Introduction

The purpose of this task is to introduce us the Logistic Regression. Our aim is to understand how to use Logistic Regression for clasiffication when we have a binary response variable or a multiclass response variable

We are going to use one dataset from Machine Learning Repository (https://archive.ics.uci.edu/ml/index.php). You can find the dataset in the link below. https://archive.ics.uci.edu/ml/datasets/wine.

Specifically there are two datasets one for the red wines and one the white wines, but we are going to use only the dataset from the red wines as the purpose of this essay is to understand better the logistic regression and how we can apply it for clasiffication. The proper analysis that someone can do is to do a statistical analysis for the two datasets (red wines and white wines) separately and then compare the results and extract some deduction about the quality of wines.

These datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are many more normal wines than excellent or poor ones). Outlier detection algorithms could be used to detect the few excellent or poor wines. Also, we are not sure if all input variables are relevant. So it could be interesting to test feature selection methods.

Citation Request:

Please include this citation if you plan to use this database:

P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis.Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

(https://archive.ics.uci.edu/ml/datasets/wine+Quality)

(https://www.vinhoverde.pt/en/)

Summary

The purpose of this essay is to build a model that can predict whether the quality of red wine is good or bad. As we are going to see later on the response variable which we want to predict is quality.

We will deal with this query with two different approaches:

1. We face the response variable as a multiclass.
2. We face the response variable as a binary.

In first case we are going to use a multiclass logistic regression and in the second one the logistic regression.

Chapter 1 Insert and explore the dataset

1.1 Import libraries

First thing first, we have to upload some necessary libraries to insert our data and after that to visualise the main information about it.

```
# Load packages and check versions
import sys
import numpy as np
import pandas as pd
import matplotlib as mpl
import sklearn
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive
```

```
df=pd.read_excel("/content/gdrive/MyDrive/wine/winequality-red_telos.xlsx")
```

1.2 Descriptive statistics

As we can see all the variables are numbers.

```
df.shape
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed_acidity         1599 non-null   float64
 1   volatile_acidity      1599 non-null   float64
 2   citric_acid           1599 non-null   float64
 3   residual_sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur_dioxide   1599 non-null   float64
 6   total_sulfur_dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
```

```
 9   sulphates            1599 non-null   float64
 10  alcohol              1599 non-null   float64
 11  quality              1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In sequence we have to check if we have any Na values.

```
#We dont have na values
df.isnull().sum()
```

```
fixed_acidity           0
volatile_acidity        0
citric_acid             0
residual_sugar          0
chlorides               0
free sulfur_dioxide     0
total_sulfur_dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

Since it is not necessary to do data cleaning we are going to visualise the distribution of each variable. We need to upload the widgets library to make the plots interactive.

The aim of the descriptive statistic is to identify whether a variable has a outliers or not and how it's values are distributed.

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")

from IPython.display import IFrame
documentation = IFrame(src='https://ipywidgets.readthedocs.io/en/latest/examples/Widget%2(
                       width=1000,
                       height=400)
display(documentation)

import ipywidgets
```

» Widget List

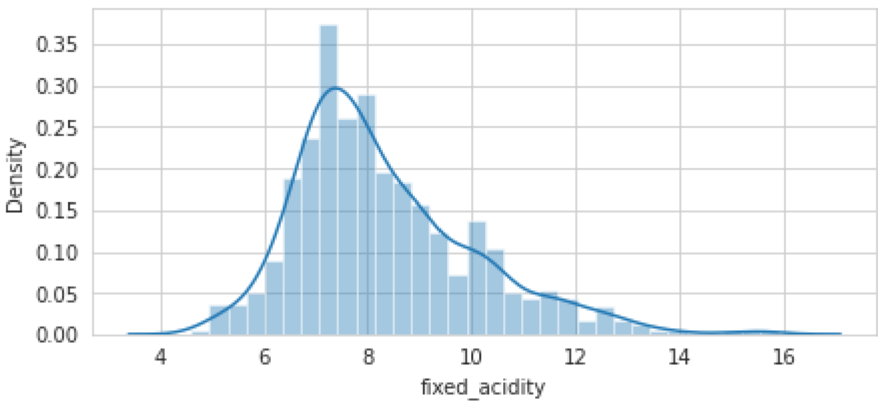
This page was generated from examples/Widget List.ipynb.
Interactive online version: launch binder.

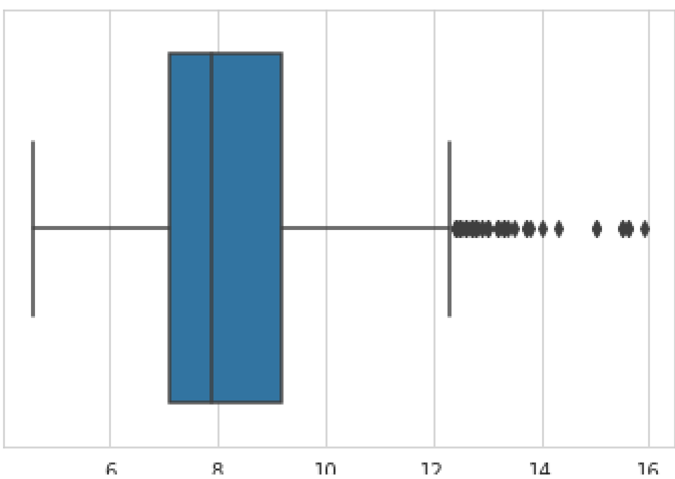
Widget List

```
@ipywidgets.interact
def plot(col=df.columns):                                   # numerical variabl
    f, ax = plt.subplots(figsize=(7, 3))
    ax = sns.distplot(df[col])                              # visualize the dis
    plt.show();
    ax = sns.boxplot(x=df[col])                             # detect outliers
    plt.show();
```

col fixed_acidity

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```





1.3 Correlation, F-statistic

Before we proceed to build a model we have to chech if we have multicolinearity among the variables. Multicolinearity is when two variables are highly correlated.

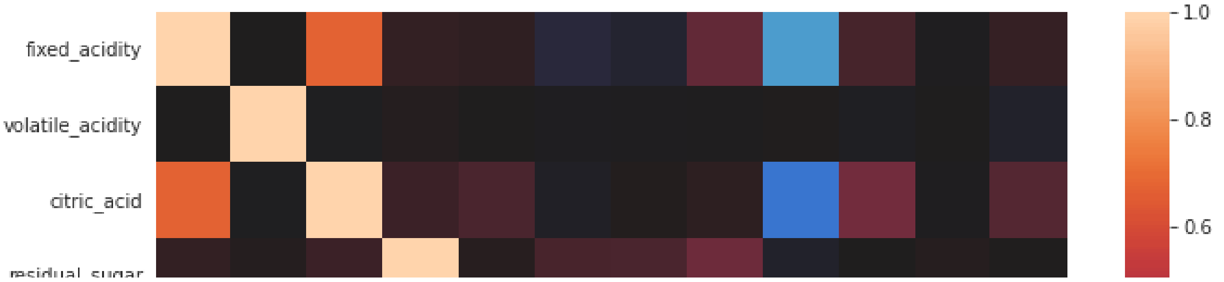
We will apply a heatmap to see which variables are correllated

```
corr=df.corr()
corr.iloc[0:5,0:5]
#five rows an d five columns
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlori
fixed_acidity	1.000000	-0.001996	0.671703	0.114777	0.093
volatile_acidity	-0.001996	1.000000	-0.033493	0.044796	0.000
citric_acid	0.671703	-0.033493	1.000000	0.143577	0.203
residual_sugar	0.114777	0.044796	0.143577	1.000000	0.055
chlorides	0.093705	0.000546	0.203823	0.055610	1.000

```
f, ax = plt.subplots(figsize=(11, 9))
sns.heatmap(corr,
xticklabels=corr.columns.values,
yticklabels=corr.columns.values,
center=0)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa71eb11190>
```



From the correlation matric above we didn't see any highly correlations among the variables. Now we have to choose which of them is necessary to keep for further analysis. To extract the more important variables we will make advantage of F-test statistics.

```
# we write all the variables in to a list
features_responses=df.columns.tolist()
```

```
X=df.iloc[:,:-1] # In this way, we see  is all the variables
y=df.iloc[:,-1] # In this way, we see  the response variable
print(X.shape, y.shape)
```

```
(1599, 11) (1599,)
```

```
from sklearn.feature_selection import f_classif

[f_stat, f_p_value]=f_classif(X,y)

# we define our dataframe

f_test_df = pd.DataFrame({'Feature':features_responses[:-1],'F statistic':f_stat,'p value'

f_test_df.sort_values('p value')

# from the below matrix we can obtain the most important variables
# important variables are those we are going to use in building models
```

	Feature	F statistic	p value
6	total_sulfur_dioxide	25.478510	8.533598e-25
9	sulphates	22.273376	1.225890e-21

```
# We take the first 8 more important variables
df.columns
variable=df[["total_sulfur_dioxide","sulphates","citric_acid","volatile_acidity","fixed_ac
```

0	fixed_acidity	6.283081	8.793967e-06

```
df.describe
df["quality"]
df["quality"].nunique()

# response variable has 6 categories
df["quality"].value_counts()
```

```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

1.4 Logistic regression basic tools

The LogisticRegression class can be configured for multinomial logistic regression by setting the “multi_class” argument to “multinomial” and the “solver” argument to a solver that supports multinomial logistic regression, such as “lbfgs“

```
from sklearn.linear_model import LogisticRegression
```

```
# define the multinomial logistic regression model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs')
```

The multinomial logistic regression model will be fit using cross-entropy loss and will predict the integer value for each integer encoded class label.

```
# evaluate multinomial logistic regression model
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
```

```
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
```

```
df.columns
```

```
Index(['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar',
       'chlorides', 'free sulfur_dioxide', 'total_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

1.5 Methodology

we are going to apply the follow methodology.

1. Before we proceed to further analysis we have to scale our data.
2. We will train our model.
3. we will evaluate it.
4. We visuazile the ROC CURVE

1) Scale τα δεδομένα μας 2) εκπαίδευση του μοντέλου μας 3) Roc Auc 4) threshold 5) oprimization με εύρεση την υπερπαραμετρο C

```
#we are need the min max sceler to scale our data
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler
min_max_sc = MinMaxScaler()
```

```
#scale_data=df.values[:,:-1]
scale_data=df[["total_sulfur_dioxide","sulphates","citric_acid","volatile_acidity","fixed_
```

```
#έχουμε κανει τα δεδομένα μας scale
scale_data_1=min_max_sc.fit_transform(scale_data)

scale_data_1=pd.DataFrame(scale_data_1)
scale_data_1.hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fa71ec537d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fa71ed33550>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fa71ecd6f10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fa71ed3a290>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fa71f692990>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fa71f6ba990>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fa71f75b390>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fa721411e10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fa721411bd0>]],
      dtype=object)
```

0 1 2

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline
from matplotlib import pyplot
```

```
scale_data_1
```

	0	1	2	3	4	5	6	7
0	0.098940	0.137725	0.00	0.000490	0.247788	0.106845	0.140845	0.606299
1	0.215548	0.209581	0.00	0.000641	0.283186	0.143573	0.338028	0.362205
2	0.169611	0.191617	0.04	0.000540	0.283186	0.133556	0.197183	0.409449
3	0.190813	0.149701	0.56	0.000135	0.584071	0.105175	0.225352	0.330709
4	0.098940	0.137725	0.00	0.000490	0.247788	0.106845	0.140845	0.606299
...	...	...	...	...	...	...	...	...
1594	0.134276	0.149701	0.08	0.000405	0.141593	0.130217	0.436620	0.559055
1595	0.159011	0.257485	0.10	0.000363	0.115044	0.083472	0.535211	0.614173
1596	0.120141	0.251497	0.13	0.000329	0.150442	0.106845	0.394366	0.535433
1597	0.134276	0.227545	0.12	0.000443	0.115044	0.105175	0.436620	0.653543
1598	0.127208	0.197605	0.47	0.000160	0.123894	0.091820	0.239437	0.511811

1599 rows × 8 columns

We split the data into train and test set

2) βήμα χωρίζουμε το σύνολο σε train και σε test set

1.5.1 OvR methodoly

One versus the rest

Data preprocessing

```
from sklearn.model_selection import train_test_split
X=scale_data_1
y=df["quality"]
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2, random_state=24)
```

1.5.2 Laber encoding στην y

```
#Lets encode target labels (y) with values between 0 and n_classes-1.
#We will use the LabelEncoder to do this.
from sklearn.preprocessing import LabelEncoder
label_encoder=LabelEncoder()
label_encoder.fit(y)
y=label_encoder.transform(y)
classes=label_encoder.classes_
```

1.5.3 Split data in train and test set

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
```

1.5.4 Normalize the data

```
from sklearn.preprocessing import MinMaxScaler
min_max_scaler=MinMaxScaler()
X_train_norm=min_max_scaler.fit_transform(X_train)
X_test_norm=min_max_scaler.fit_transform(X_test)
```

1.5.5. Classification

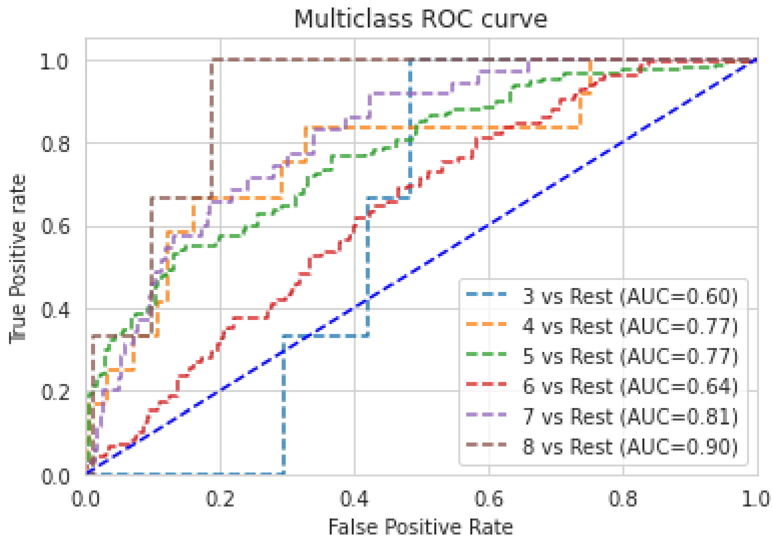
```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve,auc
```

Training Phase This will be done by parsing the training set to a classifier or classifiers Because we are dealing with 3 classes, this becomes a multiclass classification problem. We therefore us the One-vs-the-rest strategy.\ This strategy involves fitting one classifier per class. For each classifier, the class is fitted against all the other classes. Here, we use the Random Forest Classifier

Plot Auc ROC curve

1.5.6 Logistic regression OvR

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve,auc
from sklearn.linear_model import LogisticRegression


#Logistic Regression multiclass clasifier
#because we are dealing with multiclass data and so, the one versus rest strategy is used.
#learn to predict each class against the other.

RF=OneVsRestClassifier(LogisticRegression(multi_class="ovr"))
RF.fit(X_train_norm,y_train)
y_pred =RF.predict(X_test_norm)
pred_prob = RF.predict_proba(X_test_norm)


from sklearn.preprocessing import label_binarize
#binarize the y_values

y_test_binarized=label_binarize(y_test,classes=np.unique(y_test))

# roc curve for classes
fpr = {}
tpr = {}
thresh ={}
roc_auc = dict()

n_class = classes.shape[0]

for i in range(n_class):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test_binarized[:,i], pred_prob[:,i])
    roc_auc[i] = auc(fpr[i], tpr[i])

    # plotting
    plt.plot(fpr[i], tpr[i], linestyle='--',
             label='%s vs Rest (AUC=%0.2f)'%(classes[i],roc_auc[i]))

plt.plot([0,1],[0,1],'b--')
plt.xlim([0,1])
plt.ylim([0,1.05])
plt.title('Multiclass ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='lower right')
plt.show()
```



Chaprer 2

Now we gonna do the same methodology for the binary problem

2.1 Binary

We create a dummy variable from the quality so if the quality of wine is 6 and above the quality is considerated as good otherwise is considerated as bad

```
#df["quality"].unique_values()
np.unique(df["quality"])
```

```
array([3, 4, 5, 6, 7, 8])
```

```
dummy = np.where((df['quality'] == 4) | (df['quality'] == 5) | (df['quality'] == 3) ,0,1)
```

```
#print(dummy, type)
np.unique(dummy)
#print(dummy)
dummy=pd.DataFrame(dummy)
```

```
df_1=pd.concat([df,dummy],axis=1)
df_1=pd.DataFrame(df_1)
```

```
data_new1 = df_1.copy()                                      # Create copy of DataFrame
data_new1.columns = ['fixed_acidity','volatile_acidity','citric_acid','residual_sugar',
                     'chlorides','free sulfur_dioxide','total_sulfur_dioxide','density','p
```

```
data_new1.columns
```

```
Index(['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar',
```

```
          'chlorides', 'free sulfur_dioxide', 'total_sulfur_dioxide', 'density',
          'pH', 'sulphates', 'alcohol', 'quality', 'dummy'],
         dtype='object')
```

```
data_new1.info()
#print(data_new1, type)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed_acidity         1599 non-null   float64
 1   volatile_acidity      1599 non-null   float64
 2   citric_acid           1599 non-null   float64
 3   residual_sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur_dioxide   1599 non-null   float64
 6   total_sulfur_dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
 12  dummy                 1599 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 162.5 KB
```

2.2 DATA PREPROCESSING

```
scale_data=data_new1[["total_sulfur_dioxide","sulphates","citric_acid","volatile_acidity",
```

```
from sklearn.model_selection import train_test_split
#X=scale_data_1[["total_sulfur_dioxide","sulphates","citric_acid","volatile_acidity","fixe
X=scale_data[:, :-1]
y=data_new1["dummy"]
```

2.3 Label preprocessing

```
#Lets encode target labels (y) with values between 0 and n_classes-1.
#We will use the LabelEncoder to do this.
from sklearn.preprocessing import LabelEncoder
label_encoder=LabelEncoder()
label_encoder.fit(y)
y=label_encoder.transform(y)
classes=label_encoder.classes_
```

2.4 Split the data

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
```

2.5 Normalize the data

```
from sklearn.preprocessing import MinMaxScaler
min_max_scaler=MinMaxScaler()
X_train_norm=min_max_scaler.fit_transform(X_train)
X_test_norm=min_max_scaler.fit_transform(X_test)
```

2.6 Classification Logistic binary regression

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve,auc
from sklearn.linear_model import LogisticRegression
```

```
#Random Forest Classifier
#because we are dealing with multiclass data and so, the one versus rest strategy is used.
#learn to predict each class against the other.

RF=LogisticRegression()
RF.fit(X_train_norm,y_train)
y_pred =RF.predict(X_test_norm)
y_pred_prob = RF.predict_proba(X_test_norm)
```

2.7 Confussion Matrix

```
from sklearn import metrics
#ta predicting propabilities

y_pred_prob
y_pred_prob[:,1]
metrics.confusion_matrix(y_test,y_pred)
```

```
array([[115,  28],
       [ 80,  97]])
```

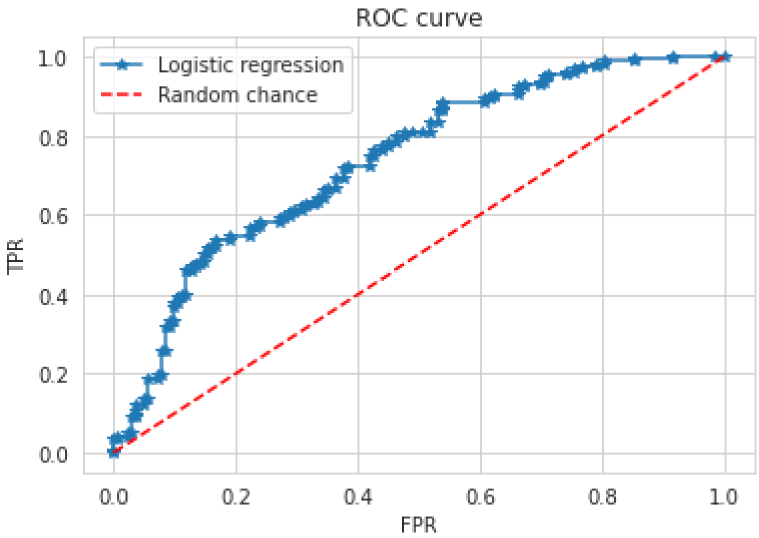
2.8 ROC AUC CURVE

```
pos_proba = y_pred_prob[:,1]
pos_proba
```

```
0.46719614, 0.44358257, 0.21945874, 0.13401218, 0.37729327,
0.38411205, 0.70184877, 0.46535233, 0.82683835, 0.26648945,
0.49868972, 0.06449684, 0.58201944, 0.19488977, 0.35373523,
0.67349527, 0.55427442, 0.59598713, 0.69262638, 0.9072665 ,
0.26837333, 0.35886961, 0.16025276, 0.35885534, 0.0922481 ,
0.7035536 , 0.10331539, 0.28995011, 0.49872261, 0.04689638,
0.36799643, 0.6185523 , 0.68629841, 0.58739765, 0.28322046,
0.47903541, 0.32673633, 0.57688509, 0.55946042, 0.25738916,
0.82788009, 0.43953355, 0.55392655, 0.04256743, 0.2555236 ,
0.68087224, 0.09848454, 0.59705142, 0.40937526, 0.37747751,
0.3766833 , 0.70622362, 0.08003987, 0.60377354, 0.48532981,
0.0841619 , 0.62404852, 0.41627628, 0.31896007, 0.30438515,
0.27533323, 0.64885739, 0.62404852, 0.69847943, 0.27029605,
0.14697268, 0.51112287, 0.4358004 , 0.51610882, 0.54439008,
0.45868168, 0.65104549, 0.27430929, 0.79459818, 0.1878779 ,
0.12310862, 0.41193415, 0.60909282, 0.83778975, 0.81282244,
0.16498232, 0.54008466, 0.6510321 , 0.26618601, 0.71854257,
0.26457149, 0.56324634, 0.68541542, 0.41624984, 0.24308131,
0.51706054, 0.89198599, 0.60200735, 0.18620085, 0.30003739,
0.38077867, 0.15408   , 0.82752011, 0.32009851, 0.304498  ,
0.40217272, 0.38834109, 0.50018131, 0.37123133, 0.36961778,
0.45473061, 0.3875015 , 0.45227056, 0.80470953, 0.76323395,
0.66170804, 0.71023502, 0.09954732, 0.05221328, 0.65756418,
0.66234847, 0.15283013, 0.7166643 , 0.19463078, 0.46840079,
0.309555  , 0.46041466, 0.80186556, 0.43032697, 0.53383554,
0.55698606, 0.70036806, 0.88602071, 0.30073791, 0.66045103,
0.72369893, 0.86582247, 0.6276584 , 0.75262294, 0.26789078,
0.40391417, 0.43156519, 0.40508358, 0.66108024, 0.81084763,
0.40143837, 0.40621462, 0.20406279, 0.53537431, 0.10964942,
0.64129692, 0.35264044, 0.60664159, 0.37392369, 0.57704194,
0.71151472, 0.29397122, 0.43085626, 0.57993906, 0.46041466,
0.42822573, 0.50963612, 0.36429959, 0.09607476, 0.7422711 ,
0.47723225, 0.67552444, 0.16871821, 0.32031118, 0.09825379,
0.36775616, 0.18437577, 0.17280791, 0.41653822, 0.653787  ,
0.08266538, 0.63532374, 0.46036022, 0.42899584, 0.33211974,
0.25841822, 0.04256743, 0.62714568, 0.60408986, 0.82683835,
0.42485107, 0.34298615, 0.32570339, 0.40883741, 0.41898034,
0.51798512, 0.34764571, 0.35991116, 0.6862632 , 0.57314207,
0.79298956, 0.34140648, 0.61998904, 0.77584226, 0.82374548,
0.36249688, 0.68537855, 0.33526633, 0.53609364, 0.74598748,
0.07521939, 0.40409492, 0.64719637, 0.15871773, 0.57884413,
0.42942124, 0.43267224, 0.28090949, 0.77820695, 0.78601586,
0.26618601, 0.57108311, 0.59962528, 0.30947999, 0.38712864,
0.34177054, 0.1919441 , 0.29464472, 0.66701758, 0.59490389,
0.32916122, 0.7056798 , 0.15223057, 0.38920073, 0.47844232,
0.77584226, 0.45373856, 0.29078451, 0.70857788, 0.71190849,
0.69779799, 0.08082588, 0.74618959, 0.45026397, 0.41756888,
0.56227453, 0.30415035, 0.04411498, 0.37224437, 0.3907869 ,
0.73376193, 0.44664591, 0.72727625, 0.71833711, 0.49370085,
0.46998016, 0.18457607, 0.4129835 , 0.46952713, 0.66383836,
0.49099935, 0.34764571, 0.0875907 , 0.32724242, 0.56455672,
0.7705793 , 0.23352948, 0.56345414, 0.43509634, 0.2787211 ,
0.25289852, 0.29724309, 0.25955745, 0.55890322, 0.70456373,
0.53287936, 0.42351088, 0.40378237, 0.40932556, 0.29001327,

0.44132454, 0.08884343, 0.304498  , 0.10467032, 0.81094426,
0.25869496, 0.68175125, 0.77447189, 0.79692358, 0.14002847,
0.76107043, 0.41396216, 0.49102676, 0.50978095, 0.4335763 ,
0.08915352, 0.10345852, 0.72318091, 0.47124032, 0.37940764,
```

```
#ypologizoyme to ROC AUC
fpr, tpr, thresholds = metrics.roc_curve(y_test, pos_proba)

plt.plot(fpr, tpr, '*-')
plt.plot([0, 1], [0, 1], 'r--')
plt.legend(['Logistic regression', 'Random chance'])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
```

```
Text(0.5, 1.0, 'ROC curve')
```




```
thresholds
# το accuracy
metrics.roc_auc_score(y_test, pos_proba)
```

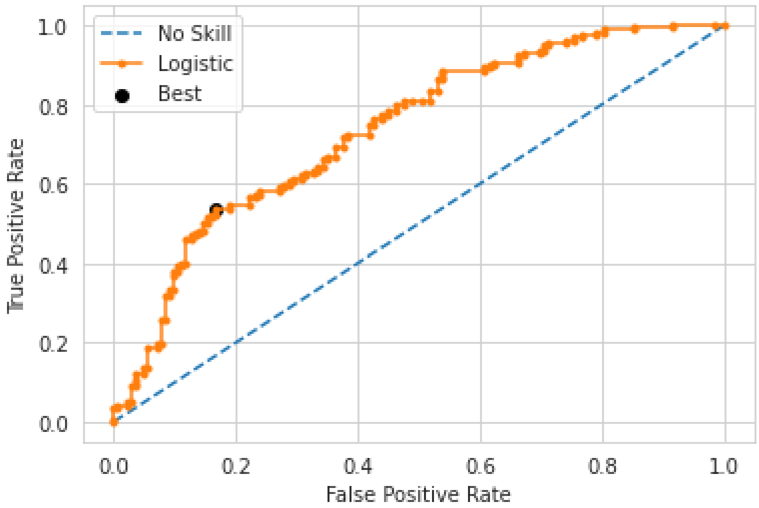
```
0.7388487218995694
```

2.9 Best threshold

```
from numpy import sqrt
from numpy import argmax
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from matplotlib import pyplot
# calculate the g-mean for each threshold
gmeans = sqrt(tpr * (1-fpr))



# locate the index of the largest g-mean
ix = argmax(gmeans)
print('Best Threshold=%f, G-Mean=%.3f' % (thresholds[ix], gmeans[ix]))
```

```
Best Threshold=0.517061, G-Mean=0.668
```

```
# plot the roc curve for the model
pyplot.plot([0,1], [0,1], linestyle='--', label='No Skill')
pyplot.plot(fpr, tpr, marker='.', label='Logistic')
pyplot.scatter(fpr[ix], tpr[ix], marker='o', color='black', label='Best')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
pyplot.legend()
# show the plot
pyplot.show()
```



Chapter 3 Hyperparameter

3.1 find the best value for C by hand

```
# tune regularization for multinomial logistic regression
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
from matplotlib import pyplot
```

```
param_grid=[
    {"penalty":["l1","l2","elasticnet","none"],
    "C":np.logspace(-4,4,20) } ]
```

```
from sklearn.model_selection import GridSearchCV
log_model=LogisticRegression()
clf=GridSearchCV(log_model, param_grid=param_grid, cv=3, verbose=True, n_jobs=-1)
```

```
X_train
y_train
```

```
array([0, 0, 1, ..., 1, 0, 1])
```

```
best_clf=clf.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 80 candidates, totalling 240 fits
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py:372: F
120 fits failed out of a total of 240.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score

Below are more details about the failures:
--------------------------------------------------------------------------------
60 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.p
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py", l
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py", l
    % (solver, penalty)
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
60 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.p
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py", l
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py", l
    % (solver, penalty)
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got elasticnet pena

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py:972: UserW
        nan 0.69899909        nan 0.61220511        nan 0.69899909
        nan 0.61689078        nan 0.69899909        nan 0.62002434
        nan 0.69899909        nan 0.63253657        nan 0.69899909
        nan 0.64034297        nan 0.69899909        nan 0.67161988
        nan 0.69899909        nan 0.67475161        nan 0.69899909
        nan 0.68023808        nan 0.69899909        nan 0.69040472
        nan 0.69899909        nan 0.69274848        nan 0.69899909
        nan 0.6950904         nan 0.69899909        nan 0.70056587
        nan 0.69899909        nan 0.69900276        nan 0.69899909
        nan 0.69978157        nan 0.69899909        nan 0.69899909
        nan 0.69899909        nan 0.69899909        nan 0.69899909
        nan 0.69821845        nan 0.69899909        nan 0.69587104
        nan 0.69899909]
  category=UserWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
```

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
print(best_clf.best_estimator_)
```

```
LogisticRegression(C=29.763514416313132)
```

```
best_clf.score(X_test,y_test)
```

```
0.703125
```

```
log_model.fit(X_train,y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
LogisticRegression()
```

```
pred_prob_t=log_model.predict_proba(X_test)
```

```
from sklearn.metrics import log_loss
C_list=np.geomspace(1e-1,1e+1,num=20)
CA=[]
Logarithimic_Loss = []

for c in C_list:
  log_reg=LogisticRegression(random_state=10,solver="lbfgs",C=c)
  log_reg.fit(X_train,y_train)
  score=log_reg.score(X_test,y_test)
  CA.append(score)
  print("the ca of C is {} and {}".format(c,score))
  log_loss2=log_loss(y_test,pred_prob_t)
  Logarithimic_Loss.append(log_loss2)
  print(" the logarithimic_Loss of C {} is {}".format(c, log_loss2))
  #print("")
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conv
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
the ca of C is 0.8858667904100825 and 0.709375
 the logarithimic_Loss of C 0.8858667904100825 is 0.5916977912853832
the ca of C is 1.1288378916846888 and 0.721875
 the logarithimic_Loss of C 1.1288378916846888 is 0.5916977912853832
the ca of C is 1.438449888287663 and 0.721875
 the logarithimic_Loss of C 1.438449888287663 is 0.5916977912853832
the ca of C is 1.8329807108324356 and 0.721875
 the logarithimic_Loss of C 1.8329807108324356 is 0.5916977912853832
the ca of C is 2.3357214690901213 and 0.721875
 the logarithimic_Loss of C 2.3357214690901213 is 0.5916977912853832
the ca of C is 2.9763514416313175 and 0.71875
 the logarithimic_Loss of C 2.9763514416313175 is 0.5916977912853832
the ca of C is 3.79269019073225 and 0.715625
 the logarithimic_Loss of C 3.79269019073225 is 0.5916977912853832
the ca of C is 4.832930238571752 and 0.7125
 the logarithimic_Loss of C 4.832930238571752 is 0.5916977912853832
the ca of C is 6.158482110660261 and 0.70625
 the logarithimic_Loss of C 6.158482110660261 is 0.5916977912853832
the ca of C is 7.847599703514611 and 0.7125
 the logarithimic_Loss of C 7.847599703514611 is 0.5916977912853832
the ca of C is 10.0 and 0.703125
 the logarithimic_Loss of C 10.0 is 0.5916977912853832
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conv
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conv
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conv
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
```

```
CA
data=np.array(CA).reshape(20,)
data_1=np.array(Logarithimic_Loss).reshape(20,)


df=zip(C_list,data,data_1)
df
```

```
<zip at 0x7fa71e1412d0>
```

```
df_final=pd.DataFrame(df,columns=["C_list","CA","Logarithimic_Loss"])
df_final.head(20)
```

	C_list	CA	Logarithimic_Loss
0	0.100000	0.687500	0.591698
1	0.127427	0.687500	0.591698
2	0.162378	0.706250	0.591698
3	0.206914	0.709375	0.591698
4	0.263665	0.712500	0.591698
5	0.335982	0.712500	0.591698
6	0.428133	0.715625	0.591698
7	0.545559	0.712500	0.591698
8	0.695193	0.712500	0.591698
9	0.885867	0.709375	0.591698
10	1.128838	0.721875	0.591698
11	1.438450	0.721875	0.591698
12	1.832981	0.721875	0.591698
13	2.335721	0.721875	0.591698
14	2.976351	0.718750	0.591698
15	3.792690	0.715625	0.591698
16	4.832930	0.712500	0.591698
17	6.158482	0.706250	0.591698
18	7.847600	0.712500	0.591698
19	10.000000	0.703125	0.591698

```
df_final.sort_values("Logarithimic_Loss",ascending=True)
# h kalyterh timh gia to C einai to 0.1
```

	C_list	CA	Logarithimic_Loss
0	0.100000	0.687500	0.591698
17	6.158482	0.706250	0.591698
16	4.832930	0.712500	0.591698
15	3.792690	0.715625	0.591698
14	2.976351	0.718750	0.591698
13	2.335721	0.721875	0.591698
12	1.832981	0.721875	0.591698
11	1.438450	0.721875	0.591698
10	1.128838	0.721875	0.591698
9	0.885867	0.709375	0.591698
8	0.695193	0.712500	0.591698
7	0.545559	0.712500	0.591698
6	0.428133	0.715625	0.591698
5	0.335982	0.712500	0.591698

3.2 Find the best value for C using the CV

Μολισ βρουμε το C τοτε ξανα υπολογιζουμε μια λογιστικη παλινδρόμηση βάζοντας το C = 0.1 με το χέρι.

```
from sklearn.linear_model import LogisticRegressionCV
from sklearn.model_selection import KFold

kf=KFold(n_splits=3, random_state=0, shuffle=True)



Log_reg3=LogisticRegressionCV(cv=kf ,random_state=15,Cs=C_list)
#Log_reg3=LogisticRegressionCV(random_state=15,Cs=C_list)
Log_reg3.fit(X_train,y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Conver
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
LogisticRegressionCV(Cs=array([ 0.1       ,  0.1274275 ,  0.16237767,  0.20691381,
0.26366509,
        0.33598183,  0.42813324,  0.54555948,  0.6951928 ,  0.88586679,
        1.12883789,  1.43844989,  1.83298071,  2.33572147,  2.97635144,
        3.79269019,  4.83293024,  6.15848211,  7.8475997 , 10.        ]),
                     cv=KFold(n_splits=3, random_state=0, shuffle=True),
                     random_state=15)
```

```
Log_reg3.score(X_test,y_test)
pred_proba=Log_reg3.predict_proba(X_test)
Log_loss3=log_loss(y_test,pred_proba)
Log_loss3
Log_reg3.C_
```

```
array([7.8475997])
```

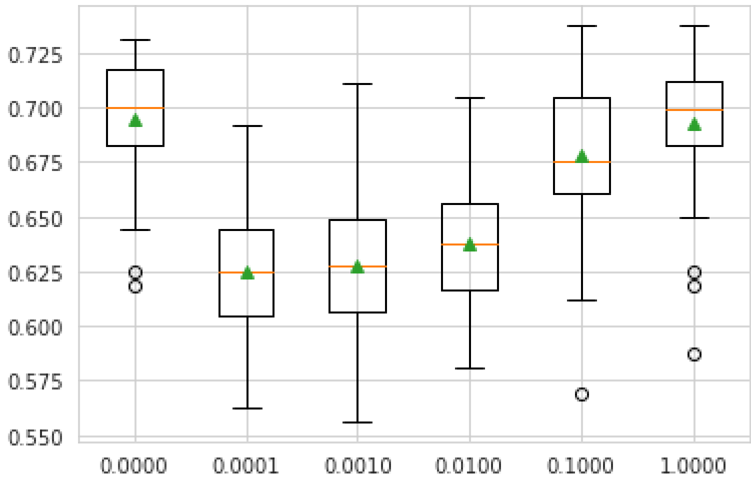
```
model = LogisticRegression()

# define the model evaluation procedure

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# evaluate the model and collect the scores

n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)

# report the model performance

print('Mean Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Mean Accuracy: 0.693 (0.035)
```

```
# get a list of models to evaluate
def get_models():
  models = dict()
  for p in [0.0, 0.0001, 0.001, 0.01, 0.1, 1.0]:
    # create name for model
    key = '%.4f' % p
    # turn off penalty in some cases
    if p == 0.0:
      # no penalty in this case
      models[key] = LogisticRegression( penalty='none')
    else:
      models[key] = LogisticRegression( C=p, penalty='l2')
  return models

# evaluate a give model using cross-validation
def evaluate_model(model, X, y):
  # define the evaluation procedure
```

```
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    # evaluate the model
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
    return scores

  # define dataset
  #X, y = get_dataset()
  # get the models to evaluate
  models = get_models()
  # evaluate the models and store results
  results, names = list(), list()
  for name, model in models.items():
    # evaluate the model and collect the scores
    scores = evaluate_model(model, X, y)
    # store the results
    results.append(scores)
    names.append(name)
    # summarize progress along the way
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
  # plot model performance for comparison
  pyplot.boxplot(results, labels=names, showmeans=True)
  pyplot.show()
```

```
>0.0000 0.695 (0.028)
>0.0001 0.625 (0.032)
>0.0010 0.627 (0.034)
>0.0100 0.638 (0.031)
>0.1000 0.678 (0.037)
>1.0000 0.693 (0.035)
```



Αυτο που μένει είναι να βάλουμε το hyperparameter που βρήκαμε στο μοντελο και να ξανα εκπαιδεύσουμε την λογιστίκη παλινδρόμηση με άλλο hyperparameter και αλλο threshold.

Multinomial regression

Tune Penalty for Multinomial Logistic Regression An important hyperparameter to tune for multinomial logistic regression is the penalty term.

This term imposes pressure on the model to seek smaller model weights. This is achieved by adding a weighted sum of the model coefficients to the loss function, encouraging the model to reduce the size of the weights along with the error while fitting the model.

A popular type of penalty is the L2 penalty that adds the (weighted) sum of the squared coefficients to the loss function. A weighting of the coefficients can be used that reduces the strength of the penalty from full penalty to a very slight penalty.

By default, the LogisticRegression class uses the L2 penalty with a weighting of coefficients set to 1.0. The type of penalty can be set via the "penalty" argument with values of "l1", "l2", "elasticnet" (e.g. both), although not all solvers support all penalty types. The weighting of the coefficients in the penalty can be set via the "C" argument.

```
# define the multinomial logistic regression model with a default penalty
LogisticRegression(multi_class='multinomial', solver='lbfgs', penalty='l2', C=1.0)
```

```
LogisticRegression(multi_class='multinomial')
```

C : float, default=1.0 Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

This means that values close to 1.0 indicate very little penalty and values close to zero indicate a strong penalty. A C value of 1.0 may indicate no penalty at all.

C close to 1.0: Light penalty. C close to 0.0: Strong penalty.

0s completed at 2:24 PM