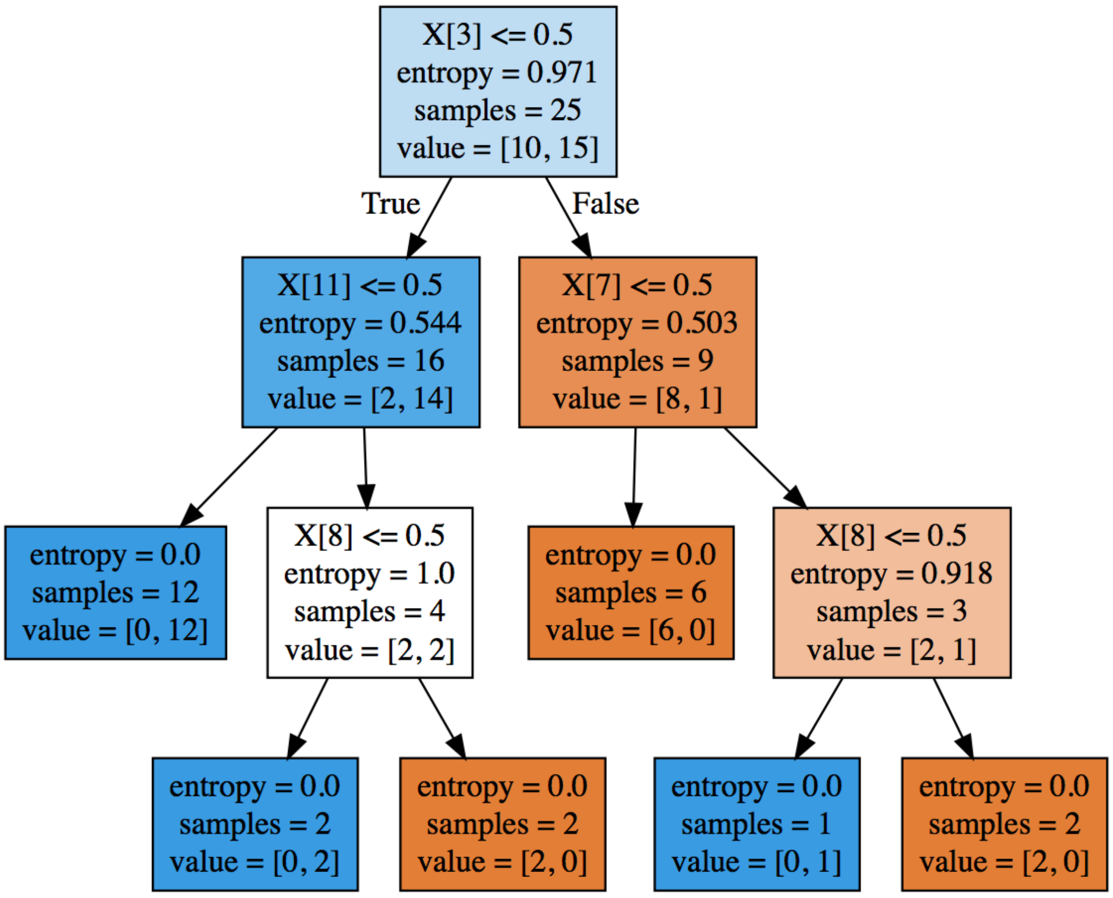


Figure 1.4. Tree D(Example 4,m=2).

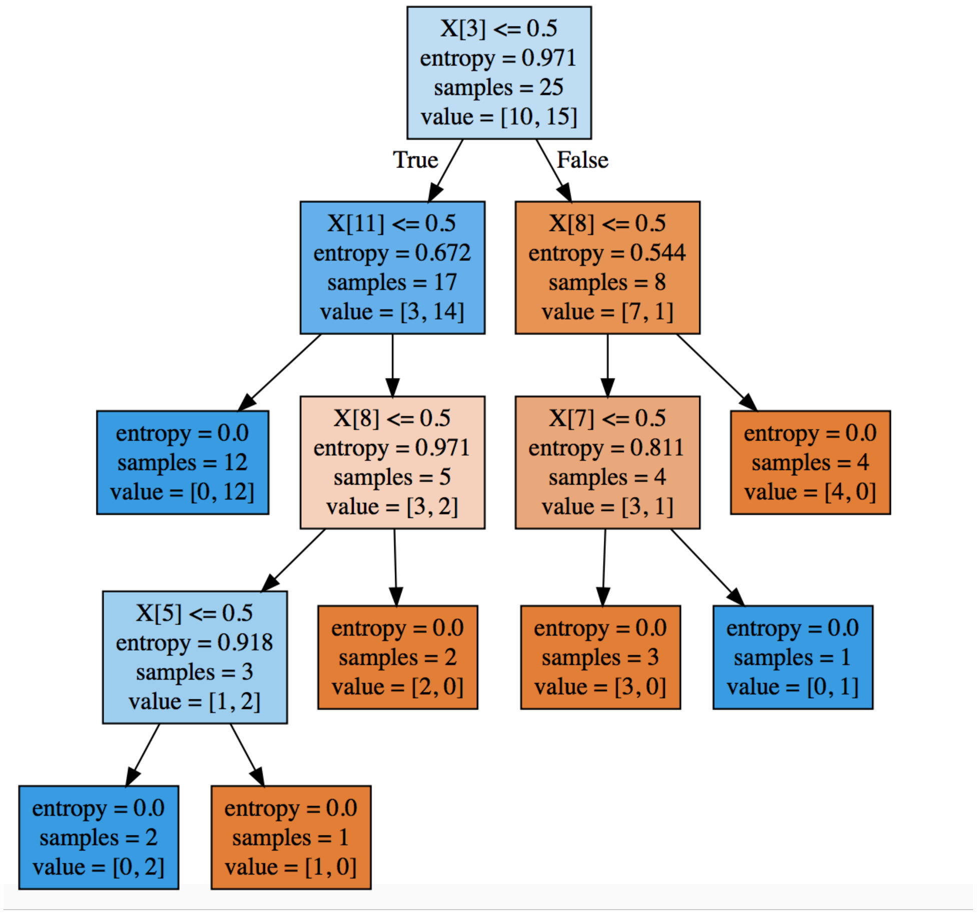


Figure 1.5. Tree E(Example 5,m=2).

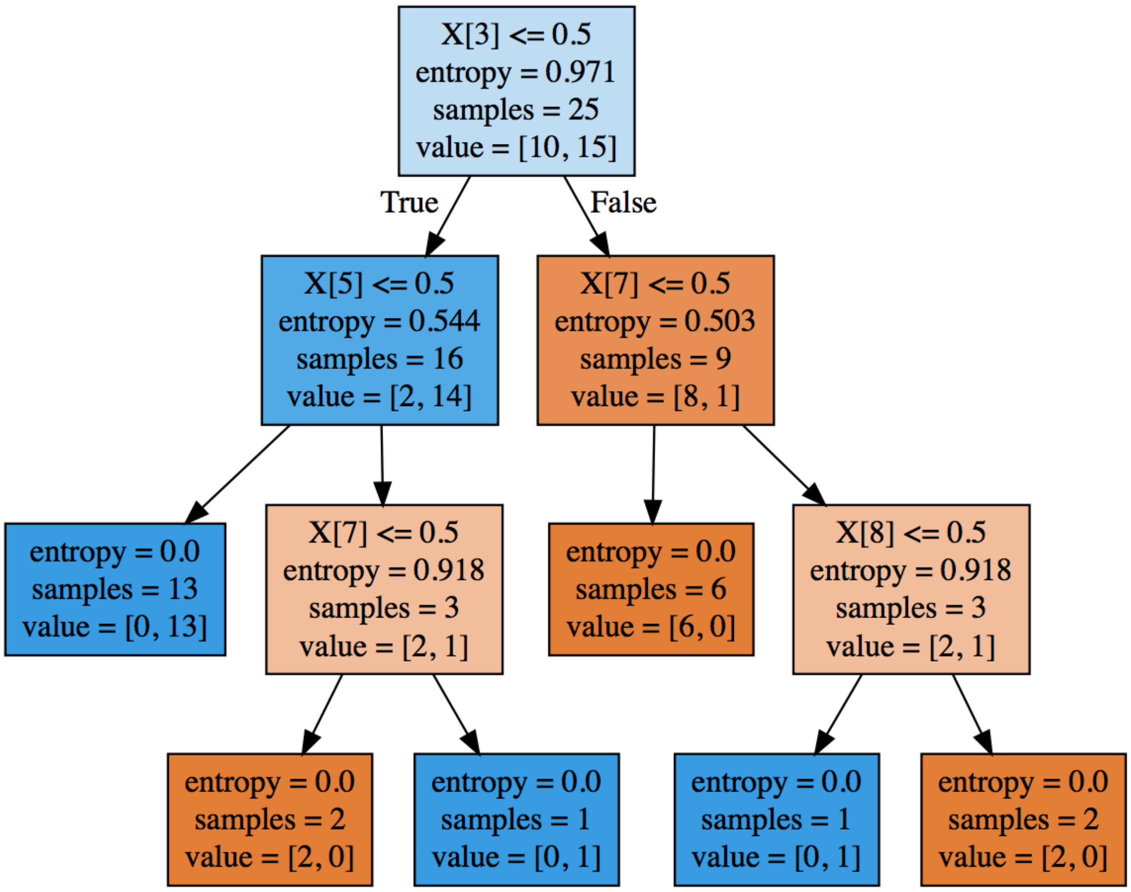


Figure 1.6. Tree F(Example 6,m=2).

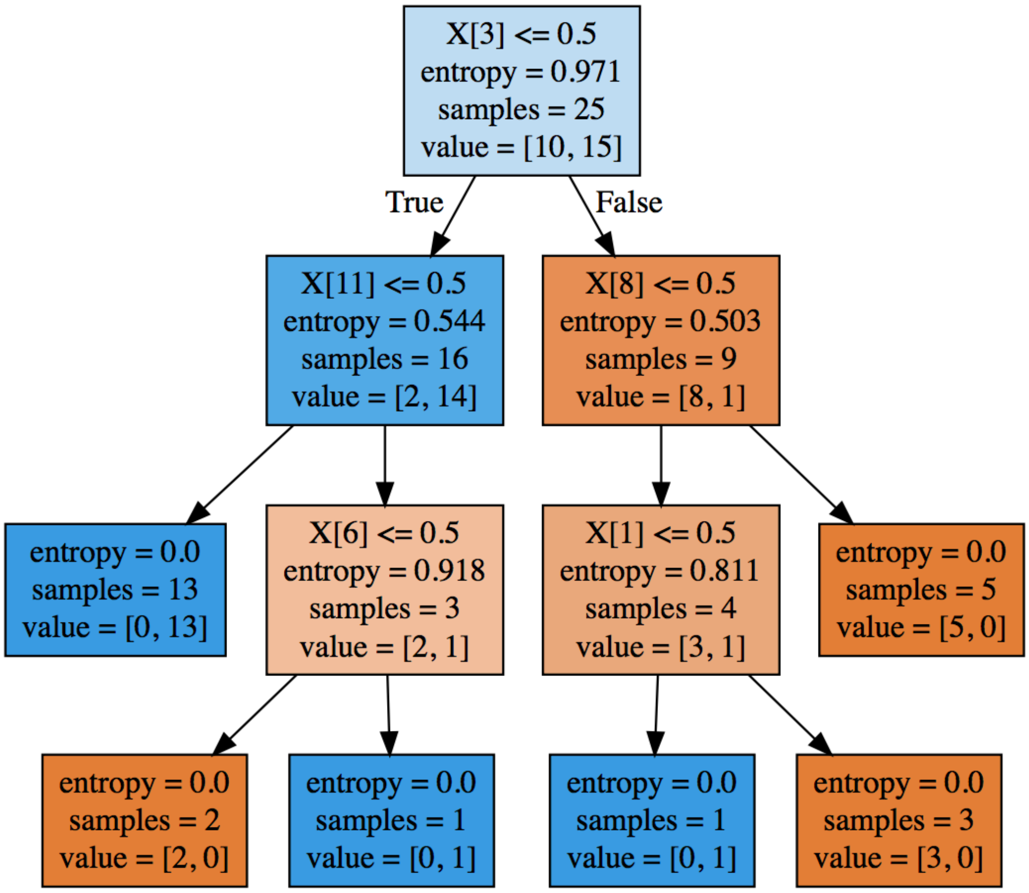


Figure 1.7. Tree G(Example 7,m=2).

*X[i] represents the number of question q=i+1 with nonzero entropy and with minimal number of samples required for splitting.

As we can see the structure of the trees is different. I assume that it happens due to dependance on the attributes entropy for the particular sampling.

But here we can see that the pattern is kinda the same.

First choice is Quesion 4. Second is Quesions 9 and 12. I assume that these two questions are more relevant.

After the training process we need to analyse predictions and prediction accuracy of our decision trees models:

Test 1			Test 2			Test 3			Test 4		
YR	EX	PR	YR	EX	PR	YR	EX	PR	YR	EX	PR
1880	P	O	1916	P	P	1964	P	P	1944	P	P
1936	P	P	1908	P	O	1928	P	P	1964	P	P
1964	P	P	1972	P	P	1900	P	O	1916	P	P
1884	O	O	1976	O	O	1892	O	P	1960	O	P
1976	O	O	1884	O	O	1896	O	O	1920	O	O
test error: 0.2			test error: 0.2			test error: 0.4			test error: 0.2		
train error: 0.0			train error: 0.0			train error: 0.0			train error: 0.0		
Test 5			Test 6			Test 7					
YR	EX	PR	YR	EX	PR	YR	EX	PR			
1864	P	P	1924	P	P	1940	P	P			
1956	P	P	1864	P	P	1908	P	O			
1936	P	P	1936	P	P	1948	P	P			
1860	O	O	1968	O	O	1952	O	P			
1976	O	O	1892	O	P	1960	O	O			
test error: 0.0			test error: 0.2			test error: 0.4					
train error: 0.0			train error: 0.0			train error: 0.0					

Table 3. Prediction for Decision Trees.

Training error is equal to zero for unpruned trees.

Task №2.

Pruning. (15 marks).

Now we want to create pruned trees. We use the same data but vary parameter m which is the minimal number of sample required for splitting.

For the same datasets we create Decision Trees with parameter m = 3,4.

The structure of the trees is almost the same, but now nodes with less than m samples will not be splitted.

Here are the predictions:

Test 1			Test 2			Test 3			Test 4		
YR	EX	PR	YR	EX	PR	YR	EX	PR	YR	EX	PR
1880	P	O	1916	P	P	1964	P	P	1944	P	P
1936	P	P	1908	P	O	1928	P	P	1964	P	P
1964	P	P	1972	P	P	1900	P	O	1916	P	P
1884	O	O	1976	O	O	1892	O	P	1960	O	P
1976	O	O	1884	O	O	1896	O	O	1920	O	O
test error: 0.2			test error: 0.2			test error: 0.4			test error: 0.2		
train error: 0.0			train error: 0.0			train error: 0.0			train error: 0.0		
Test 5			Test 6			Test 7					
YR	EX	PR	YR	EX	PR	YR	EX	PR			
1864	P	P	1924	P	P	1940	P	P			
1956	P	P	1864	P	P	1908	P	O			
1936	P	P	1936	P	P	1948	P	P			
1860	O	O	1968	O	O	1952	O	P			
1976	O	O	1892	O	P	1960	O	O			
test error: 0.0			test error: 0.2			test error: 0.4					
train error: 0.0			train error: 0.0			train error: 0.0					

Table 4. Predictions for m=3

Test 1			Test 2			Test 3			Test 4		
YR	EX	PR	YR	EX	PR	YR	EX	PR	YR	EX	PR
1880	P	O	1916	P	P	1964	P	P	1944	P	P
1936	P	P	1908	P	O	1928	P	P	1964	P	P
1964	P	P	1972	P	P	1900	P	O	1916	P	P
1884	O	O	1976	O	O	1892	O	O	1960	O	P
1976	O	O	1884	O	O	1896	O	O	1920	O	O
test error: 0.2			test error: 0.2			test error: 0.2			test error: 0.2		
train error: 0.04			train error: 0.0			train error: 0.08			train error: 0.04		
Test 5			Test 6			Test 7					
YR	EX	PR	YR	EX	PR	YR	EX	PR			
1864	P	P	1924	P	P	1940	P	P			
1956	P	P	1864	P	P	1908	P	O			
1936	P	P	1936	P	O	1948	P	P			
1860	O	O	1968	O	O	1952	O	O			
1976	O	O	1892	O	P	1960	O	O			
test error: 0.0			test error: 0.4			test error: 0.2					
train error: 0.04			train error: 0.08			train error: 0.04					

Table 5. Predictions for m=4.

Then we need to measure accuracy. In order to do that I averaged the errors of tests for different m's:

M	2	3	4
Learning Error	0.0	0.0	0.0457
Test Error	0.2285	0.2285	0.1999

Table 6. Errors for corresponding m.

Plotting averaged test error against learning error:

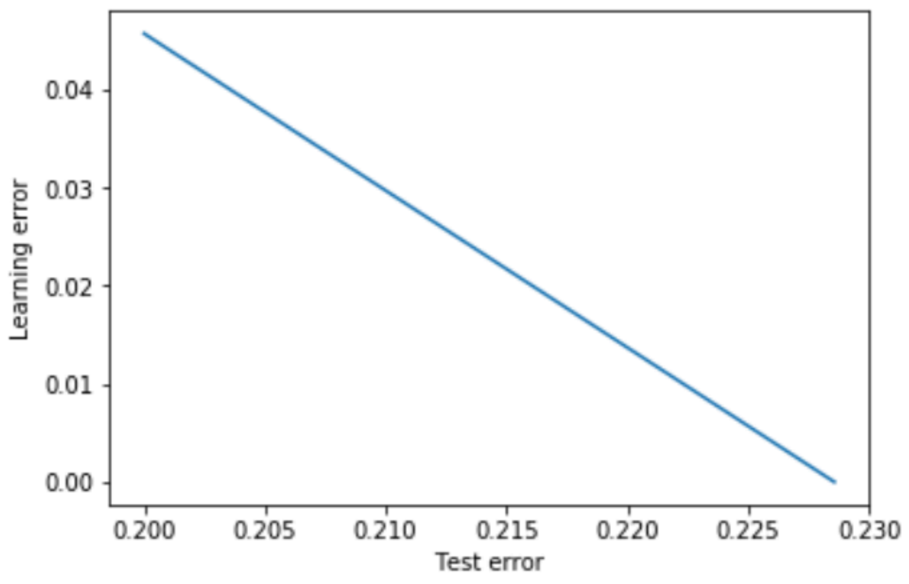


Figure 2.1 Dependency of errors

As we can see learning error increasing with decreasing test error.

Here I'm plotting error against m:

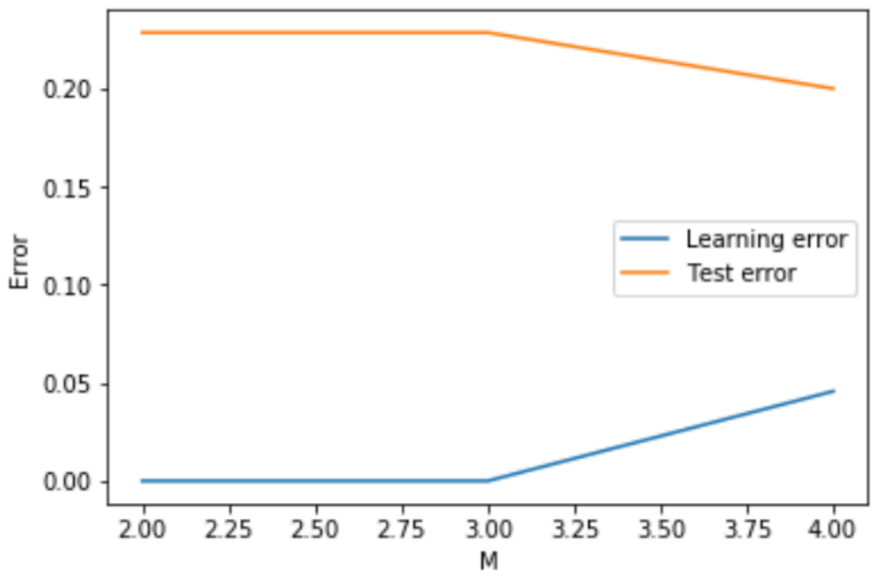


Figure 2.2 Dependency of errors on m

As we can see with increasing m test error is decreasing, so we can make a conclusion that m=4 is the best parameter.

Task №3.

Comparing to kNN (15 marks)

For this task I've performed KNN classification for the same 7 datasets as I've used earlier.

Here are the predictions:

Test 1				Test 2				Test 3				Test 4			
YR	EX	1nn	3nn	YR	EX	1nn	3nn	YR	EX	1nn	3nn	YR	EX	1nn	3nn
1928	P	P	P	1900	P	P	P	1888	P	P	P	190	P	O	O
1868	P	P	P	1940	P	P	P	1944	P	P	P	190	P	P	P
1904	P	P	P	1872	P	P	P	1940	P	P	P	194	P	P	P
1976	O	O	O	1896	O	O	O	1976	O	O	O	186	O	O	O
1980	O	P	P	1892	O	P	P	1952	O	P	P	197	O	O	O
1nn error: 0.2				1nn error: 0.2				1nn error: 0.2				1nn error: 0.2			
3nn error: 0.2				3nn error: 0.2				3nn error: 0.2				3nn error: 0.2			
Test 5				Test 6				Test 7							
YR	EX	1nn	3nn	YR	EX	1nn	3nn	YR	EX	1nn	3nn				
1908	P	O	O	1972	P	P	P	1972	P	P	P				
1880	P	P	P	1864	P	P	P	1880	P	O	O				
1964	P	P	P	1900	P	P	P	1948	P	P	P				
1892	O	P	P	1884	O	O	O	1932	O	O	O				
1920	O	P	P	1968	O	O	O	1892	O	P	P				
1nn error: 0.6				1nn error: 0.0				1nn error: 0.4							
3nn error: 0.6				3nn error: 0.0				3nn error: 0.4							

Table 7. KNN predictions.

Then I'm averaging the errors for 1nn and 3nn and comparing to Decision Tree with m=4:

`1-NN Error`	`25.71%`
`3-NN Error`	`25.71%`
`DT Error(m=4)`	`19.99%`

Table 8. KNN and DT comparison.

As we can see Decision tree for m=4 gives us better prediction for this particular series of datasets.

Task №4.

Linear separability (15 marks)

I’ve solved this problem using linear Fisher’s discriminant and Rosenbladt’s Perceptron.

Here are the results:

	Fisher’s	Perceptron
Error	12.9%	9.67%
Accuracy	87.09%	90.32%

Table 9. Fisher’l LD and Perceptron comparison.

As we can see Perceptron produce better accuracy.

Task №5.

Clustering (15 marks).

In order to perform this task I've used PCA for dimension reduction.

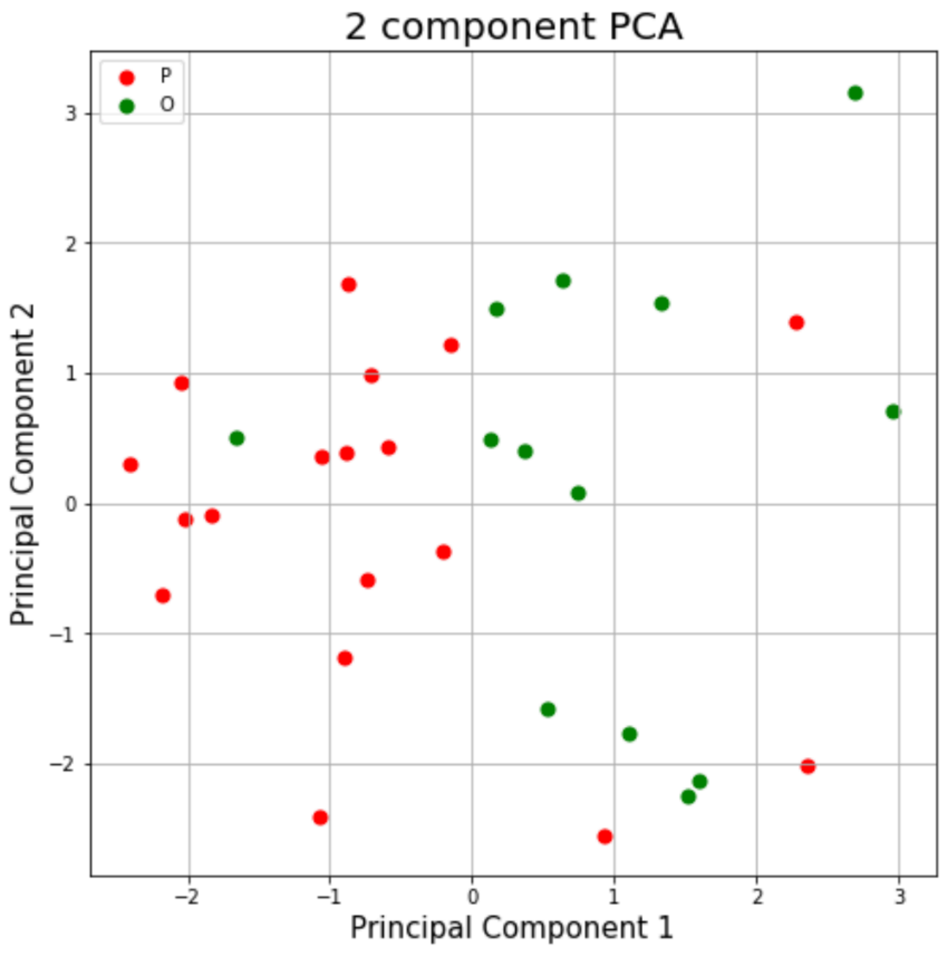
First of all I'm Scaling the data. Then I'm applying PCA:

	principal component 1	principal component 2	Number
0	-0.733624	-0.592749	P
1	-1.072247	-2.406507	P
2	-1.838161	-0.088198	P
3	-0.707474	0.985254	P
4	-0.867994	1.683181	P
5	-1.053729	0.363474	P
6	-0.152601	1.223934	P
7	-0.204854	-0.374043	P
8	-2.409987	0.297885	P
9	-2.027308	-0.118625	P
10	-2.184416	-0.709978	P
11	-0.586911	0.433986	P
12	-2.048256	0.921038	P
13	-0.880058	0.392528	P
14	-0.906208	-1.185475	P
15	2.358328	-2.010808	P
16	2.27427	1.389127	P
17	0.933406	-2.562047	P
18	0.361657	0.404436	O
19	0.534164	-1.583237	O
20	0.746619	0.075966	O
21	1.600978	-2.13664	O
22	0.164481	1.494529	O
23	2.95572	0.715209	O
24	0.636441	1.710445	O
25	1.512533	-2.242343	O
26	0.133078	0.49731	O

27	1.108289	-1.773144	O
28	-1.665577	0.504528	O
29	1.325097	1.53808	O
30	2.694347	3.152883	O

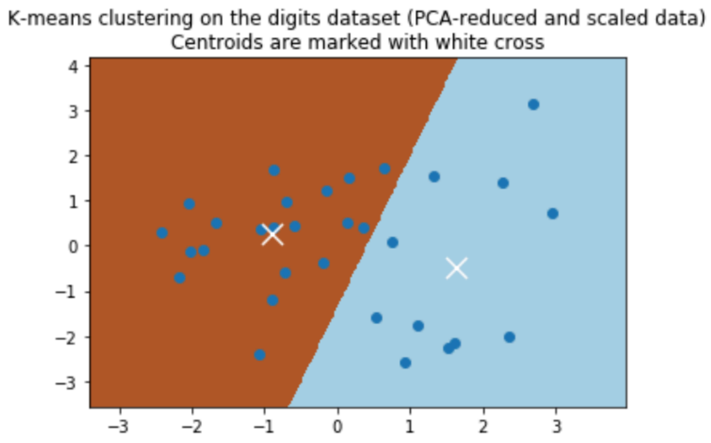
Table 10. PCA components.

Here is the projection of 12-D dataset on 2-D



After that I've performed 2-Mean clustering[1]:

```
n_classes: 2,     n_samples 31,    n_features 12
___________________________________________________________________________________
init              time     inertia  homo     compl    v-meas   ARI      AMI      silhouette
k-means++         0.01s    318      0.721    0.708    0.714    0.750    0.700    0.131
random            0.01s    317      0.467    0.458    0.462    0.535    0.444    0.204
PCA-based         0.00s    319      0.161    0.168    0.164    0.209    0.139    0.019
___________________________________________________________________________________
```



Here we can see two clusters:

A clustering result satisfies homogeneity because all of its clusters contain only data points which are members of a single class.

This metric is independent of the absolute values of the labels: a permutation of the class or cluster label values won't change the score value in any way.[2]

A clustering result satisfies completeness because all the data points that are members of a given class are elements of the same cluster.

This metric is independent of the absolute values of the labels: a permutation of the class or cluster label values won't change the score value in any way.[3]

The Silhouette Coefficient is calculated using the mean intra-cluster distance (`a`) and the mean nearest-cluster distance (`b`) for each sample. The Silhouette Coefficient for a sample is `(b - a) / max(a, b)`. To clarify, `b` is the distance between a sample and the nearest

[1] This part of the code based on demo. http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html#sphx-glr-auto-examples-cluster-plot-kmeans-digits-py

[2] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.homogeneity_score.html#sklearn.metrics.homogeneity_score

[3] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.completeness_score.html#sklearn.metrics.completeness_score

cluster that the sample is not a part of. Note that Silhouette Coefficient is only defined if number of labels is 2 <= n_labels <= n_samples - 1.

This function returns the mean Silhouette Coefficient over all samples. The best value is 1 and the worst value is -1. Values near 0 indicate overlapping clusters. Negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar.[4]

The V-measure is the harmonic mean between homogeneity and completeness[5]

The adjusted Rand index is thus ensured to have a value close to 0.0 for random labeling independently of the number of clusters and samples and exactly 1.0 when the clusterings are identical (up to a permutation).[6]

Adjusted Mutual Information (AMI) is an adjustment of the Mutual Information (MI) score to account for chance. It accounts for the fact that the MI is generally higher for two clusterings with a larger number of clusters, regardless of whether there is actually more information shared. This metric is independent of the absolute values of the labels: a permutation of the class or cluster label values won't change the score value in any way[7]

[4] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html#sklearn.metrics.silhouette_score

[5] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.v_measure_score.html#sklearn.metrics.v_measure_score

[6] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html#sklearn.metrics.adjusted_rand_score

[7] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_mutual_info_score.html#sklearn.metrics.adjusted_mutual_info_score

Task №6.

Trump election (10 marks).

Here we need to answer the questions in terms of Trump election.

Question	Answer
Has the P-party been in power for more than one term?,	yes
2) Did the P-party receive more than 50% of the popular vote in the last election?,	yes
3) Was there significant activity of a third party during the election year?,	no
4) Was there serious competition in the P-party primaries?,	yes
5) Was the P-party candidate the president at the time of the election?,	no
6) Was there a depression or recession in the election year?,	yes
7) Was there a growth in the gross national product of more than 2.1% in the year of the election?,	yes
8) Did the P-party president make any substantial political changes during his term?	yes
9) Did significant social tension exist during the term of the P-party?,	yes
10) Was the P-party administration guilty of any serious mistakes or scandals?,	yes
11) Was the P-party candidate a national hero?,	yes
12) Was the O-party candidate a national hero?,	no

Table 11. Answers in my understanding of US political situation at the elections time.

Then I've used complete dataset as the training set and use this answers to predict the winner.

Here I have following results according to that P - Hillary wins, O - Trump wins:

Classifier	Prediction
DT	Trump
1-NN	Hillary
3-NN	Hillary
Fisher's	Trump
Perceptron	Trump

Appendix.

Python code(I'm recommend to use Comp_Task_2.ipynb:

```
# coding: utf-8


# In[454]:



get_ipython().magic('matplotlib inline')


import matplotlib.pyplot as plt
import math as math
from scipy import stats
import numpy as np
import pandas as pd
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
from sklearn.tree import export_graphviz
import graphviz
from IPython.display import Image
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA




def dataset():
    #SELECT THE DATA
    EL_df = pd.read_csv('/Users/glebvorobcuk/Desktop/ACNM 2018/Term 2/Data Mining/Comp. Task 2/EL_df.csv')
    #print(EL_df)
    #select random values from each class and remove it from training set
    TRAINING_df=EL_df
    TEST_df=pd.DataFrame()
    P_rand_tmp=pd.DataFrame()
    O_rand_tmp=pd.DataFrame()
    P_rand=pd.DataFrame()
    O_rand=pd.DataFrame()
    for i in range(0,3):
        P_rand_tmp = pd.DataFrame(EL_df.loc[EL_df['Number'].isin(["P"])].sample(n=1))
        P_rand=P_rand.append(P_rand_tmp)
        TRAINING_df.drop(P_rand_tmp.index,inplace=True)
    for j in range(0,2):
        O_rand_tmp = pd.DataFrame(EL_df.loc[EL_df['Number'].isin(["O"])].sample(n=1))
        O_rand = O_rand.append(O_rand_tmp)
        TRAINING_df.drop(O_rand_tmp.index,inplace=True)

    TEST_df=TEST_df.append(P_rand)
    TEST_df=TEST_df.append(O_rand)
    RETURN=[TRAINING_df,TEST_df]
    RETURN_df=pd.concat(RETURN)

    return RETURN_df

#m=2 by default
def DT(dataset, m):
    #print(dataset)
    TRAINING_df=dataset.iloc[0:25,:]
    TEST_df=dataset.iloc[26:,:]



    #target attribute
    X_train = TRAINING_df.loc[:,'Q1':'Q12']
    y_train = TRAINING_df.Number
    year_train=TRAINING_df.Year
    X_test = TEST_df.loc[:,'Q1':'Q12']
    y_test = TEST_df.Number
    year_test=TEST_df.Year

    #copy dataset to prevent changes
    X_train_knn = X_train
    X_test_knn = X_test
    y_train_knn = y_train
    y_test_knn = y_test

   # print(TRAINING_df)
    #print(TEST_df)
    #create decision tree
    clf_entropy = DecisionTreeClassifier(criterion = "entropy", random_state =
100,min_samples_split=m)
    clf_entropy.fit(X_train, y_train)
    dot_data = tree.export_graphviz(clf_entropy, out_file=None,filled = True)
    graph = graphviz.Source(dot_data)
    graph

    #create 1-nn and 3-nn classifiers



    #predict
        #test
    pred_test = clf_entropy.predict(X_test)
    pred_test = pd.DataFrame(pred_test,index=year_test.index)
    pred_test_ = pd.concat([year_test,y_test,pred_test], axis=1)
    pred_test_.columns=['Year','Exact','Predicted']
    print('m:',m)
    print(pred_test_)
    err_test =1 - accuracy_score(y_test, pred_test)
    print('test error:',err_test)
```

```
     #train
    pred_tr = clf_entropy.predict(X_train)
    pred_tr = pd.DataFrame(pred_tr,index=year_train.index)
    pred_tr_ = pd.concat([year_train,y_train,pred_tr], axis=1)
    pred_tr_.columns=['Year','Exact','Predicted']
    err_tr =1 - accuracy_score(y_train, pred_tr)
    print('train error:',err_tr)




    #return trees with predicrions and accuracy scores accuracy scores
    return [graph,
            pred_test_,
            err_test,
            pred_tr_,
            err_tr,
            m]



def knn(dataset):
    TRAINING_df=dataset.iloc[0:25,:]
    TEST_df=dataset.iloc[26:,:]
    #target attribute
    X_train = TRAINING_df.loc[:,'Q1':'Q12']
    y_train = TRAINING_df.Number
    year_train=TRAINING_df.Year
    X_test = TEST_df.loc[:,'Q1':'Q12']
    y_test = TEST_df.Number
    year_test=TEST_df.Year
```

```
#copy dataset to prevent changes
X_train_knn = X_train
X_test_knn = X_test
y_train_knn = y_train
y_test_knn = y_test



#knn
knn1 = KNeighborsClassifier(n_neighbors=1)
knn1.fit(X_train,y_train)
pred_test = knn1.predict(X_test)
pred_test = pd.DataFrame(pred_test,index=year_test.index)
pred_test_ = pd.concat([year_test,y_test,pred_test], axis=1)
pred_test_.columns=['Year','Exact','Predicted']
err_test1 =1 - accuracy_score(y_test, pred_test)

knn3 = KNeighborsClassifier(n_neighbors=3)
knn3.fit(X_train,y_train)
pred_test2 = knn3.predict(X_test)
pred_test2 = pd.DataFrame(pred_test,index=year_test.index)
pred_test2_ = pd.concat([pred_test_,pred_test], axis=1)
pred_test2_.columns=['Year','Exact','1nn Predicted','3nn Predicted']
print(pred_test2_)
err_test2 =1 - accuracy_score(y_test, pred_test2)
print('1nn error:',err_test1)
print('3nn error:',err_test2)



return [pred_test2_,err_test1,err_test2]
```

```
#random sampling
dataset_1=dataset()
dataset_2=dataset()
dataset_3=dataset()
dataset_4=dataset()
dataset_5=dataset()
dataset_6=dataset()
dataset_7=dataset()
#preparing trees




#knn for best m for 7 datasets?



# In[438]:




m=2
A=DT(dataset_1,m)
B=DT(dataset_2,m)
C=DT(dataset_3,m)
D=DT(dataset_4,m)
E=DT(dataset_5,m)
F=DT(dataset_6,m)
G=DT(dataset_7,m)

m=3
H=DT(dataset_1,m)
I=DT(dataset_2,m)
```

```
J=DT(dataset_3,m)
K=DT(dataset_4,m)
L=DT(dataset_5,m)
M=DT(dataset_6,m)
N=DT(dataset_7,m)

m=4
O=DT(dataset_1,m)
P=DT(dataset_2,m)
Q=DT(dataset_3,m)
R=DT(dataset_4,m)
S=DT(dataset_5,m)
T=DT(dataset_6,m)
U=DT(dataset_7,m)

#calculate average errors for different m
AVG_TE_m2=np.mean([A[2],B[2],C[2],D[2],E[2],F[2],G[2]])
AVG_TE_m3=np.mean([H[2],I[2],J[2],K[2],L[2],M[2],N[2]])
AVG_TE_m4=np.mean([O[2],P[2],Q[2],R[2],S[2],T[2],U[2]])

AVG_LE_m2=np.mean([A[4],B[4],C[4],D[4],E[4],F[4],G[4]])
AVG_LE_m3=np.mean([H[4],I[4],J[4],K[4],L[4],M[4],N[4]])
AVG_LE_m4=np.mean([O[4],P[4],Q[4],R[4],S[4],T[4],U[4]])




# In[455]:


G[0]


# In[467]:
```

```
knn1=knn(dataset_1)
knn2=knn(dataset_2)
knn3=knn(dataset_3)
knn4=knn(dataset_4)
knn5=knn(dataset_5)
knn6=knn(dataset_6)
knn7=knn(dataset_7)
AVG_1NN=np.mean([knn1[1],knn2[1],knn3[1],knn4[1],knn5[1],knn6[1],knn7[1]])
AVG_3NN=np.mean([knn1[2],knn2[2],knn3[2],knn4[2],knn5[2],knn6[2],knn7[2]])
print(AVG_1NN,AVG_3NN)

AVG_1NN=np.mean([0.2,0.2,0.2,0.2,0.6,0.0,0.4])
AVG_3NN=np.mean([0.2,0.2,0.2,0.2,0.6,0.0,0.4])
print(AVG_1NN,AVG_3NN)


# In[465]:


#learning curves and best m selection
plt.plot([AVG_TE_m2,AVG_TE_m3,AVG_TE_m4],
[AVG_LE_m2,AVG_LE_m3,AVG_LE_m4])
plt.ylabel("Learning error")
plt.xlabel("Test error")
plt.show
print([AVG_LE_m2,AVG_LE_m3,AVG_LE_m4])
print([AVG_TE_m2,AVG_TE_m3,AVG_TE_m4])


# In[466]:
```

```
plt.plot([2,3,4],[AVG_LE_m2,AVG_LE_m3,AVG_LE_m4])
plt.plot([2,3,4],[AVG_TE_m2,AVG_TE_m3,AVG_TE_m4])
plt.xlabel("M")
plt.ylabel("Error")
plt.legend(['Learning error','Test error'])
plt.show
print('le',[AVG_LE_m2,AVG_LE_m3,AVG_LE_m4])
print('te',[AVG_TE_m2,AVG_TE_m3,AVG_TE_m4])


# In[477]:


#lda


EL_df = pd.read_csv('/Users/glebvorobcuk/Desktop/ACNM 2018/Term 2/Data Mining/
Comp. Task 2/EL_df.csv')
X = EL_df.loc[:,'Q1':'Q12']
y = EL_df.Number
Fisher = LDA()
Fisher.fit(x, y)
print('accuracy:',Fisher.score(x, y, sample_weight=None)*100,'%','\nerror:',(1-Fisher.score(x, y,
sample_weight=None))*100,'%')

Perceptron = PCP()
Perceptron.fit(x, y)
print('accuracy:',Perceptron.score(x, y, sample_weight=None)*100,'%','\nerror:',(1-
Perceptron.score(x, y, sample_weight=None))*100,'%')


# In[398]:
```

UNIVERSITY OF LEICESTER

```
# In[399]:



#perceptron


from sklearn.linear_model import Perceptron as PCP
def pcp(dataset):
    TRAINING_df=dataset.iloc[0:25,:]
    TEST_df=dataset.iloc[26:,:]
    #target attribute
    X_train = TRAINING_df.loc[:,'Q1':'Q12']
    y_train = TRAINING_df.Number
    year_train=TRAINING_df.Year
    X_test = TEST_df.loc[:,'Q1':'Q12']
    y_test = TEST_df.Number
    year_test=TEST_df.Year

    Perceptron = PCP()
    #training
    Perceptron.fit(X_train, y_train)
    pred5= Perceptron.predict(X_train)
    pred5_tr = pd.DataFrame(pred5,index=year_train.index)
    pred5_tr_= pd.concat([year_train,y_train,pred5_tr], axis=1)
    pred5_tr_.columns=['Year','Exact','Predicted']
    print(pred5_tr_)
    print (1-accuracy_score(y_train, pred5))
    print(Perceptron.coef_ )
    #test
    pred5= Perceptron.predict(X_test)
    pred5_tr = pd.DataFrame(pred5,index=year_test.index)
    pred5_tr_= pd.concat([year_test,y_test,pred5_tr], axis=1)
    pred5_tr_.columns=['Year','Exact','Predicted']
    print(pred5_tr_)
    print (1-accuracy_score(y_test, pred5))
    print(Perceptron.coef_ )

pcp(dataset_1)
pcp(dataset_2)
pcp(dataset_3)
pcp(dataset_4)
pcp(dataset_5)
pcp(dataset_6)
pcp(dataset_7)



# In[474]:



#k-means clustering
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Separating out the features
X = dataset_1.loc[:,'Q1':'Q12']
# Separating out the target
y = dataset_1.Number
# Standardizing the features
x = StandardScaler().fit_transform(X)
```

```
#reduce dimetionality
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
finalDf = pd.concat([principalDf, dataset_1.Number], axis = 1)
print(finalDf)
print(pca.explained_variance_ratio_)
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = ['P', 'O']
colors = ['r', 'g']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['Number'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()


# In[475]:


print(__doc__)

from time import time
import numpy as np
import matplotlib.pyplot as plt

from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale

np.random.seed(42)

data = dataset_1.loc[:,'Q1':'Q12']
#scaling
data = StandardScaler().fit_transform(X)
from sklearn.datasets import load_digits
digits = load_digits()
labels = dataset_1.Number
n_samples=31
n_features = 12
n_digits = len(np.unique(y))
sample_size = 11

print("n_classes: %d, \t n_samples %d, \t n_features %d"
      % (n_digits, n_samples, n_features))



print(82 * '_')
print('init\t\ttime\tinertia\thomo\tcompl\tv-meas\tARI\tAMI\tsilhouette')



def bench_k_means(estimator, name, data):
    t0 = time()
    estimator.fit(x)
    print('%-9s\t%.2fs\t%i\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f'
          % (name, (time() - t0), estimator.inertia_,
             metrics.homogeneity_score(labels, estimator.labels_),
             metrics.completeness_score(labels, estimator.labels_),
             metrics.v_measure_score(labels, estimator.labels_),
             metrics.adjusted_rand_score(labels, estimator.labels_),
             metrics.adjusted_mutual_info_score(labels,  estimator.labels_),
             metrics.silhouette_score(data, estimator.labels_,
                                      metric='euclidean',
                                      sample_size=sample_size)))

bench_k_means(KMeans(init='k-means++', n_clusters=n_digits, n_init=10),
              name="k-means++", data=data)

bench_k_means(KMeans(init='random', n_clusters=n_digits, n_init=10),
              name="random", data=data)

# in this case the seeding of the centers is deterministic, hence we run the
# kmeans algorithm only once with n_init=1
pca = PCA(n_components=n_digits).fit(data)
bench_k_means(KMeans(init=pca.components_, n_clusters=n_digits, n_init=1),
              name="PCA-based",
              data=data)
print(82 * '_')

#
#################################################################
######################
# Visualize the results on PCA-reduced data

reduced_data = PCA(n_components=2).fit_transform(data)
kmeans = KMeans(init='k-means++', n_clusters=n_digits, n_init=10)
kmeans.fit(reduced_data)

# Step size of the mesh. Decrease to increase the quality of the VQ.
h = .02     # point in the mesh [x_min, x_max]x[y_min, y_max].
```

```
# Plot the decision boundary. For that, we will assign a color to each
x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:, 0].max() + 1
y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Obtain labels for each point in mesh. Use last trained model.
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1)
plt.clf()
plt.imshow(Z, interpolation='nearest',
           extent=(xx.min(), xx.max(), yy.min(), yy.max()),
           cmap=plt.cm.Paired,
           aspect='auto', origin='lower')

plt.scatter(reduced_data[:, 0], reduced_data[:, 1])
# Plot the centroids as a white X
centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1],
            marker='x', s=169, linewidths=5,
            color='w', zorder=10)
plt.title('K-means clustering on the digits dataset (PCA-reduced and scaled data)\n'
          'Centroids are marked with white cross')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.show()



# In[452]:
```

```
# In[483]:


#trump election
EL_df = pd.read_csv('/Users/glebvorobcuk/Desktop/ACNM 2018/Term 2/Data Mining/
Comp. Task 2/EL_df.csv')
EL_y=pd.DataFrame(EL_df.Number)
EL_x=pd.DataFrame(EL_df.loc[:,'Q1':'Q12'])
EL_y.Number.replace(['P'], 1,inplace=True)
EL_y.Number.replace(['O'], 0,inplace=True)
DT_EL=DecisionTreeClassifier(criterion = "entropy", random_state = 100)
Perceptron = PCP()
Fisher = LDA()
knn1 = KNeighborsClassifier(n_neighbors=1)
knn3 = KNeighborsClassifier(n_neighbors=3)
print(EL_y)
DT_EL.fit(EL_x, EL_y)
knn1.fit(X_train,y_train)
knn3.fit(X_train,y_train)
Perceptron.fit(EL_x, EL_y)
Fisher.fit(EL_x, EL_y)
XX=[[1,1,0,1,0,1,1,1,1,1,1,0]]
Election_DT = DT_EL.predict(XX)
Election_knn1 = knn1.predict(XX)
Election_knn3 = knn1.predict(XX)
Election_Perceptron = Perceptron.predict(XX)
Election_PFisher =Fisher.predict(XX)
print(Election_DT)
print(Election_knn1)
print(Election_knn3)
print(Election_Perceptron)
print(Election_PFisher)

#answer question
```