**Team 31**

*Κανελλόπουλος Σωτήριος (03117101)*

*Καραντώνης Αντώνιος (03117439)*

*Κυριακίδης Δημήτριος (03117077)*

# Νευρωνικά Δίκτυα

## 1η Εργασία: Επιβλεπόμενη Μάθηση - Ταξινόμηση

### *Μελέτη datasets του αποθετηρίου UCI και της πλατφόρμας Kaggle*

# 1ο Μέρος: UCI

Το dataset που μας ανατέθηκε είναι το U11 "Quality Assessment of Digital Colposcopies", το οποίο αφορά την αξιολόγηση ποιότητας ενός συνόλου κολποσκοπήσεων προερχόμενων από το νοσοκομείο Hospital Universitario de Caracas. Σχετικά με το dataset αυτό, επισημαίνουμε τα εξής:

- Τα δεδομένα ήταν χωρισμένα σε τρία αρχεία csv, αναλόγως με την μέθοδο εξέτασης (κατά Hinselmann, Green ή Schiller) και ενοποιήθηκαν σε ένα panda dataframe.
- Το συνολικό πλήθος δειγμάτων είναι 287. Οι στήλες είναι 69, οι 62 εκ των οποίων είναι χαρακτηριστικά (μετρήσεις από την κολποσκόπηση, όπως cervix_area) και οι υπόλοιπες 7 είναι ετικέτες. Όλα τα χαρακτηριστικά είναι αριθμητικά δεδομένα.
- Στα αρχικά αρχεία υπάρχουν επικεφαλίδες - περιγραφές των χαρακτηριστικών, αλλά όχι αρίθμηση γραμμών.
- Οι ετικέτες των κλάσεων είναι 0 και 1 ("bad" ή "good") και αφορούν την ποιότητα της κολποσκόπησης. Οι ετικέτες βρίσκονται στις 7 τελευταίες στήλες (63-69), εκ των οποίων οι 6 πρώτες είναι υποκειμενικές αξιολογήσεις ειδικών και η 7η εκφράζει την επικρατούσα άποψη.
- Δεν υπάρχουν απουσιάζουσες τιμές. Οι τιμές των χαρακτηριστικών που είναι 0 ή 1 φαίνεται πως οφείλονται σε εικόνες του dataset εξ ολοκλήρου μαύρες ή άσπρες (π.χ. cervix_area = 0 αντιστοιχεί σε μαύρη εικόνα).

In [1]:
```
## CONCATENATION
## https://stackoverflow.com/a/36416258
## https://docs.python.org/3/library/glob.html

import pandas as pd
import glob
import os
import warnings

path = r"C:\Users\KYRIAKOS\Documents\JPNotebooks\NN\Assignment_1\Quality_Assessment_Digital_Colposcopy"
## path = r"C:\Users\anton\OneDrive\Desktop\Quality_Assessment_Digital_Colposcopy"

all_files = glob.iglob(os.path.join(path, "*.csv"))
## os.path.join makes concatenation independent of OS

df_from_each_file = (pd.read_csv(f) for f in all_files) ## dataframe
concatenated_df = pd.concat(df_from_each_file, ignore_index = True)
```

In [2]:
```
print(concatenated_df.shape)
```

```
(287, 69)
```

In [3]:
```
a = concatenated_df["consensus"].tolist()
print(a.count(1.0) / len(a))
print(a.count(0.0) / len(a))
```

```
0.7526132404181185
0.24738675958188153
```

Παρατηρούμε ότι το dataset **δεν είναι ισορροπημένο**, αφού η κλάση 1 εμφανίζεται με περίπου τριπλάσια συχνότητα από την 0.

In [4]:
```
#concatenated_df.to_csv(r"C:\Users\KYRIAKOS\Documents\JPNotebooks\NN\Assignment_1\Quality_Assessment_Digital_Colposcopy",
#concatenated_df.to_csv(r"C:\Users\anton\OneDrive\Desktop", encoding='utf-8')
```

In [5]:
```
labels_df = concatenated_df.iloc[:, [68]] ## τα labels της επικρατούσας άποψης είναι στην 68η στήλη
features_df = concatenated_df.iloc[:, 0:61]

features = features_df.values
labels = labels_df.values.reshape(287,)
```

In [6]:
```
from sklearn.model_selection import train_test_split

train, test, train_labels, test_labels = train_test_split(features, labels, test_size = 0.3)
```

In [7]:
```
import numpy as np
```

In [8]:
```
uci_accuracy = {}
uci_f1score = {}
ootb_train_times = {}
ootb_test_times = {}
opt_train_times = {}
opt_test_times = {}
```

# Out of the box

## Dummy

In [9]:
```
## https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html
from sklearn.dummy import DummyClassifier
from sklearn.metrics import classification_report, f1_score, accuracy_score
from time import perf_counter

## “stratified”: generates predictions by respecting the training set’s class distribution
dc_stratified = DummyClassifier(strategy = "stratified")

start = perf_counter()

model = dc_stratified.fit(train, train_labels)

end = perf_counter()
ootb_train_times.update({"Dummy" : end - start})

start = perf_counter()

dc_pred = dc_stratified.predict(test)

end = perf_counter()
ootb_test_times.update({"Dummy" : end - start})

dc_acc = accuracy_score(test_labels, dc_pred)
dc_f1 = f1_score(test_labels, dc_pred)
uci_accuracy.update({"Dummy" : dc_acc})
uci_f1score.update({"Dummy" : dc_f1})
print(classification_report(test_labels, dc_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.21      0.19      0.20        26
         1.0       0.67      0.69      0.68        61

    accuracy                           0.54        87
   macro avg       0.44      0.44      0.44        87
weighted avg       0.53      0.54      0.53        87
```

Η μέθοδος f1_score επιστρέφει εκ προεπιλογής το f1 score για την κλάση 1.

## Gaussian Naive Bayes

In [10]:
```
%%time
from sklearn.naive_bayes import GaussianNB
```

```
start = perf_counter()
gnb = GaussianNB()
# κάνουμε εκπαίδευση (fit) δηλαδή ουσιαστικά υπολογίζουμε μέση τιμή και διακύμανση για όλα τα χαρακτηριστικά και κλάσεις α
model = gnb.fit(train, train_labels)

end = perf_counter()
ootb_train_times.update({"GNB" : end - start})

start = perf_counter()
gnb_pred = dc_stratified.predict(test)
end = perf_counter()
ootb_test_times.update({"GNB" : end - start})

gnb_acc = accuracy_score(test_labels, gnb_pred)
gnb_f1 = f1_score(test_labels, gnb_pred)
uci_accuracy.update({"GNB" : gnb_acc})
uci_f1score.update({"GNB" : gnb_f1})
print(classification_report(test_labels, gnb_pred))
print(gnb.score(test, test_labels))
```

```
              precision    recall  f1-score   support

         0.0       0.43      0.23      0.30        26
         1.0       0.73      0.87      0.79        61

    accuracy                           0.68        87
   macro avg       0.58      0.55      0.55        87
weighted avg       0.64      0.68      0.64        87

0.7126436781609196
Wall time: 7 ms
```

## K Nearest Neighbors

In [11]:

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

start = perf_counter()
knn.fit(train, train_labels)
end = perf_counter()
ootb_train_times.update({"kNN" : end - start})


start = perf_counter()
knn_pred = knn.predict(test)
end = perf_counter()
ootb_test_times.update({"kNN" : end - start})


knn_acc = accuracy_score(test_labels, knn_pred)
knn_f1 = f1_score(test_labels, knn_pred)
uci_accuracy.update({"kNN" : knn_acc})
uci_f1score.update({"kNN" : knn_f1})
print(classification_report(test_labels, knn_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.64      0.27      0.38        26
         1.0       0.75      0.93      0.83        61

    accuracy                           0.74        87
   macro avg       0.69      0.60      0.61        87
weighted avg       0.72      0.74      0.70        87
```

## Logistic Regression

In [12]:

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()

start = perf_counter()
lr.fit(train, train_labels)
end = perf_counter()
ootb_train_times.update({"LR" : end - start})

start = perf_counter()
```

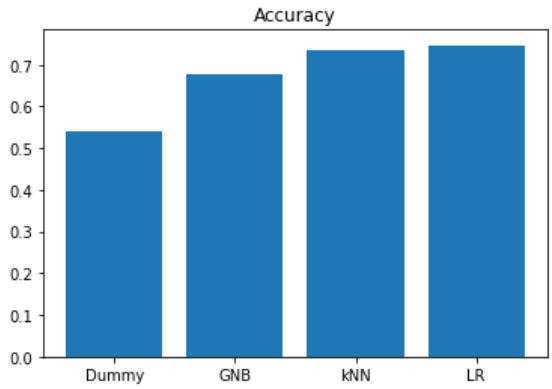
```
lr_pred = lr.predict(test)
end = perf_counter()
ootb_test_times.update({"LR" : end - start})

lr_acc = accuracy_score(test_labels, lr_pred)
lr_f1 = f1_score(test_labels, lr_pred)
uci_accuracy.update({"LR" : lr_acc})
uci_f1score.update({"LR" : lr_f1})
print(classification_report(test_labels, lr_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.83      0.19      0.31        26
         1.0       0.74      0.98      0.85        61

    accuracy                           0.75        87
   macro avg       0.79      0.59      0.58        87
weighted avg       0.77      0.75      0.69        87

c:\users\kyriakos\appdata\local\programs\python\python39\lib\site-packages\sklearn\linear_model\_logistic.py:814: Converge
nceWarning: lbfgs failed to converge (status=1):
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
  n_iter_i = _check_optimize_result(
```

## Comparison

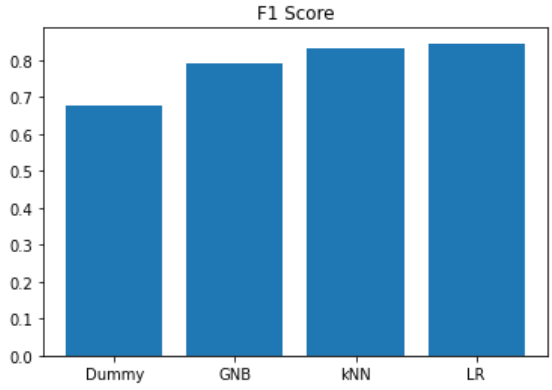
In [13]:

```
import matplotlib.pyplot as plt

plt.title("Accuracy")
plt.bar(*zip(*uci_accuracy.items()))
plt.show()
```




In [14]:

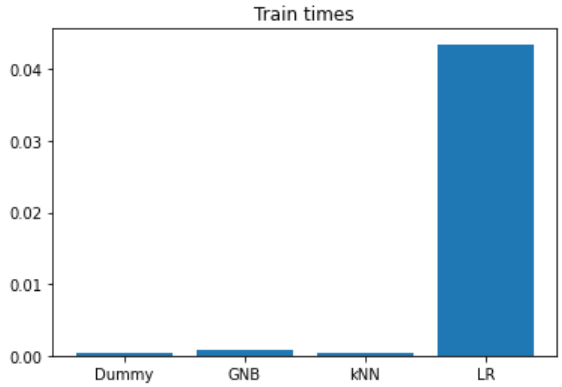
```
plt.title("F1 Score")
plt.bar(*zip(*uci_f1score.items()))
plt.show()
```



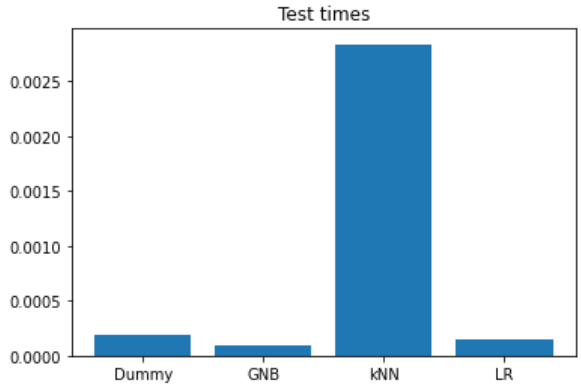

In [15]:

```
plt.title("Train times")
```

```
plt.bar(*zip(*ootb_train_times.items()))
plt.show()
```




In [16]:
```
plt.title("Test times")
plt.bar(*zip(*ootb_test_times.items()))
plt.show()
```




Παρατηρούμε ότι όλοι οι ταξινομητές είναι αρκετά καλύτεροι από τον dummy ως προς τις δύο μετρικές που μας ενδιαφέρουν. Ο ταξινομητής logistic regression φαίνεται να παράγει τα καλύτερα αποτελέσματα.

Ως προς το χρόνο παρατηρούμε ότι ο LR είναι πολύ πιο αργός από τους άλλους στο training, ενώ ο kNN είναι πολύ πιο αργός από τους άλλους στο testing.

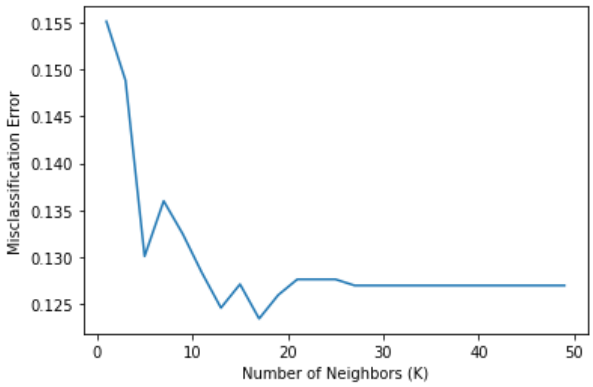
## Βελτιστοποίηση

Για την βελτιστοποίηση υπερπαραμέτρων θα μπορούσαμε να ακολουθήσουμε την μέθοδο cross validation που φαίνεται παρακάτω, η οποία βρίσκει βέλτιστη τιμή για μεμονωμένη παράμετρο.

In [18]:
```
from sklearn.model_selection import cross_val_score

# Κρατάμε μόνο τα περιττά k από το 1 έως το 50
neighbors = list(range(1, 50, 2))
# empty list that will hold cv scores
cv_scores = []
# perform 10-fold cross validation
for k in neighbors:
    knn_opt = KNeighborsClassifier(n_neighbors = k)
    scores = cross_val_score(knn_opt, train, train_labels, cv = 10, scoring = 'accuracy')
    cv_scores.append(scores.mean())
```

In [19]:
```
# το σφάλμα είναι το αντίστροφο της πιστότητας
mean_error = [1 - x for x in cv_scores]

# plot misclassification error vs k
plt.plot(neighbors, mean_error)
plt.xlabel("Number of Neighbors (K)")
plt.ylabel("Misclassification Error")
plt.show()

# determining best k
```

```
optimal_k = neighbors[mean_error.index(min(mean_error))]
print("The optimal number of neighbors (calculated in the training set) is %d" % optimal_k)
```




```
The optimal number of neighbors (calculated in the training set) is 17
```

Όμως κρίθηκε πιό σκόπιμο να χρησιμοποιηθεί η μέθοδος GridSearchCV για το cross validation ώστε να εξεταστούν αυτομάτως και με αποδοτικό τρόπο όλοι οι δυνατοί συνδυασμοί, και να μην χρειαστεί η εξαντλητική επανάληψη της ανωτέρω διαδικασίας.

In [20]:
```
#from sklearn.pipeline import Pipeline
from imblearn.pipeline import Pipeline

# φέρνουμε τις γνωστές μας κλάσεις για preprocessing
from sklearn.feature_selection import VarianceThreshold
from sklearn.preprocessing import StandardScaler, MinMaxScaler # φέρνουμε τον StandarScaler ως transformer που έχει .trans
from imblearn.over_sampling import RandomOverSampler
from sklearn.decomposition import PCA

# αρχικοποιούμε τον εκτιμητή (ταξινομητής) και τους μετασχηματιστές χωρίς υπερ-παραμέτρους
selector = VarianceThreshold()
std_scaler = StandardScaler()
minmax_scaler = MinMaxScaler() # (feature_range=(0, 1), *, copy=True, clip=False)
ros = RandomOverSampler()
pca = PCA()
```

In [21]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

vthreshold = [0, 0.05, 0.1, 0.15] ## προσαρμόζουμε τις τιμές μας στο variance που παρατηρήσαμε
n_components = [None, 10, 15, 20, 25, 30] ## for pca
minmax_ranges = [(0, 1), (0, 10), (0, 100)] ## for min max scaler
```

In [22]:
```
opt_accuracies = {}
opt_f1 = {}
```

## Gaussian Naive Bayes

In [23]:
```
NB_accuracies = {}
NB_f1_scores = {}
```

In [24]:
```
var_smoothings = np.logspace(-2, -11, num = 10)
## Return numbers spaced evenly on a log scale
## default var_smoothing = E-09

clf = GaussianNB()

pipe1 = Pipeline(steps = [("selector", selector), ("scaler", std_scaler), ("sampler", ros), ("pca", pca), ("gnb", clf)], m

pipe2 = Pipeline(steps = [("selector", selector), ("scaler", minmax_scaler), ("sampler", ros), ("pca", pca), ("gnb", clf)]

pipe3 = Pipeline(steps = [("selector", selector), ("pca", pca), ("gnb", clf)], memory = "tmp")
```

In [25]:
```
%%time

warnings.filterwarnings("ignore")
```

```
params_NB = {"gnb__var_smoothing": var_smoothings, "selector__threshold" : vthreshold, "pca__n_components" : n_components}

gsNB_acc = GridSearchCV(estimator = pipe1, param_grid = params_NB, cv = 10, scoring = "accuracy")
gsNB_f1 = GridSearchCV(estimator = pipe1, param_grid = params_NB, cv = 10, scoring = "f1")

## Δύο βελτιστοποιήσεις, ως προς accuracy και f1 score

gsNB_acc.fit(train, train_labels)
gsNB_f1.fit(train, train_labels)

print("Optimized for accuracy: ", gsNB_acc.best_params_)
print("Optimized for f1 score: ", gsNB_f1.best_params_)
```

```
Optimized for accuracy:  {'gnb__var_smoothing': 0.0001, 'pca__n_components': 15, 'selector__threshold': 0.05}
Optimized for f1 score:  {'gnb__var_smoothing': 1e-09, 'pca__n_components': None, 'selector__threshold': 0}
Wall time: 38.8 s
```

In [26]:

```
## print(gsNB_acc.cv_results_)
gsNB_acc_best1 = gsNB_acc.best_estimator_
gsNB_f1_best1 = gsNB_f1.best_estimator_

NB_accuracies.update({"pipe_1" : gsNB_acc.best_score_})
NB_f1_scores.update({"pipe_1" : gsNB_f1.best_score_})
```

In [27]:

```
%%time

warnings.filterwarnings("ignore")

params_NB = {"gnb__var_smoothing": var_smoothings, "selector__threshold" : vthreshold, "pca__n_components" : n_components,

gsNB_acc = GridSearchCV(estimator = pipe2, param_grid = params_NB, cv = 10, scoring = "accuracy")
gsNB_f1 = GridSearchCV(estimator = pipe2, param_grid = params_NB, cv = 10, scoring = "f1")

gsNB_acc.fit(train, train_labels)
gsNB_f1.fit(train, train_labels)

print("Optimized for accuracy: ", gsNB_acc.best_params_)
print("Optimized for f1 score: ", gsNB_f1.best_params_)
```

```
Optimized for accuracy:  {'gnb__var_smoothing': 0.01, 'pca__n_components': 20, 'scaler__feature_range': (0, 1), 'selector_
_threshold': 0}
Optimized for f1 score:  {'gnb__var_smoothing': 0.01, 'pca__n_components': 20, 'scaler__feature_range': (0, 1), 'selector_
_threshold': 0}
Wall time: 1min 59s
```

In [28]:

```
gsNB_acc_best2 = gsNB_acc.best_estimator_
gsNB_f1_best2 = gsNB_f1.best_estimator_

NB_accuracies.update({"pipe_2" : gsNB_acc.best_score_})
NB_f1_scores.update({"pipe_2" : gsNB_f1.best_score_})
```

In [29]:

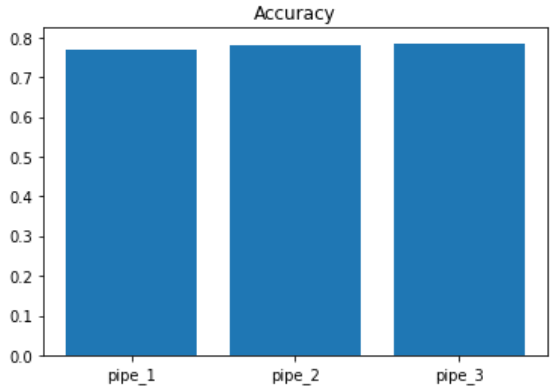
```
%%time

warnings.filterwarnings("ignore")

params_NB = {"gnb__var_smoothing": var_smoothings, "selector__threshold" : vthreshold, "pca__n_components" : n_components}

gsNB_acc = GridSearchCV(estimator = pipe3, param_grid = params_NB, cv = 10, scoring = "accuracy")
gsNB_f1 = GridSearchCV(estimator = pipe3, param_grid = params_NB, cv = 10, scoring = "f1")

gsNB_acc.fit(train, train_labels)
gsNB_f1.fit(train, train_labels)

print("Optimized for accuracy: ", gsNB_acc.best_params_)
print("Optimized for f1 score: ", gsNB_f1.best_params_)
```

```
Optimized for accuracy:  {'gnb__var_smoothing': 0.01, 'pca__n_components': None, 'selector__threshold': 0}
Optimized for f1 score:  {'gnb__var_smoothing': 0.01, 'pca__n_components': None, 'selector__threshold': 0}
Wall time: 24.8 s
```

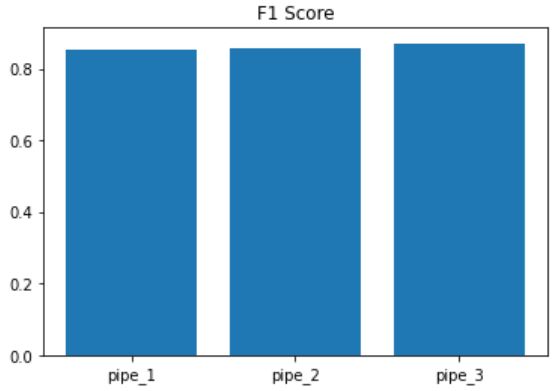
In [30]:

```
gsNB_acc_best3 = gsNB_acc.best_estimator_
gsNB_f1_best3 = gsNB_f1.best_estimator_

NB_accuracies.update({"pipe_3" : gsNB_acc.best_score_})
NB_f1_scores.update({"pipe_3" : gsNB_f1.best_score_})
```

In [31]:
```
plt.title("Accuracy")
plt.bar(*zip(*NB_accuracies.items()))
plt.show()
```




In [32]:
```
plt.title("F1 Score")
plt.bar(*zip(*NB_f1_scores.items()))
plt.show()
```




*Παρατηρούμε ότι βέλτιστη pipeline είναι η τρίτη.*

In [33]:
```
start = perf_counter()
gsNB_acc_best3.fit(train, train_labels)
end = perf_counter()
opt_train_times.update({"GNB" : end - start})

start = perf_counter()
pred = gsNB_acc_best3.predict(test)
end = perf_counter()
opt_test_times.update({"GNB" : end - start})

opt_accuracies.update({"GNB" : accuracy_score(test_labels, pred)})
```

In [34]:
```
gsNB_f1_best3.fit(train, train_labels)
pred = gsNB_f1_best3.predict(test)
opt_f1.update({"GNB" : f1_score(test_labels, pred)})
```

## K Nearest Neighbors

In [35]:
```
kNN_accuracies = {}
kNN_f1_scores = {}
```

In [36]:
```
neighbors = list(range(1, 40, 2))
## default leaf size = 30
p = [1, 2] ## default value is 2
## Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidea

clf = KNeighborsClassifier(n_jobs = 1) # η παράμετρος n_jobs = 1 χρησιμοποιεί όλους τους πυρήνες του υπολογιστή
```

```
pipe1 = Pipeline(steps = [("selector", selector), ("scaler", std_scaler), ("sampler", ros), ("pca", pca), ("kNN", clf)], m

pipe2 = Pipeline(steps = [("selector", selector), ("scaler", minmax_scaler), ("sampler", ros), ("pca", pca), ("kNN", clf)]

pipe3 = Pipeline(steps = [("selector", selector), ("pca", pca), ("kNN", clf)], memory = "tmp")
```

In [37]:

```
%%time

warnings.filterwarnings("ignore")

params_kNN = {"kNN__n_neighbors" : neighbors, "kNN__p" : p, "selector__threshold" : vthreshold, "pca__n_components" : n_co

gskNN_acc = GridSearchCV(estimator = pipe1, param_grid = params_kNN, cv = 10, scoring = "accuracy")
gskNN_f1 = GridSearchCV(estimator = pipe1, param_grid = params_kNN, cv = 10, scoring = "f1")

gskNN_acc.fit(train, train_labels)
gskNN_f1.fit(train, train_labels)

print("Optimized for accuracy: ", gskNN_acc.best_params_)
print("Optimized for f1 score: ", gskNN_f1.best_params_)
```

```
Optimized for accuracy:  {'kNN__n_neighbors': 1, 'kNN__p': 2, 'pca__n_components': None, 'selector__threshold': 0.05}
Optimized for f1 score:  {'kNN__n_neighbors': 1, 'kNN__p': 2, 'pca__n_components': None, 'selector__threshold': 0.05}
Wall time: 2min 47s
```

In [38]:

```
gskNN_acc_best1 = gskNN_acc.best_estimator_
gskNN_f1_best1 = gskNN_f1.best_estimator_

kNN_accuracies.update({"pipe_1" : gskNN_acc.best_score_})
kNN_f1_scores.update({"pipe_1" : gskNN_f1.best_score_})
```

In [39]:

```
%%time

warnings.filterwarnings("ignore")

params_kNN = {"kNN__n_neighbors" : neighbors, "kNN__p" : p, "selector__threshold" : vthreshold, "pca__n_components" : n_co

gskNN_acc = GridSearchCV(estimator = pipe2, param_grid = params_kNN, cv = 10, scoring = "accuracy")
gskNN_f1 = GridSearchCV(estimator = pipe2, param_grid = params_kNN, cv = 10, scoring = "f1")

gskNN_acc.fit(train, train_labels)
gskNN_f1.fit(train, train_labels)

print("Optimized for accuracy: ", gskNN_acc.best_params_)
print("Optimized for f1 score: ", gskNN_f1.best_params_)
```

```
Optimized for accuracy:  {'kNN__n_neighbors': 1, 'kNN__p': 2, 'pca__n_components': None, 'scaler__feature_range': (0, 1),
'selector__threshold': 0.05}
Optimized for f1 score:  {'kNN__n_neighbors': 1, 'kNN__p': 2, 'pca__n_components': None, 'scaler__feature_range': (0, 1),
'selector__threshold': 0.05}
Wall time: 8min 25s
```

In [40]:

```
gskNN_acc_best2 = gskNN_acc.best_estimator_
gskNN_f1_best2 = gskNN_f1.best_estimator_

kNN_accuracies.update({"pipe_2" : gskNN_acc.best_score_})
kNN_f1_scores.update({"pipe_2" : gskNN_f1.best_score_})
```

In [41]:

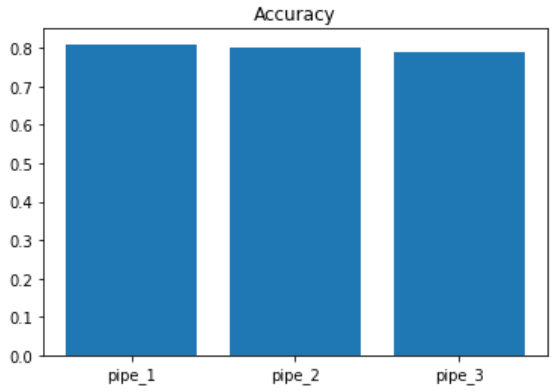
```
%%time

warnings.filterwarnings("ignore")

params_kNN = {"kNN__n_neighbors" : neighbors, "kNN__p" : p, "selector__threshold" : vthreshold, "pca__n_components" : n_co

gskNN_acc = GridSearchCV(estimator = pipe3, param_grid = params_kNN, cv = 10, scoring = "accuracy")
gskNN_f1 = GridSearchCV(estimator = pipe3, param_grid = params_kNN, cv = 10, scoring = "f1")

gskNN_acc.fit(train, train_labels)
gskNN_f1.fit(train, train_labels)

print("Optimized for accuracy: ", gskNN_acc.best_params_)
print("Optimized for f1 score: ", gskNN_f1.best_params_)
```

```
## print("best k: ", gs.best_estimator_.get_params()['n_neighbors'])
## print("best p: ", gs.best_estimator_.get_params()['p'])
```

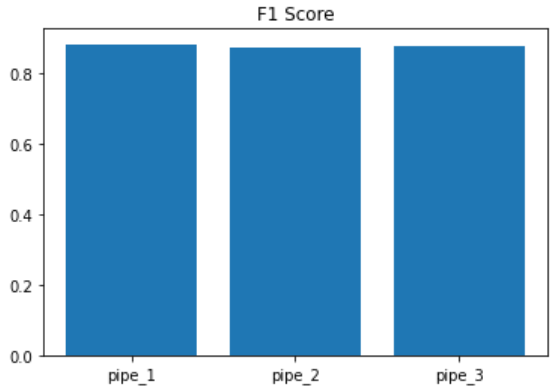
```
Optimized for accuracy:  {'kNN__n_neighbors': 17, 'kNN__p': 2, 'pca__n_components': 10, 'selector__threshold': 0}
Optimized for f1 score:  {'kNN__n_neighbors': 17, 'kNN__p': 2, 'pca__n_components': 10, 'selector__threshold': 0}
Wall time: 1min 52s
```

In [42]:
```
gskNN_acc_best3 = gskNN_acc.best_estimator_
gskNN_f1_best3 = gskNN_f1.best_estimator_

kNN_accuracies.update({"pipe_3" : gskNN_acc.best_score_})
kNN_f1_scores.update({"pipe_3" : gskNN_f1.best_score_})
```

In [43]:
```
plt.title("Accuracy")
plt.bar(*zip(*kNN_accuracies.items()))
plt.show()
```




In [44]:
```
plt.title("F1 Score")
plt.bar(*zip(*kNN_f1_scores.items()))
plt.show()
```




*Παρατηρούμε ότι βέλτιστη pipeline είναι η πρώτη.*

In [68]:
```
start = perf_counter()
gskNN_acc_best1.fit(train, train_labels)
end = perf_counter()
opt_train_times.update({"kNN" : end - start})

start = perf_counter()
pred = gskNN_acc_best1.predict(test)
end = perf_counter()
opt_test_times.update({"kNN" : end - start})

opt_accuracies.update({"kNN" : accuracy_score(test_labels, pred)})
```

In [69]:
```
gskNN_f1_best1.fit(train, train_labels)
pred = gskNN_f1_best1.predict(test)
opt_f1.update({"kNN" : f1_score(test_labels, pred)})
```

## Logistic Regression

In [47]:
```
LR_accuracies = {}
LR_f1_scores = {}
```

In [48]:
```
max_iters = list(range(1, 20, 2))
solvers = ["newton-cg", "lbfgs", "liblinear", "sag", "saga"]
penalties = ["l1", "l2", "elasticnet", "none"]
## tolerances = np.logspace(-2, -22, num = 6)
## default value E-04
classWeights = ["balanced", None]

clf = LogisticRegression()

pipe1 = Pipeline(steps = [("selector", selector), ("scaler", std_scaler), ("sampler", ros), ("pca", pca), ("lr", clf)], me

pipe2 = Pipeline(steps = [("selector", selector), ("scaler", minmax_scaler), ("sampler", ros), ("pca", pca), ("lr", clf)],

pipe3 = Pipeline(steps = [("selector", selector), ("pca", pca), ("lr", clf)], memory = "tmp")
```

In [49]:
```
%%time

warnings.filterwarnings("ignore")

paramsLR = {"lr__max_iter" : max_iters, "lr__solver" : solvers, "lr__penalty" : penalties, "lr__class_weight" : classWeigh

gsLR_acc = GridSearchCV(estimator = pipe1, param_grid = paramsLR, cv = 10, scoring = "accuracy")
gsLR_f1 = GridSearchCV(estimator = pipe1, param_grid = paramsLR, cv = 10, scoring = "f1")

gsLR_acc.fit(train, train_labels)
gsLR_f1.fit(train, train_labels)

print("Optimized for accuracy: ", gsLR_acc.best_params_)
print("Optimized for f1 score: ", gsLR_f1.best_params_)
```

```
Optimized for accuracy:  {'lr__class_weight': 'balanced', 'lr__max_iter': 3, 'lr__penalty': 'none', 'lr__solver': 'saga',
'pca__n_components': 25, 'selector__threshold': 0.1}
Optimized for f1 score:  {'lr__class_weight': None, 'lr__max_iter': 1, 'lr__penalty': 'l1', 'lr__solver': 'saga', 'pca__n_
components': 30, 'selector__threshold': 0}
Wall time: 28min 45s
```

In [50]:
```
gsLR_acc_best1 = gsLR_acc.best_estimator_
gsLR_f1_best1 = gsLR_f1.best_estimator_

LR_accuracies.update({"pipe_1" : gsLR_acc.best_score_})
LR_f1_scores.update({"pipe_1" : gsLR_f1.best_score_})
```

In [51]:
```
%%time

warnings.filterwarnings("ignore")

paramsLR = {"lr__max_iter" : max_iters, "lr__solver" : solvers, "lr__penalty" : penalties, "lr__class_weight" : classWeigh

gsLR_acc = GridSearchCV(estimator = pipe2, param_grid = paramsLR, cv = 10, scoring = "accuracy")
gsLR_f1 = GridSearchCV(estimator = pipe2, param_grid = paramsLR, cv = 10, scoring = "f1")

gsLR_acc.fit(train, train_labels)
gsLR_f1.fit(train, train_labels)

print("Optimized for accuracy: ", gsLR_acc.best_params_)
print("Optimized for f1 score: ", gsLR_f1.best_params_)
```

```
Optimized for accuracy:  {'lr__class_weight': None, 'lr__max_iter': 1, 'lr__penalty': 'l1', 'lr__solver': 'saga', 'pca__n_
components': None, 'scaler__feature_range': (0, 100), 'selector__threshold': 0}
Optimized for f1 score:  {'lr__class_weight': 'balanced', 'lr__max_iter': 5, 'lr__penalty': 'l2', 'lr__solver': 'lbfgs',
'pca__n_components': 15, 'scaler__feature_range': (0, 1), 'selector__threshold': 0}
Wall time: 1h 36min 44s
```

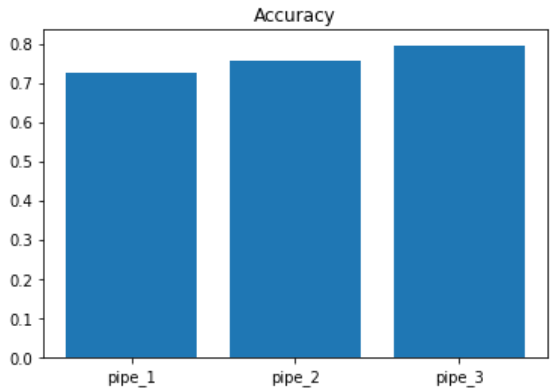
In [52]:
```
gsLR_acc_best2 = gsLR_acc.best_estimator_
gsLR_f1_best2 = gsLR_f1.best_estimator_

LR_accuracies.update({"pipe_2" : gsLR_acc.best_score_})
LR_f1_scores.update({"pipe_2" : gsLR_f1.best_score_})
```

In [53]:
```
%%time

warnings.filterwarnings("ignore")

paramsLR = {"lr__max_iter" : max_iters, "lr__solver" : solvers, "lr__penalty" : penalties, "lr__class_weight" : classWeigh

gsLR_acc = GridSearchCV(estimator = pipe3, param_grid = paramsLR, cv = 10, scoring = "accuracy")
gsLR_f1 = GridSearchCV(estimator = pipe3, param_grid = paramsLR, cv = 10, scoring = "f1")

gsLR_acc.fit(train, train_labels)
gsLR_f1.fit(train, train_labels)

print("Optimized for accuracy: ", gsLR_acc.best_params_)
print("Optimized for f1 score: ", gsLR_f1.best_params_)
```

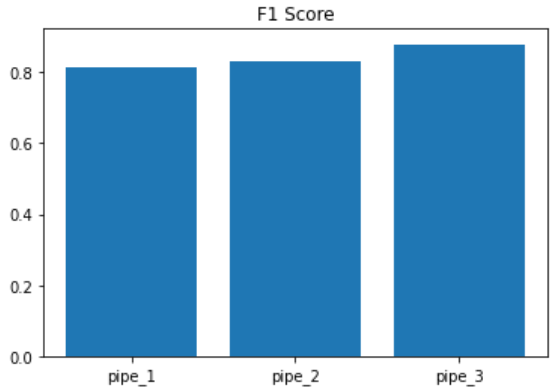
```
Optimized for accuracy:  {'lr__class_weight': None, 'lr__max_iter': 1, 'lr__penalty': 'l1', 'lr__solver': 'liblinear', 'pc
a__n_components': 10, 'selector__threshold': 0}
Optimized for f1 score:  {'lr__class_weight': None, 'lr__max_iter': 1, 'lr__penalty': 'l1', 'lr__solver': 'liblinear', 'pc
a__n_components': 10, 'selector__threshold': 0}
Wall time: 20min 44s
```

In [54]:
```
gsLR_acc_best3 = gsLR_acc.best_estimator_
gsLR_f1_best3 = gsLR_f1.best_estimator_

LR_accuracies.update({"pipe_3" : gsLR_acc.best_score_})
LR_f1_scores.update({"pipe_3" : gsLR_f1.best_score_})
```

In [55]:
```
plt.title("Accuracy")
plt.bar(*zip(*LR_accuracies.items()))
plt.show()
```




In [56]:
```
plt.title("F1 Score")
plt.bar(*zip(*LR_f1_scores.items()))
plt.show()
```




*Παρατηρούμε ότι βέλτιστη pipeline είναι η τρίτη.*

In [57]:
```
start = perf_counter()
gsLR_acc_best3.fit(train, train_labels)
end = perf_counter()
```

```
opt_train_times.update({"LR" : end - start})

start = perf_counter()
pred = gsLR_acc_best3.predict(test)
end = perf_counter()
opt_test_times.update({"LR" : end - start})

opt_accuracies.update({"LR" : accuracy_score(test_labels, pred)})
```

In [58]:
```
gsLR_acc_best3.fit(train, train_labels)
pred = gsLR_acc_best3.predict(test)

opt_f1.update({"LR" : f1_score(test_labels, pred)})
```

Για τις παραπάνω βελτιστοποιήσεις ελέγξαμε μερικές φορές ποιες τιμές προκύπτουν για τις παραμέτρους και μετακινήσαμε τα διαστήματα των δυνατών τιμών κατάλληλα ώστε να μην πέφτουμε στα άκρα τους. Ωστόσο, επειδή τρέχουμε τρεις pipelines για κάθε ταξινομητή, είναι αδύνατον να επιλέξουμε τιμές που να ταιριάζουν και για τις τρεις ταυτοχρόνως. Επομένως, είναι αναμενόμενο σε ορισμένες περιπτώσεις να επιλέγεται ακραία τιμή του εκάστοτε διαστήματος.

# Αποτελέσματα

## Συγκρίσεις

Παρακάτω φαίνονται τα αποτελέσματα μας από τις βελτιστοποιήσεις που κάναμε, καθώς και συγκρίσεις με τους αντίστοιχους out-of-the-box ταξινομητές.

*Accuracy και f1 score για κάθε ταξινομητή, out of the box και βελτιστοποιημένο.*

| Classifier | Acc-OotB | Acc-optimal | F1-OotB | F1-optimal |
|---|---|---|---|---|
| Dummy | 0.540 | - | 0.677 | - |
| Gaussian Naive Bayes | 0.678 | 0.770 | 0.791 | 0.859 |
| K Nearest Neighbors | 0.736 | 0.747 | 0.832 | 0.833 |
| Logistic Regression | 0.747 | 0.747 | 0.845 | 0.845 |

*Χρόνοι training και testing για κάθε ταξινομητή, σε ms.*

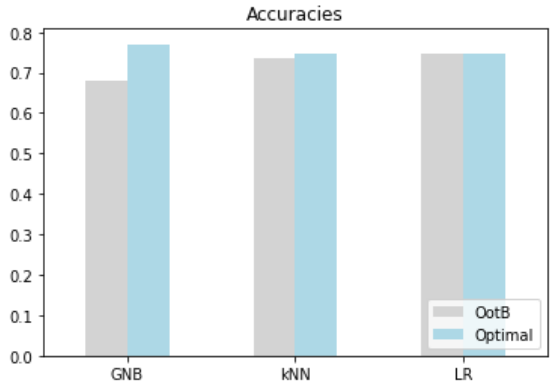
| Classifier | Train time-OotB | Train time-optimal | Test time-OotB | Test time-optimal |
|---|---|---|---|---|
| Dummy | 0.41 | - | 0.19 | - |
| Gaussian Naive Bayes | 0.78 | 6.14 | 0.09 | 0.79 |
| K Nearest Neighbors | 0.39 | 10.47 | 2.83 | 3.84 |
| Logistic Regression | 43.52 | 6.21 | 0.15 | 0.39 |

In [70]:
```
final_accuracies = {}
final_f1 = {}
for i in ["GNB", "kNN", "LR"]:
    final_accuracies.update({i : []})
    final_accuracies[i].append(uci_accuracy[i]) ## ootb
    final_accuracies[i].append(opt_accuracies[i])
    final_f1.update({i : []})
    final_f1[i].append(uci_f1score[i])
    final_f1[i].append(opt_f1[i])
```

*Τα αποτελέσματά μας σε γραφήματα:*

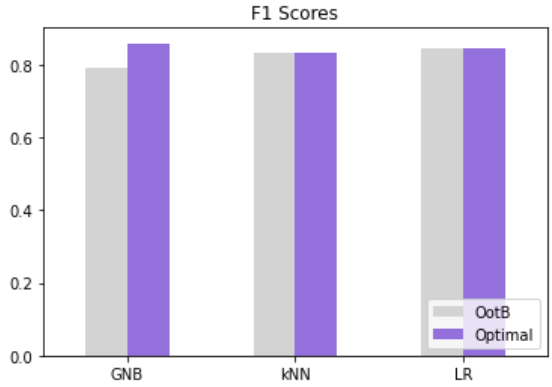
In [71]:
```
labels = list(final_accuracies.keys())
ootb_accs = [i[0] for i in final_accuracies.values()]
opt_accs = [i[1] for i in final_accuracies.values()]
df = pd.DataFrame({"OotB" : ootb_accs, "Optimal" : opt_accs}, index = labels)

## https://matplotlib.org/stable/gallery/color/named_colors.html

ax = df.plot.bar(rot = 0, color = {"OotB" : "lightgray", "Optimal" : "lightblue"}, title = "Accuracies")
ax.legend(loc = "lower right")
```

Out[71]:
```
<matplotlib.legend.Legend at 0x1e4a37df4c0>
```




In [72]:
```
labels = list(final_f1.keys())
ootb_f1s = [i[0] for i in final_f1.values()]
opt_f1s = [i[1] for i in final_f1.values()]
df = pd.DataFrame({"OotB" : ootb_f1s, "Optimal" : opt_f1s}, index = labels)

ax = df.plot.bar(rot = 0, color = {"OotB" : "lightgray", "Optimal" : "mediumpurple"}, title = "F1 Scores")
ax.legend(loc = "lower right")
```

Out[72]:
```
<matplotlib.legend.Legend at 0x1e4ad206f40>
```




## Συμπεράσματα και παρατηρήσεις

Συμπεραίνουμε ότι ο καλύτερος ταξινομητής (βελτιστοποιημένος και ως προς accuracy και ως προς f1 score) είναι ο **GNB**, ενώ ο χειρότερος είναι ο **kNN**. Μάλιστα, ο GNB είναι ο μόνος που παρουσιάζει αξιοσημείωτη διαφορά πριν από και μετά την βελτιστοποίηση. Όπως παρατηρούμε παρακάτω από τα confusion matrices, κάνει πολλά λάθη στην ταξινόμηση ως προς την κλάση 0, όμως επειδή αυτή περιλαμβάνει πολύ μικρό ποσοστό των στοιχείων ο ταξινομητής καταλήγει να έχει αρκετά καλές τιμές μετρικών. Επειδή το συγκεκριμένο dataset του προβλήματος είναι αρκετά imbalanced και μικρό, παρ' όλο που χρησιμοποιήθηκαν transformers για υπερδειγματοληψία (RandomOverSampler), δεν ήταν δυνατό να επιτευχθεί ικανοποιητική ακρίβεια ως προς την κλάση 0.

Παρατηρούμε ότι, ενώ δοκιμάστηκαν πολλές παράμετροι για τις pipelines του LR, αυτός δεν παρουσίασε καμία βελτίωση στην επίδοσή του στο test set μετά την βελτιστοποίηση. Αυτό μάλλον οφείλεται στο ανεπαρκές μέγεθος των δεδομένων, το οποίο δεν εγγυάται ότι η βελτιστοποίηση με cross validation στο train set θα οδηγήσει σε βελτίωση επί του test set.

Σχετικά με τους χρόνους παρατηρούμε τα εξής:

- Ο OotB logistic regression είναι πολύ πιο αργός από τους άλλους ταξινομητές στην εκπαίδευση.
- Οι βελτιστοποιημένοι ταξινομητές GNB και kNN χρειάζονται περισσότερο χρόνο εκπαίδευσης, πιθανώς επειδή τα pipelines βελτιστοποίησης περιέχουν επιπλέον transformers προεπεξεργασίας.
- Ο βελτιστοποιημένος LR έγινε γρηγορότερος στο training, λογικά επειδή κάποια από τις παραμέτρους που επηρεάσαμε οδηγεί σε καλύτερη ταχύτητα.
- Ο kNN γενικά χρειάζεται περισσότερο χρόνο από τους άλλους ταξινομητές στο testing, μάλλον γιατί η διαδικασία εύρεσης παρόμοιων γειτόνων είναι αρκετά αργή.
- Ο ταξινομητής GNB είναι αρκετά γρήγορος σε όλες τις περιπτώσεις.

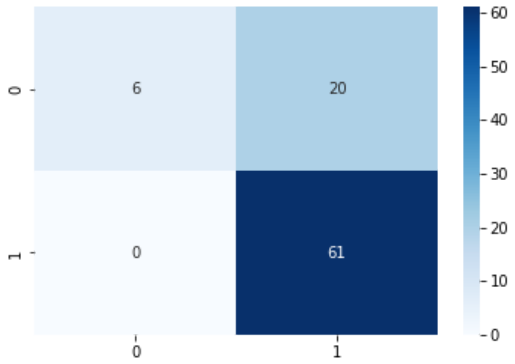
Λαμβάνοντας υπ' όψιν τόσο τις τιμές των μετρικών όσο και τις παραπάνω παρατηρήσεις σχετικά με τις χρονικές επιδόσεις, είναι σαφές ότι ο βελτιστοποιημένος ταξινομητής Gaussian Naive Bayes είναι η καλύτερη επιλογή για το συγκεκριμένο πρόβλημα.

## Confusion matrices

**Από βελτιστοποίηση ως προς accuracy**

In [76]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

## https://seaborn.pydata.org/generated/seaborn.heatmap.html
## https://matplotlib.org/stable/tutorials/colors/colormaps.html

## best is GNB
pred_NB = gsNB_acc_best3.predict(test)
# Compute confusion matrix
cnf_matrix = confusion_matrix(test_labels, pred_NB)
# τυπώνουμε το confusion matrix
sns.heatmap(cnf_matrix, annot = True, cmap = "Blues")
```

Out[76]:
```
<AxesSubplot:>
```




In [84]:
```
## worst is kNN
pred_kNN = gskNN_acc_best1.predict(test)
# Compute confusion matrix
cnf_matrix = confusion_matrix(test_labels, pred_kNN)
# τυπώνουμε το confusion matrix
sns.heatmap(cnf_matrix, annot = True, cmap = "Purples")
```

Out[84]:
```
<AxesSubplot:>
```