Data analysis

In [2]:
```
import pandas as pd, numpy as np, matplotlib.pyplot as plt, xgboost as
xgb, sklearn, copy, graphviz, joblib
#classifiers
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.ensemble import VotingClassifier
#analysis tool
from deepchecks.tabular import Dataset
from deepchecks.tabular.suites import train_test_validation,
model_evaluation
#Others
from sklearn.model_selection import GridSearchCV, GroupShuffleSplit,
cross_val_score
from sklearn.metrics import confusion_matrix, roc_curve,
mean_squared_error, auc, accuracy_score
from sklearn.feature_selection import SelectKBest, mutual_info_classif
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline
from IPython.display import display, Markdown
```

In [3]:
```
import datetime
t0 = datetime.datetime.now()
df = pd.read_excel('./Reduced Features for TAI project.xlsx')
df=df.set_index('Patient ID')
load_time = (datetime.datetime.now() - t0).total_seconds()
print(load_time, "seconds load time")
```

```
10.04253 seconds load time
```

In [4]:
```
X = df.copy()
y = X.pop('Label')
```

In [5]:
```
X.head()
```

Out[5]:

Patient ID	original_shape_Elongation	original_firstorder_Kurtosis	original_firstorder_Skewness	LHL
1	0.661690	2.817688	0.769536	
1	0.750849	2.084800	0.442780	
1	0.619781	2.590759	0.384512	
2	0.388733	2.932863	0.738215	
2	0.820531	2.814081	0.773252	

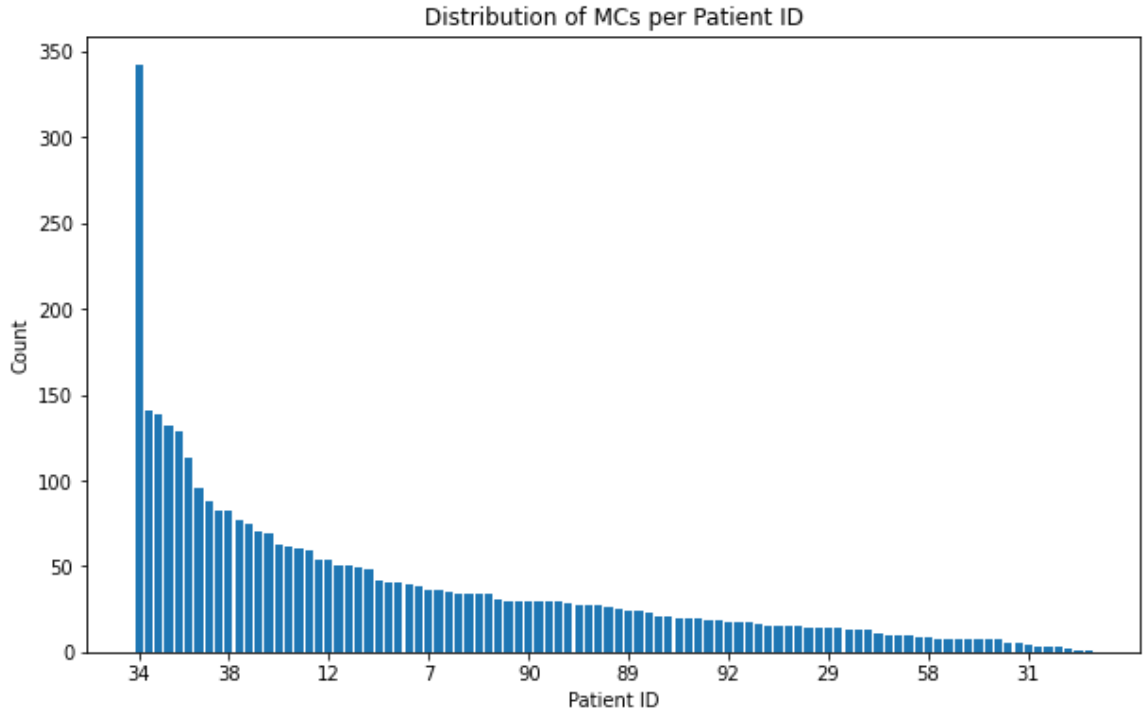
5 rows × 150 columns

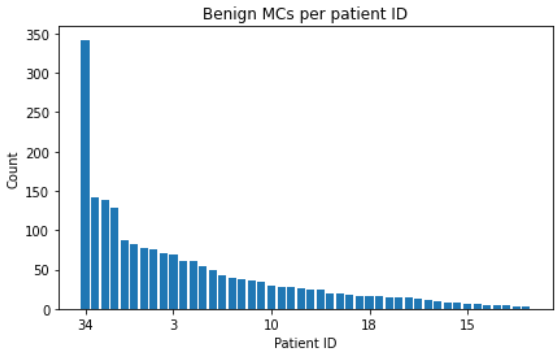
In [6]:
```
y.head()
```

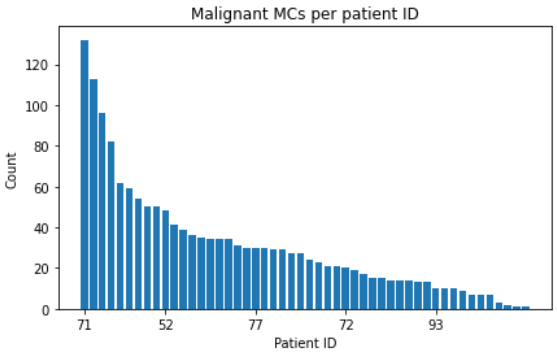
Out[6]:
```
Patient ID
1    0
1    0
1    0
2    0
2    0
Name: Label, dtype: int64
```

In [8]:

```
patient_count = X.index.value_counts()
#patient count plot
plt.figure(figsize=(10, 6))
plt.title('Distribution of MCs per Patient ID')
plt.bar(list(range(0, len(patient_count))), patient_count)
ticks=[0] + list(range(9, 90, 10))
plt.xticks(ticks=ticks, labels=patient_count.index[ticks])
plt.xlabel('Patient ID')
plt.ylabel('Count')
plt.show()

label_pat = y.groupby('Patient
ID').value_counts().sort_values(ascending=False) #this will give the
amount of labels each patient id has
pat_0 = label_pat[[x[1] == 0 for x in label_pat.index]]
#(label_pat.index contains tuples with inside the patient ID and
label)
pat_1 = label_pat[[x[1] == 1 for x in label_pat.index]]

#subplots comparing patient count between labels
fig, ax = plt.subplots(1, 2, figsize=(15, 4))
#patient ID count for label 0 without patient ID 34
ax[0].set_title('Benign MCs per patient ID')
ax[0].set_xlabel('Patient ID')
ax[0].set_ylabel('Count')
ax[0].bar(list(range(len(pat_0))), pat_0)
ticks = [0]+list(range(9, 40, 10))
ax[0].set_xticks(ticks=ticks)
ax[0].set_xticklabels(labels=[x[0] for x in pat_0.iloc[ticks].index])
#patient ID count for label 1
ax[1].set_title('Malignant MCs per patient ID')
ax[1].set_xlabel('Patient ID')
ax[1].set_ylabel('Count')
ax[1].bar(list(range(len(pat_1))), pat_1)
ticks = [0]+list(range(9, 40, 10))
ax[1].set_xticks(ticks=ticks)
ax[1].set_xticklabels([x[0] for x in pat_1.iloc[ticks].index])
plt.show()
```

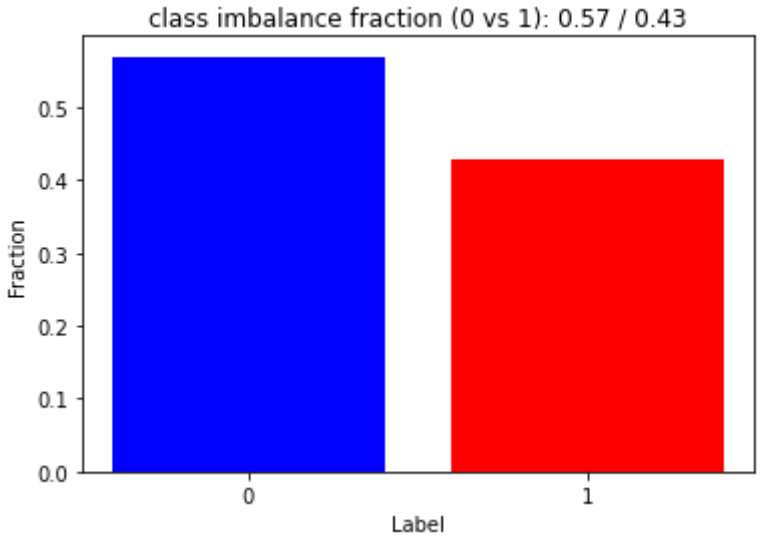






In [9]:

```
#Check if there is class imbalance
class_count = y.value_counts()
label_0_frac = round(class_count[0]/class_count.sum(), 2)
label_1_frac = round(class_count[1]/class_count.sum(), 2)
plt.figure()
plt.title(f'class imbalance fraction (0 vs 1): {label_0_frac} /
{label_1_frac}')
plt.bar([0, 1], [label_0_frac, label_1_frac], color=['b', 'r'])
plt.xticks(ticks=[0, 1], labels=[0, 1])
plt.xlabel('Label')
plt.ylabel('Fraction')
plt.show()
```



Looks alright

Feature reduction with Pearson correlation

Testing

In [10]:
```
corr_matrix = X.copy().corr()
```

In [11]:
```
corr_matrix
```

Out[11]:

	original_shape_Elongation	original_firstorder_Kurtc
original_shape_Elongation	1.000000	-0.157
original_firstorder_Kurtosis	-0.157728	1.000
original_firstorder_Skewness	-0.129982	0.869
wavelet-LHL_glrlm_RunVariance	-0.027935	-0.151
wavelet2-LHL_gldm_LargeDependenceEmphasis	-0.002718	-0.154
...	...	
wavelet2-LHH_firstorder_RobustMeanAbsoluteDeviation	-0.036797	0.138
original_shape_SphericalDisproportion	-0.360294	0.325
wavelet2-HHL_glcm_Idmn	-0.030738	-0.030
wavelet2-LHH_firstorder_RootMeanSquared	-0.027682	0.103
wavelet2-HLL_firstorder_Maximum	-0.021997	-0.036

150 rows × 150 columns

In [12]:

```
##The following code was also used by me in another group project for
Data Analytics in Health and Connected Care this semester

#This code uses the lower triangular portion of the correlation matrix
(as it is symmetrical and 1 along diagonal)
threshold = 0.9
high_corr = []
#will put indexes of features that are correlated with one another if
the absolute value is above the upper defined threshold
#this will make high_corr a nested list in which we get lists of
features correlated with each other
for i in range(1, corr_matrix.shape[0]):
    list_corr = [corr_matrix.index[i]]
    feat = corr_matrix.iloc[i, :i].index[corr_matrix.iloc[i, :i] >
threshold]
    list_corr = list(np.append(list_corr, feat))
    if(len(list_corr) > 1):
        high_corr.append(list_corr)

high_corr.sort(key=len) #sort the lists inside high corr according to
their length from low to high
                        #this is important for the following operation
below
to_remove=[]
#this code will check and remove sublists in high_corr
for i in range(len(high_corr)):
    for j in range(i+1, len(high_corr)):
        if set(high_corr[i]) <= set(high_corr[j]): #this line of code
checks if high_corr[i] is a sublist in high_corr[j]
            to_remove.append(high_corr[i]) #will put the list in an
array to remove it later (if removed now the length of high_corr will
change)
            break #break for loop and go to next index i
and cause the for loop to go out of range
for sublist in to_remove: #remove sublists
    high_corr.remove(sublist)


high_corr_copy = copy.deepcopy(high_corr) #create complete new deep
copy so as to not edit anything in high_corr
to_keep = []
```

```
#this code will go over all the lists of correlated features and put
the feature that has the lowest absolute total of correlation in a
list to keep
#that feature will then be removed from all other lists to avoid
possible duplicates
#all the other features that are not in the to_keep list will be
removed as they are considered to be represented by the features in
the to_keep list
while high_corr_copy != []:
    feature_list = high_corr_copy.pop(0) #this list contains the
features correlated to each other and is also removed from
high_corr_copy
    sum_corr = [sum(np.abs(corr_matrix.loc[:, feature])) for feature
in feature_list] #total sum is calculated for each feature in
feature_list

    idx = np.argmin(sum_corr)
    unique = feature_list[idx] #this feature has the lowest absolute
total correlation and will represent all the other features it is
highly
                               #correlated with
    to_keep.append(unique)
    #the code below will remove the feature from all other lists of
correlated features in high_corr_copy
    for feature_list in high_corr_copy:
        for feature in feature_list:
            if feature == unique:
                feature_list.remove(unique)
            if feature_list == []: #if ever the unique features
selected have made a list empty then it should be removed
                high_corr_copy.remove(feature_list)
remove_feat = set([])
for l in high_corr: #all features in high_corr that are not in the
to_keep list are, will be put in a remove_feat so that they can be
removed
    for feat in l:  #from the dataset
        if (feat not in to_keep):
            remove_feat.add(feat)
print(f'Initial feature amount: {len(X.columns)}')
print(f'Features to remove: {len(remove_feat)}')
print(f'New feature amount: {150-len(remove_feat)}')
```

```
Initial feature amount: 150
Features to remove: 48
New feature amount: 102
```

This code will now be put in a class so that it contains the "fit" and "transform" method allowing it to be used by the sklearn Pipeline object. This is important as the code above has been used on the whole dataset, it should only be used on the train dataset and not the test dataset.

In [7]:

```
#fit code is explained in the cell above
class PearsonFeatureReduction:
    def __init__(self, threshold=0.9):
        self.threshold = threshold
        self.remove_feat = set([])

    def fit(self, X: pd.DataFrame, *args, **kwargs): #*args and
**kwargs are put there as when the pipeline is fitted y_train will be
passed as an
        corr_matrix = X.copy().corr()                #argument, thus
it will also be passed in the methods of PearsonFeatureReduction
        high_corr = []                               #and y_train does
nothing in these methods so *args and **kwargs allows y_train to be
passed
                                                     #in the function
and do nothing
        for i in range(1, corr_matrix.shape[0]):
            list_corr = [corr_matrix.index[i]]
            feat = corr_matrix.iloc[i, :i].index[corr_matrix.iloc[i,
:i] > self.threshold]
            list_corr = list(np.append(list_corr, feat))
            if(len(list_corr) > 1):
                high_corr.append(list_corr)

        high_corr.sort(key=len)
        to_remove=[]

        for i in range(len(high_corr)):
            for j in range(i+1, len(high_corr)):
                if set(high_corr[i]) <= set(high_corr[j]):
                    to_remove.append(high_corr[i])
                    break
        for sublist in to_remove:
            high_corr.remove(sublist)

        high_corr_copy = copy.deepcopy(high_corr)
        to_keep = []
        while high_corr_copy != []:
            feature_list = high_corr_copy.pop(0)
            sum_corr = [sum(np.abs(corr_matrix.loc[:, feature])) for
feature in feature list]
```

```
            idx = np.argmin(sum_corr)
            unique = feature_list[idx]
            to_keep.append(unique)

            for feature_list in high_corr_copy:
                for feature in feature_list:
                    if feature == unique:
                        feature_list.remove(unique)
                    if feature_list == []:
                        high_corr_copy.remove(feature_list)
        self.remove_feat.clear()
        for l in high_corr:
            for feat in l:
                if (feat not in to_keep):
                    self.remove_feat.add(feat)
        return

    def transform(self, X: pd.DataFrame, *args, **kwargs):
        return X.drop(columns=self.remove_feat).copy() #removes the
features from the dataset

    def fit_transform(self, X: pd.DataFrame, *args, **kwargs):
        self.fit(X)
        return self.transform(X)
    def set_params(*args, **kwargs):
        return
```

First task: classification of individual microcalcifications

In [8]:
```
##SOURCE: https://stackoverflow.com/questions/54797508/how-to-generate-a-train-test-split-based-on-a-group-id

#make train and test split, train split will be scored with cross-validation and that is why no validation set is defined
splitter = GroupShuffleSplit(test_size=.20, n_splits=2, random_state = 0)
split = splitter.split(X, y, groups=X.index) #allows us to split the data according to the patient ID
train_inds, test_inds = next(split)

X_train, y_train = [X.iloc[train_inds].copy(),
y.iloc[train_inds].copy()]
X_test, y_test = [X.iloc[test_inds].copy(), y.iloc[test_inds].copy()]
```

In [9]:
```
pat_train = len(set(X_train.index))
pat_test = len(set(X_test.index))
total = pat_train + pat_test
print(f"Patient ID training/test split: {round(pat_train/total, 2)}/{round(pat_test/total, 2)}")
MCs_train = len(X_train)
MCs_test = len(X_test)
total = MCs_train + MCs_test
print(f"MCs or row training/test split: {round(MCs_train/total, 2)}/{round(MCs_test/total, 2)}")
```

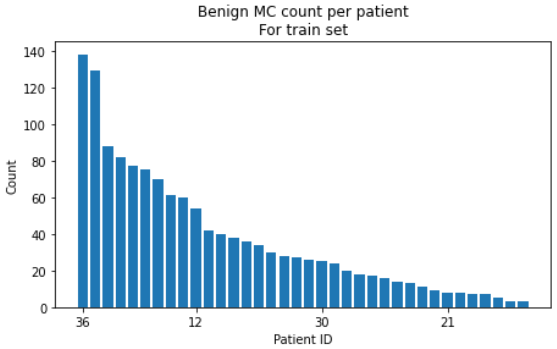
```
Patient ID training/test split: 0.79/0.21
MCs or row training/test split: 0.74/0.26
```

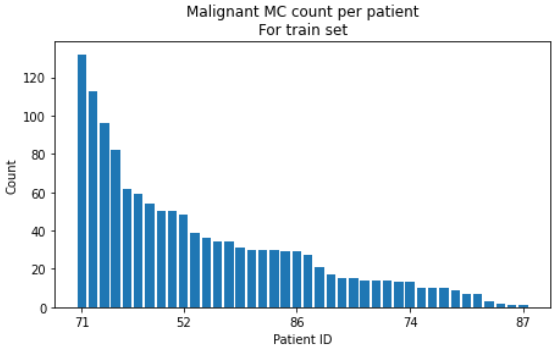
In [13]:
```
#check that no patient id is in both train and test
for val in set(X_test.index):
    print(val in set(X_train.index))
```

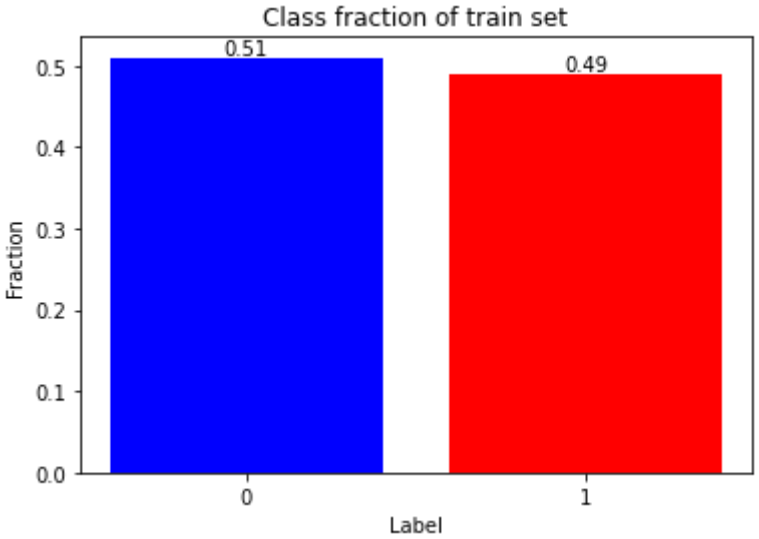
```
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
```

In [9]:

```
fig, ax = plt.subplots(1, 2, figsize=(16, 4))
label_pat = y_train.groupby('Patient
ID').value_counts().sort_values(ascending=False) #this will give the
label count per patient ID
pat_0 = label_pat[[x[1] == 0 for x in label_pat.index]] #this list
gives the label 0 count per patient ID
pat_1 = label_pat[[x[1] == 1 for x in label_pat.index]] #this list
gives the label 1 count per patient ID
ax[0].bar(list(range(len(pat_0))), pat_0)
ticks = np.array([0]+list(range(9, 30, 10)))
ax[0].set_xticks(ticks=ticks)
ax[0].set_xticklabels([x[0] for x in pat_0.iloc[ticks].index])
ax[0].set_xlabel('Patient ID')
ax[0].set_ylabel('Count')
ax[0].set_title('Benign MC count per patient\nFor train set')
ax[1].bar(list(range(len(pat_1))), pat_1)
ticks = np.array([0]+list(range(9, 40, 10)))
ax[1].set_xticks(ticks=ticks)
ax[1].set_xticklabels([x[0] for x in pat_1.iloc[ticks].index])
ax[1].set_xlabel('Patient ID')
ax[1].set_ylabel('Count')
ax[1].set_title('Malignant MC count per patient\nFor train set')
plt.show()

plt.figure()
plt.title('Class fraction of train set')
label_0_frac = round((pat_0.sum())/(pat_0.sum()+pat_1.sum()), 2)
label_1_frac = round((pat_1.sum())/(pat_0.sum()+pat_1.sum()), 2)
bars = plt.bar([0, 1], [label_0_frac, label_1_frac], color=['b', 'r'])
plt.bar_label(bars)
plt.xticks(ticks=[0, 1], labels=[0,1])
plt.xlabel('Label')
plt.ylabel('Fraction')
plt.show()
```


Benign MC count per patient
For train set
Count
Patient ID
36
12
30
21


Malignant MC count per patient
For train set
Count
Patient ID
71
52
86
74
87


Class fraction of train set
0.51
0.49
Fraction
Label
0
1

In [10]:

```
##all the functions in this cell have also been used by me in that
same group project for Data Analytics in Health and Connected Care
##which was mentioned earlier

def get_mean_cv_rmse(pipeline, X, y, scoring:
str='neg_root_mean_squared_error', cv: int = 3, **kwargs) -> float:
    rmse = -1*cross_val_score(pipeline, X, y, scoring=scoring, cv=cv,
**kwargs) #-> https://www.kaggle.com/code/alexisbcook/cross-validation
    return np.mean(rmse)

def plot_metrics(y_true, y_pred, y_prob_score=None, title: str='Test
confusion matrix'):
    #confusion matrix
    conf_matrix = sklearn.metrics.confusion_matrix(y_true, y_pred)
    cm_disp =
sklearn.metrics.ConfusionMatrixDisplay(conf_matrix).plot()
    plt.title(title)
    plt.show()
    #Sensitivity and specificity
    tn, fp, fn, tp = conf_matrix.ravel() #-> shown in https://scikit-
learn.org/stable/modules/generated
/sklearn.metrics.confusion_matrix.html
    sensitivity = tp / (tp + fn)
    specificity = tn / (tn + fp)
    print("Sensitivity:", sensitivity,"\nSpecificity:", specificity)
    print("Accuracy score:", accuracy_score(y_true, y_pred))

    if(y_prob_score is None):
        return
    #ROC curve
    fpr, tpr, _ = roc_curve(y_true, y_prob_score)
    roc_auc = auc(fpr, tpr)
    plt.figure()
    plt.plot(fpr, tpr,
         label='ROC curve (AUC = {0:0.2f})'
               ''.format(roc_auc))
    plt.plot([0, 1], [0, 1], 'k--', label='Random classifier')
    plt.legend()
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
```

```
    plt.ylabel('True Positive Rate')
    plt.show()

    return
#plots the mean cross-validated RMSEs in a bar plot
def compare_RMSEs(RMSEs, labels, title: str='Mean RMSE of cross
validation'):
    plt.figure(figsize=(10, 6))
    plt.title(title)
    x = np.arange(len(RMSEs))

    bars = plt.bar(x, RMSEs)
    plt.bar_label(bars)
    plt.xticks(ticks=x, labels=labels)
    plt.ylabel("RMSE")
    plt.show()
    return
```

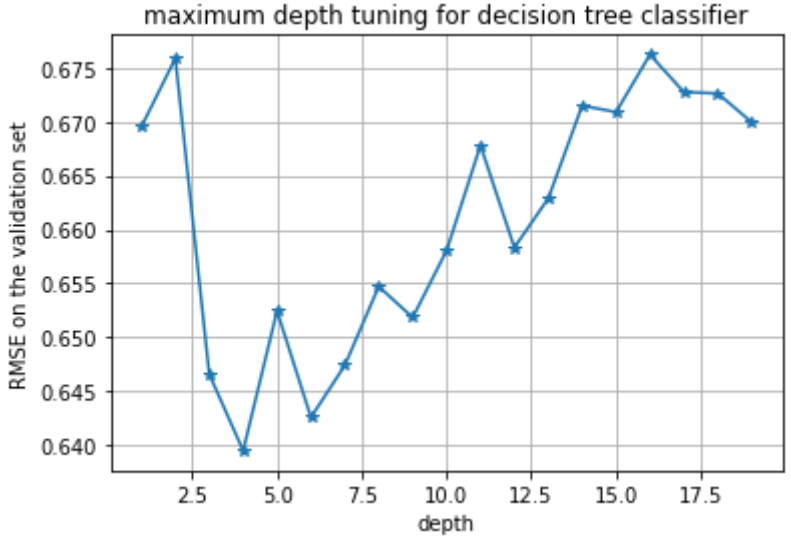
Decision tree classifier

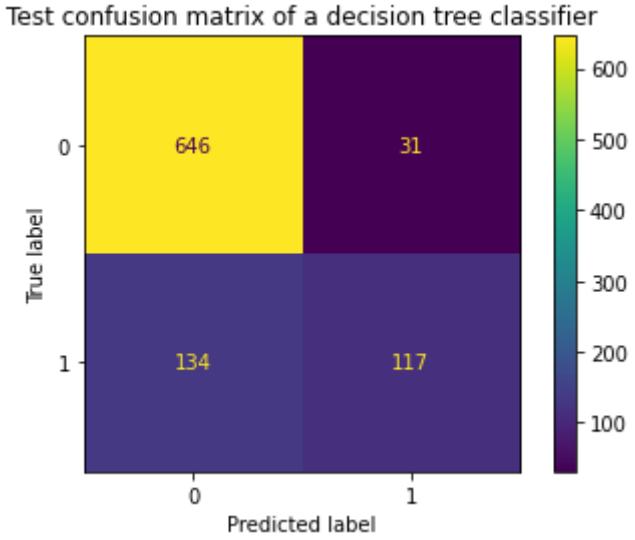
Tuning max depth parameter

In [141…

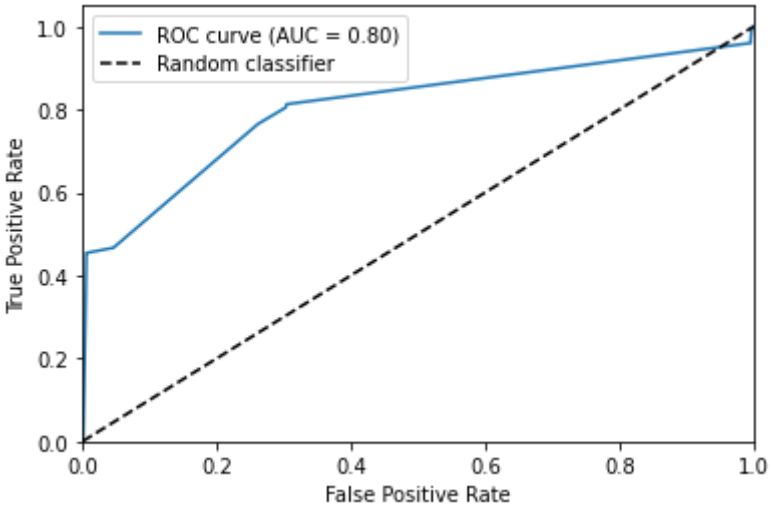
```
valRMSE=[]
depth = np.arange(1, 20, 1)
for parameter in depth:
    clf = DecisionTreeClassifier(random_state = 0,
max_depth=parameter)
    error = get_mean_cv_rmse(clf, X_train, y_train)
    valRMSE.append(error)

plt.plot(depth, valRMSE,'-*')
plt.xlabel("depth")
plt.ylabel("RMSE on the validation set")
plt.title('maximum depth tuning for decision tree classifier')
plt.grid()
#Choose the best parameter value
minindex = np.argmin(valRMSE)
optimal_depth = depth[minindex]
print(f'Chosen parameter: {optimal_depth}')

tree = DecisionTreeClassifier(random_state = 0,
max_depth=optimal_depth)
tree.fit(X_train,y_train)
prediction = tree.predict(X_test)
prob_score = tree.predict_proba(X_test)[:, 1]

plot_metrics(y_test, prediction, prob_score, 'Test confusion matrix of
a decision tree classifier')
```

```
Chosen parameter: 4
```






```
Sensitivity: 0.46613545816733065
Specificity: 0.9542097488921714
Accuracy score: 0.822198275862069
```



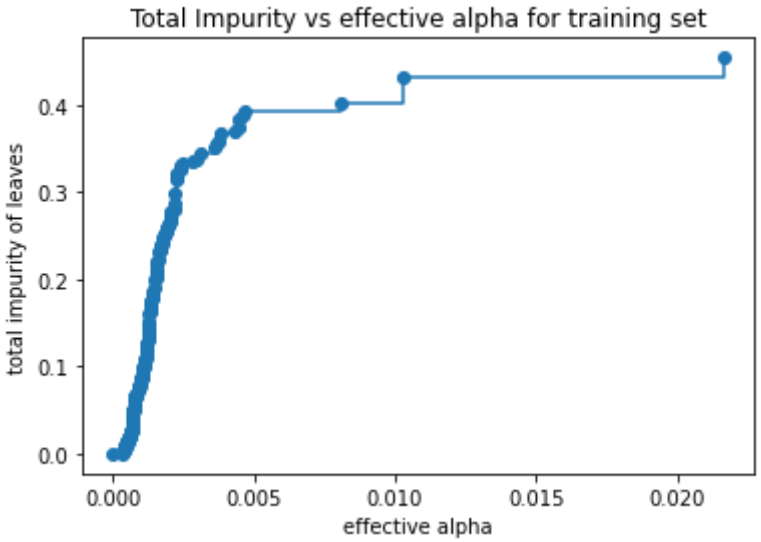
Tuning ccp_alpha parameter

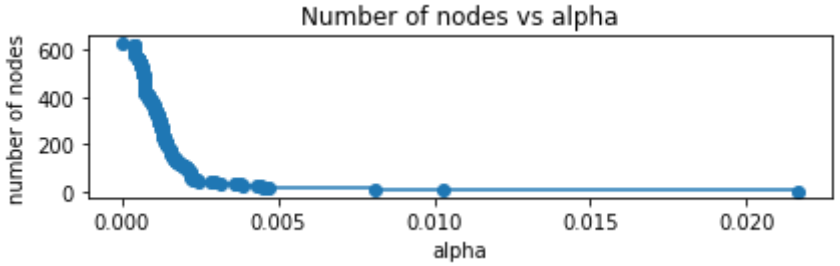
In [114…

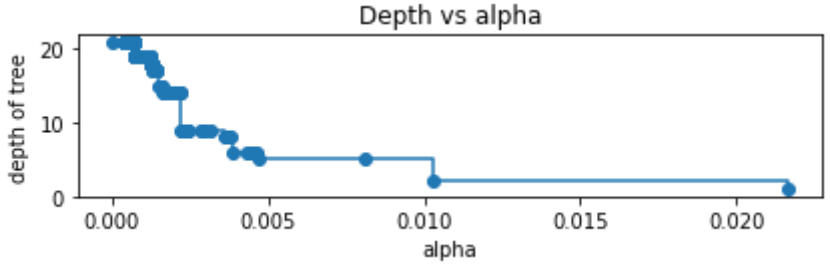
```
##SOURCE: https://scikit-learn.org/stable/auto_examples
/tree/plot_cost_complexity_pruning.html
#pruning
clf = DecisionTreeClassifier(random_state=0)
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

fig, ax = plt.subplots()
ax.plot(ccp_alphas[:-1], impurities[:-1], marker="o",
drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")

clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)
print(
    "Number of nodes in the last tree is: {} with ccp_alpha:
{}".format(
        clfs[-1].tree_.node_count, ccp_alphas[-1]
    )
)

clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

node_counts = [clf.tree_.node_count for clf in clfs]
depth = [clf.tree_.max_depth for clf in clfs]
fig, ax = plt.subplots(2, 1)
ax[0].plot(ccp_alphas, node_counts, marker="o", drawstyle="steps-
post")
ax[0].set_xlabel("alpha")
ax[0].set_ylabel("number of nodes")
ax[0].set_title("Number of nodes vs alpha")
ax[1].plot(ccp_alphas, depth, marker="o", drawstyle="steps-post")
ax[1].set_xlabel("alpha")
ax[1].set_ylabel("depth of tree")
ax[1].set_title("Depth vs alpha")
```

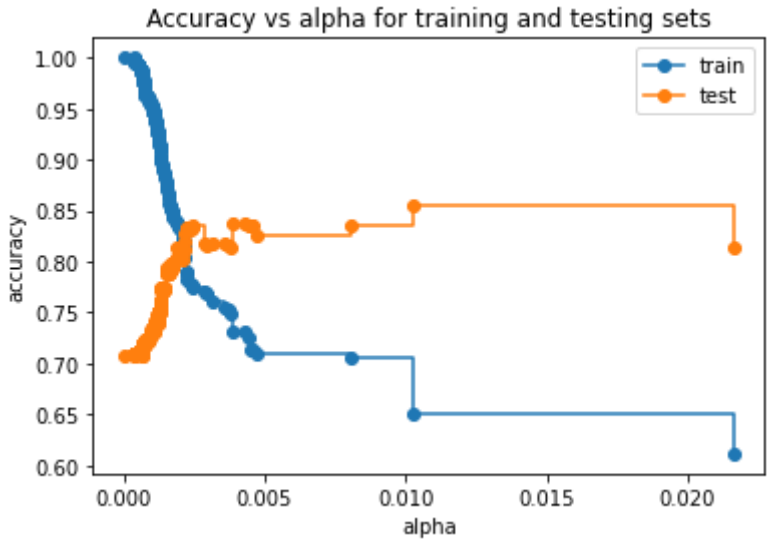
```
fig.tight_layout()

train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]

fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker="o", label="train",
drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker="o", label="test",
drawstyle="steps-post")
ax.legend()
plt.show()
```

```
Number of nodes in the last tree is: 1 with ccp_alpha: 0.04602366870989438
```





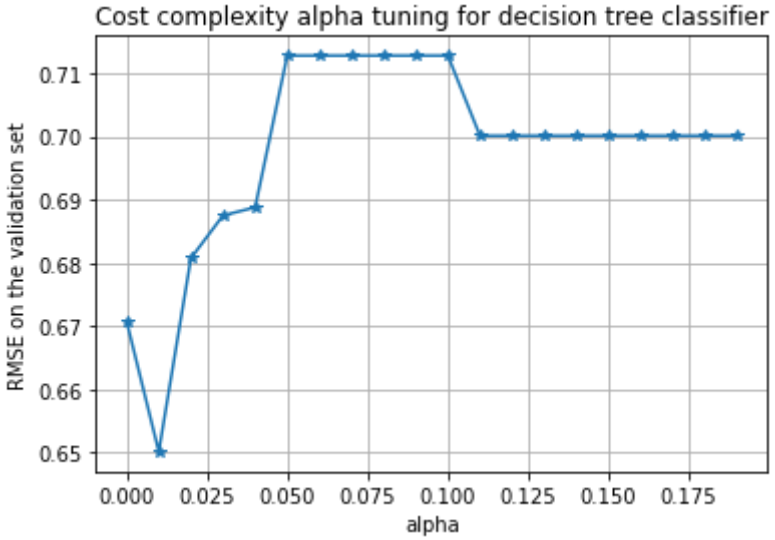


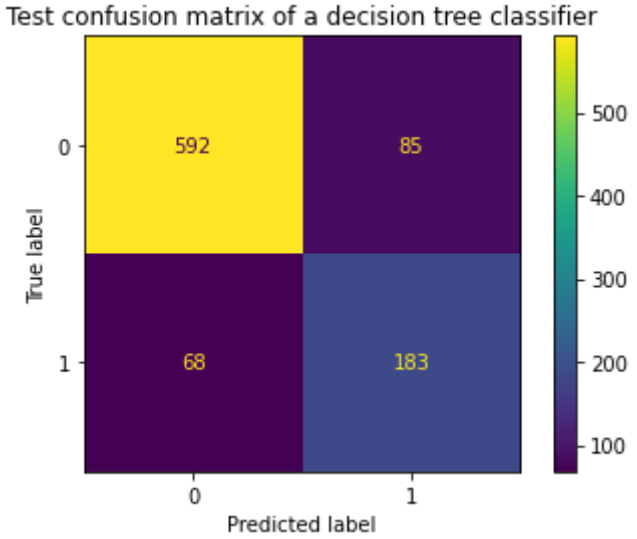


In [140…

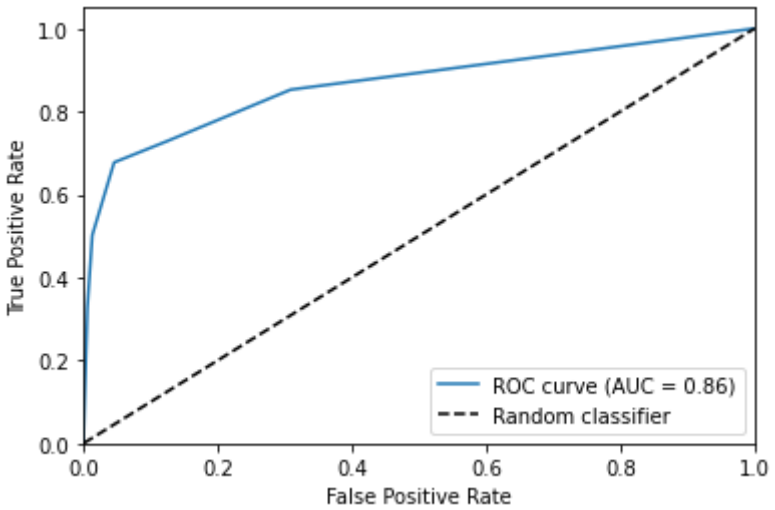
```
valRMSE=[]
alpha = 0.01*np.arange(0, 20, 1)
for parameter in alpha:
    clf = DecisionTreeClassifier(random_state = 0,
ccp_alpha=parameter)
    error = get_mean_cv_rmse(clf, X_train, y_train)
    valRMSE.append(error)

plt.plot(alpha, valRMSE,'-*')
plt.xlabel("alpha")
plt.ylabel("RMSE on the validation set")
plt.title('Cost complexity alpha tuning for decision tree classifier')
plt.grid()
#Choose the best parameter value
minindex = np.argmin(valRMSE)
optimal_alpha = alpha[minindex]
print(f'Chosen parameter: {optimal_alpha}')

tree = DecisionTreeClassifier(random_state = 0,
ccp_alpha=optimal_alpha)
tree.fit(X_train,y_train)
prediction = tree.predict(X_test)
prob_score = tree.predict_proba(X_test)[:, 1]

plot_metrics(y_test, prediction, prob_score, 'Test confusion matrix of
a decision tree classifier')
```

```
Chosen parameter: 0.01
```





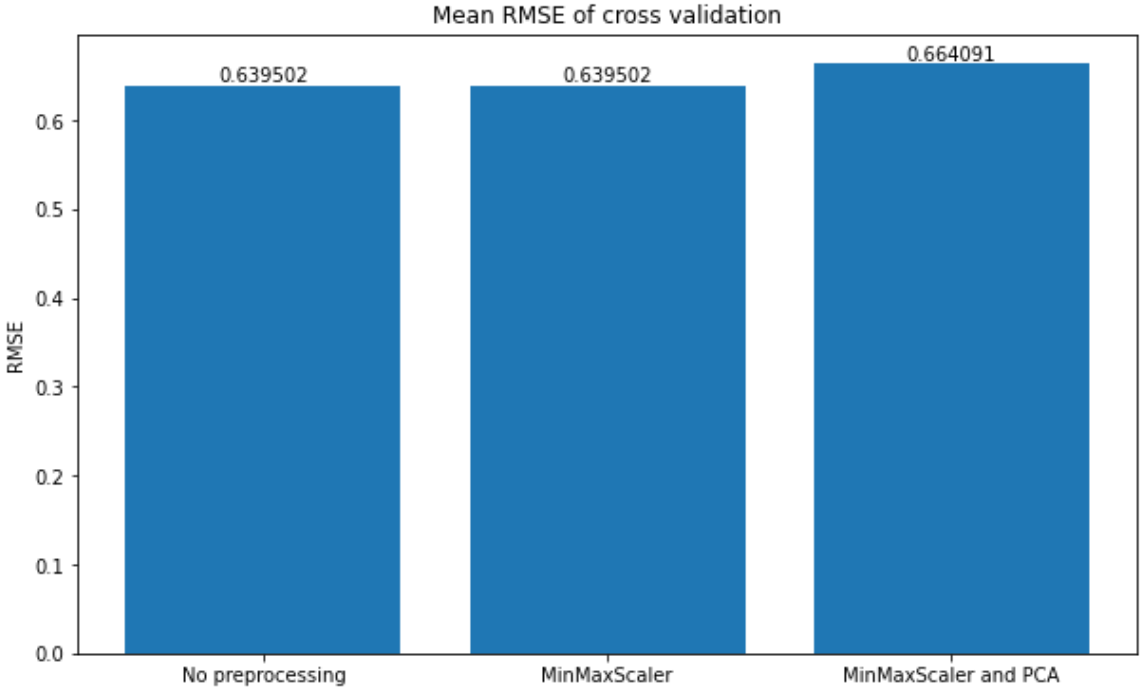

```
Sensitivity: 0.7290836653386454
Specificity: 0.8744460856720827
Accuracy score: 0.8351293103448276
```

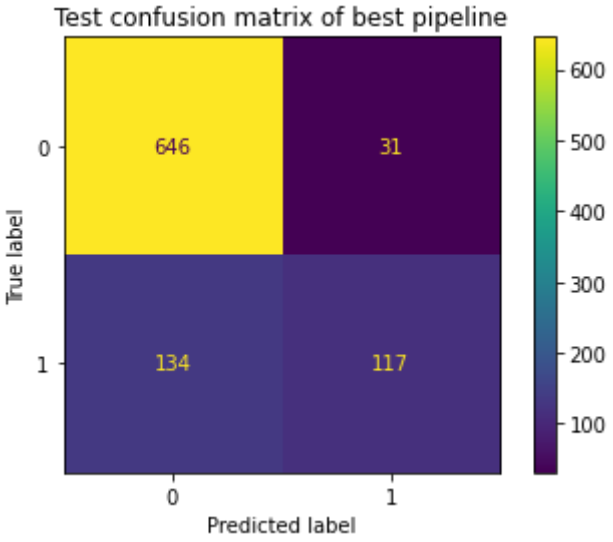


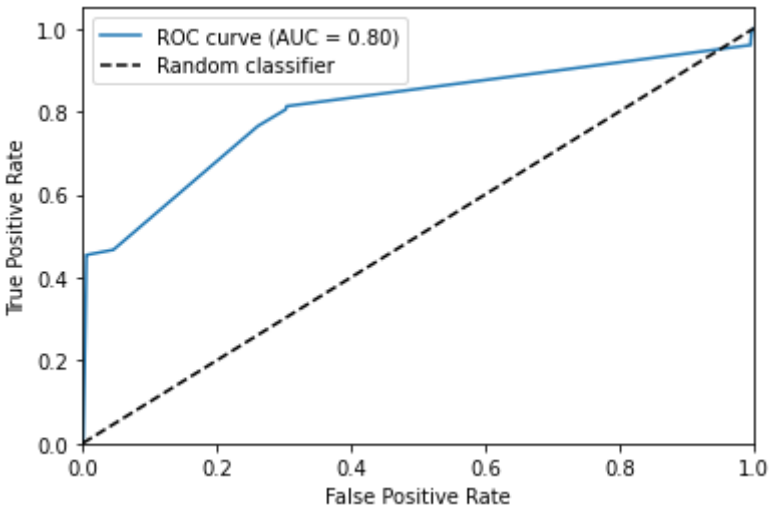
Applying preprocessing: MinMax and PCA

In [20]:

```
mean_cv_rmse = []
pipelines = []
#no preprocessing
tree = DecisionTreeClassifier(random_state = 0, max_depth =
optimal_depth)
pipelines.append(tree)
score = get_mean_cv_rmse(tree, X_train, y_train)
mean_cv_rmse.append(score)
#minmax
pipeline_tree = Pipeline([('MinMaxScaler', MinMaxScaler((0,1))),
('model', tree)]) #Pipeline can sequentially apply a list of
transforms and then at
pipelines.append(pipeline_tree)
#the end an estimator, the transformers need to have
score = get_mean_cv_rmse(pipeline_tree, X_train, y_train)
#the method fit and transform
mean_cv_rmse.append(score)
#minmax and PCA
pipeline_tree = Pipeline([('MinMaxScaler', MinMaxScaler((0,1))),
('PCA', PCA(random_state=0)),('model', tree)])
params={'PCA__n_components': np.arange(1, 151, dtype=int)}
grid_search_pca = GridSearchCV(pipeline_tree, param_grid=params)
grid_search_pca.fit(X_train, y_train)
pipeline_tree.set_params(**grid_search_pca.best_params_)
pipelines.append(pipeline_tree)
score = get_mean_cv_rmse(pipeline_tree, X_train, y_train)
mean_cv_rmse.append(score)

compare_RMSEs(mean_cv_rmse, labels=["No preprocessing",
"MinMaxScaler", "MinMaxScaler and PCA"])

best_tree_pipeline = pipelines[np.argmin(mean_cv_rmse)]
best_tree_pipeline.fit(X_train, y_train)
prediction = best_tree_pipeline.predict(X_test)
prob_score = best_tree_pipeline.predict_proba(X_test)[:,1]

plot_metrics(y_test, prediction, prob_score, 'Test confusion matrix of
best pipeline')
```








```
Sensitivity: 0.46613545816733065
Specificity: 0.9542097488921714
Accuracy score: 0.822198275862069
```

Parameter tuning using GridSearchCV

In [21]:
```
##### Tuning using gridsearchcv #####
parameters = {'criterion': ['gini', 'entropy', 'log_loss'],
              'max_depth': np.arange(1, 20, 1),
              'splitter': ['best', 'random'],
              'max_features': ['sqrt', 'log2', None]
             }
grid_search = GridSearchCV(DecisionTreeClassifier(random_state=0),
verbose=2, param_grid = parameters, cv = 3, n_jobs=-1)
```

In [22]:
```
grid_search.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 342 candidates, totalling 1026 fits
```

Out[22]:

▸ GridSearchCV

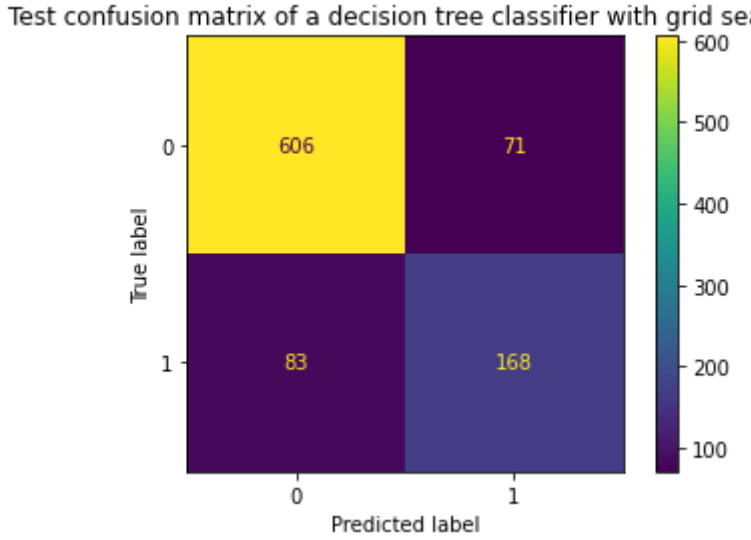
▸ estimator: DecisionTreeClassifier

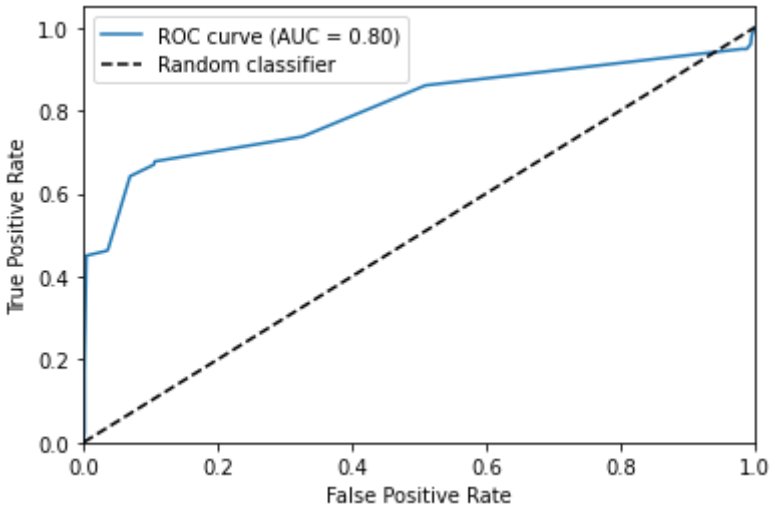
▸ DecisionTreeClassifier

In [23]:
```
grid_search.best_params_
```

Out[23]:
```
{'criterion': 'gini',
 'max_depth': 5,
 'max_features': None,
 'splitter': 'random'}
```

In [24]:
```
tree_grid_search = DecisionTreeClassifier(random_state = 0,
                                          criterion =
grid_search.best_estimator_.criterion,
                                          max_depth =
grid_search.best_estimator_.max_depth,
                                          max_features =
grid_search.best_estimator_.max_features)
tree_grid_search.fit(X_train,y_train)
prediction = tree_grid_search.predict(X_test)
prob_score = tree_grid_search.predict_proba(X_test)[:,1]
plot_metrics(y_test, prediction, prob_score, 'Test confusion matrix of
a decision tree classifier with grid search')
```

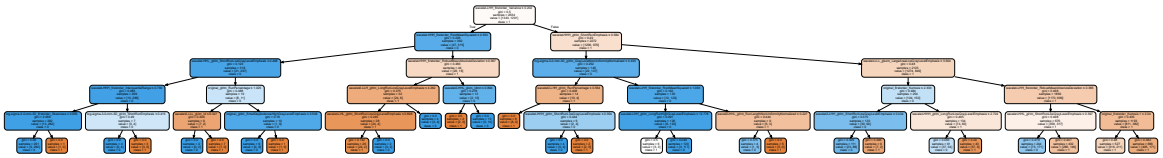





```
Sensitivity: 0.6693227091633466
Specificity: 0.8951255539143279
Accuracy score: 0.834051724137931
```

In [25]:
```
#SOURCE:
#https://www.kaggle.com/code/funxexcel/p2-decision-tree-
hyperpratameter-tuning-python/notebook
#https://stackoverflow.com/questions/35064304/runtimeerror-make-sure-
the-graphviz-executables-are-on-your-systems-path-aft
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz
/bin/'

dot_data = export_graphviz(tree_grid_search,
                           out_file = None,
                           feature_names = X.columns,        #Provide X
Variables Column Names
                           class_names = ['1','0'],       # Provide Target
Variable Column Name
                           filled = True, rounded=True,      # Controls the
look of the nodes and colours it
                           special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

Out[25]:

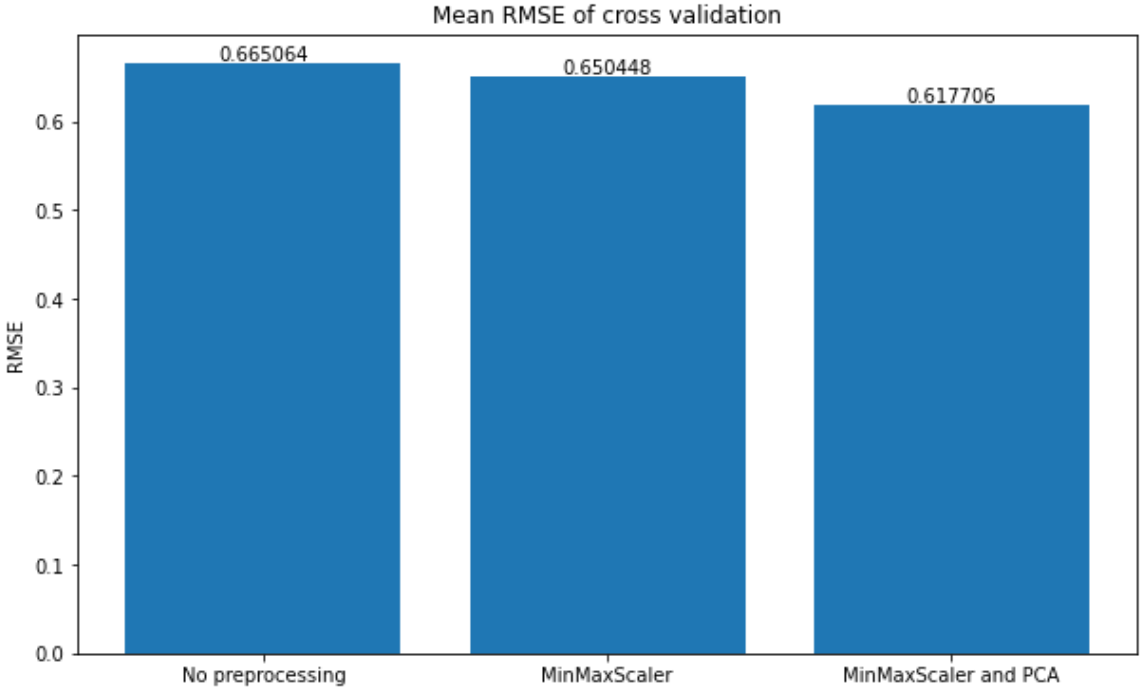


Naive Bayes classifier

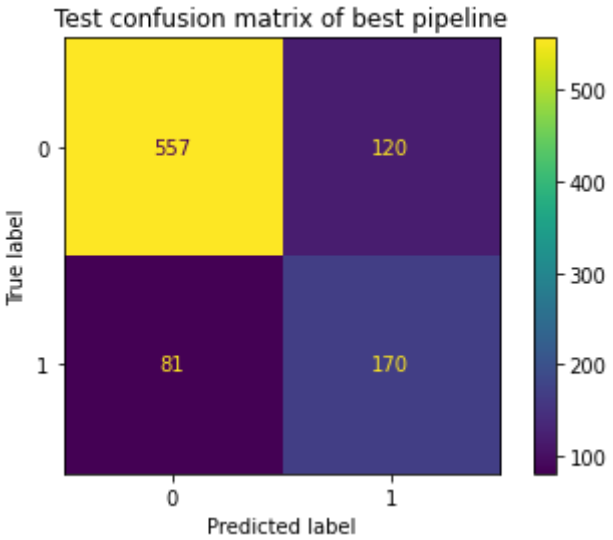
Applying preprocessing: MinMax and PCA

In [112…

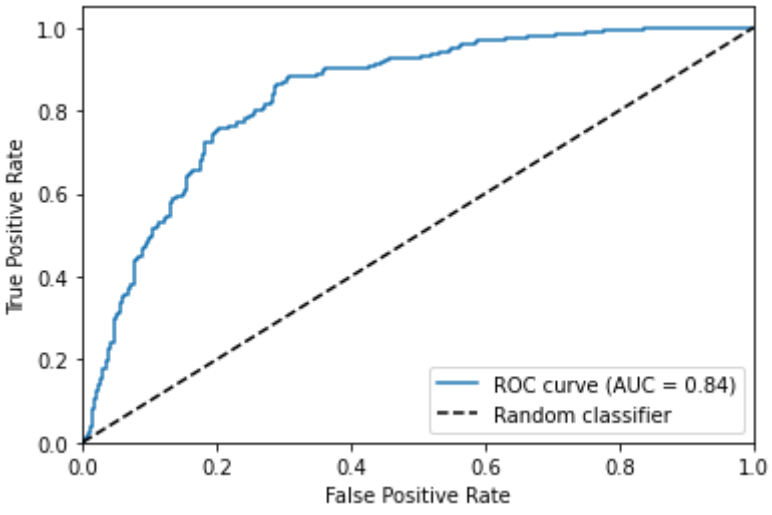
```
mean_cv_rmse = []
pipelines = []
#no preprocessing
bayes = GaussianNB()
pipelines.append(Pipeline([('model', bayes)]))
score = get_mean_cv_rmse(bayes, X_train, y_train)
mean_cv_rmse.append(score)
#minmax
pipeline_bayes = Pipeline([('MinMax', MinMaxScaler((0,1))), ('model',
bayes)])
pipelines.append(pipeline_bayes)
score = get_mean_cv_rmse(pipeline_bayes, X_train, y_train)
mean_cv_rmse.append(score)
#minmax and PCA
pipeline_bayes = Pipeline([('MinMax', MinMaxScaler((0,1))), ('PCA',
PCA(random_state=0)),('model', bayes)])
params={'PCA__n_components': np.arange(1, 151, dtype=int)}
grid_search_pca = GridSearchCV(pipeline_bayes, param_grid=params,
cv=3)
grid_search_pca.fit(X_train, y_train)
pipeline_bayes.set_params(**grid_search_pca.best_params_)

pipelines.append(pipeline_bayes)
score = get_mean_cv_rmse(pipeline_bayes, X_train, y_train)
mean_cv_rmse.append(score)

compare_RMSEs(mean_cv_rmse, labels=["No preprocessing",
"MinMaxScaler", "MinMaxScaler and PCA"])

best_bayes_pipeline = pipelines[np.argmin(mean_cv_rmse)]
best_bayes_pipeline.fit(X_train, y_train)

prediction = best_bayes_pipeline.predict(X_test)
prob_score = best_bayes_pipeline.predict_proba(X_test)[:,1]
print('Best pipeline:', str(best_bayes_pipeline.get_params()
['steps']))
plot_metrics(y_test, prediction, prob_score, 'Test confusion matrix of
best pipeline')
```




```
Best pipeline: [('MinMax', MinMaxScaler()), ('PCA', PCA(n_components=29, random_
state=0)), ('model', GaussianNB())]
```



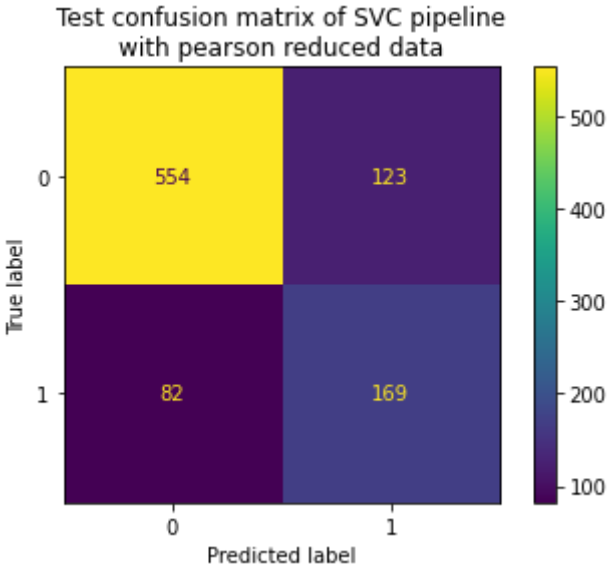

```
Sensitivity: 0.6772908366533864
Specificity: 0.8227474150664698
Accuracy score: 0.7834051724137931
```



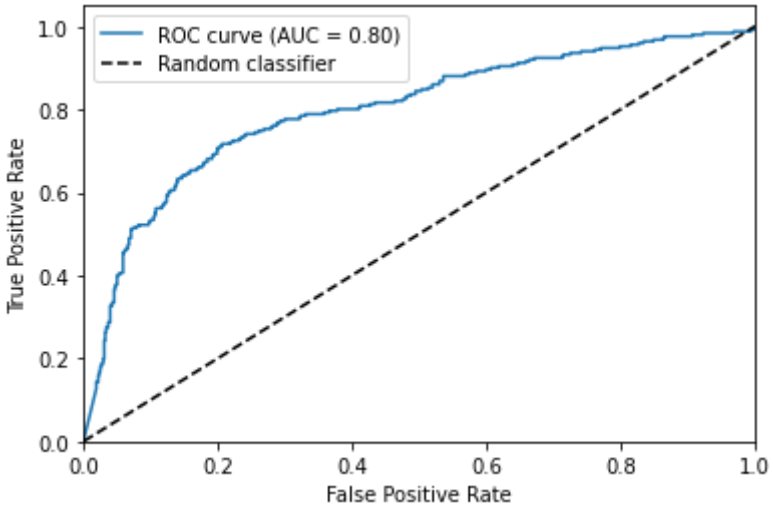
Applying preprocessing: Pearson correlation feature reduction

In [53]:
```
bayes = GaussianNB()
pipeline_bayes = Pipeline([('pearson', PearsonFeatureReduction()),
('MinMaxScaler', MinMaxScaler((0,1))), ('model', bayes)])
parameters = {'pearson__threshold': .01*np.arange(60, 100, 5)}
grid_search = GridSearchCV(pipeline_bayes, param_grid = parameters,
verbose=1, cv=3)
grid_search.fit(X, y)
print(grid_search.best_params_)
pipeline_bayes.set_params(**grid_search.best_params_)
pipeline_bayes.fit(X_train, y_train)

prediction = pipeline_bayes.predict(X_test)
prob_score = pipeline_bayes.predict_proba(X_test)[:,1]
plot_metrics(y_test, prediction, prob_score, 'Test confusion matrix of
GaussianNB pipeline\nwith pearson reduced data')
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
{'pearson__threshold': 0.6}
```




```
Sensitivity: 0.6733067729083665
Specificity: 0.8183161004431314
Accuracy score: 0.7790948275862069
```

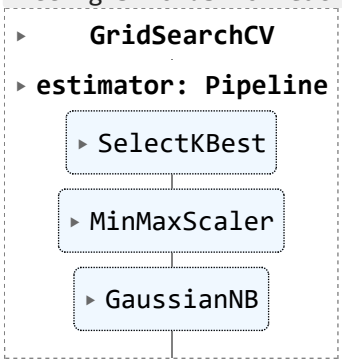


Applying preprocessing: SelectKBest

In [59]:
```
#gridsearch will be performed for the best pipeline above to further
improve the accuracy
bayes = GaussianNB()
pipeline_bayes = Pipeline([('KBest',
SelectKBest(mutual_info_classif)), ('MinMaxScaler',
MinMaxScaler((0,1))), ('model', bayes)])
params = {'KBest__k': np.arange(1, 151, dtype=int)}
grid_search = GridSearchCV(pipeline_bayes, param_grid=params, cv=3,
verbose=1, n_jobs=-1)
grid_search.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 150 candidates, totalling 450 fits
```

Out[59]:



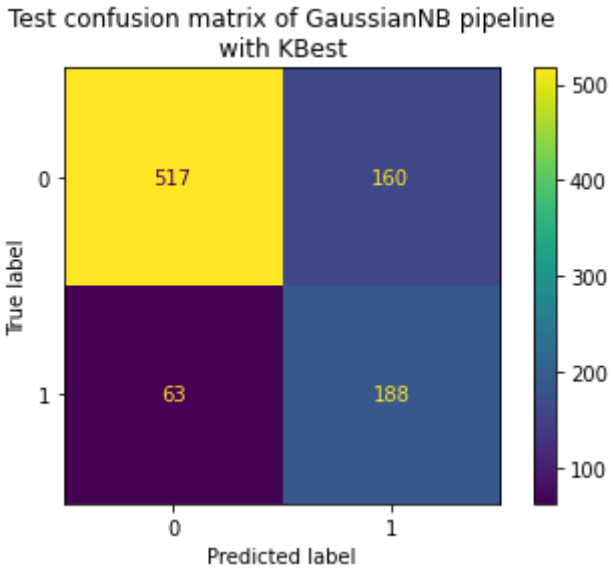
In [78]:
```
print(grid_search.best_params_)
kbest = SelectKBest(mutual_info_classif, k =
grid_search.best_params_['KBest__k'])
kbest.fit(X_train, y_train)
print(kbest.get_feature_names_out())
```

```
{'KBest__k': 2}
['wavelet-LHH_firstorder_InterquartileRange'
 'wavelet-LHH_firstorder_MeanAbsoluteDeviation']
```

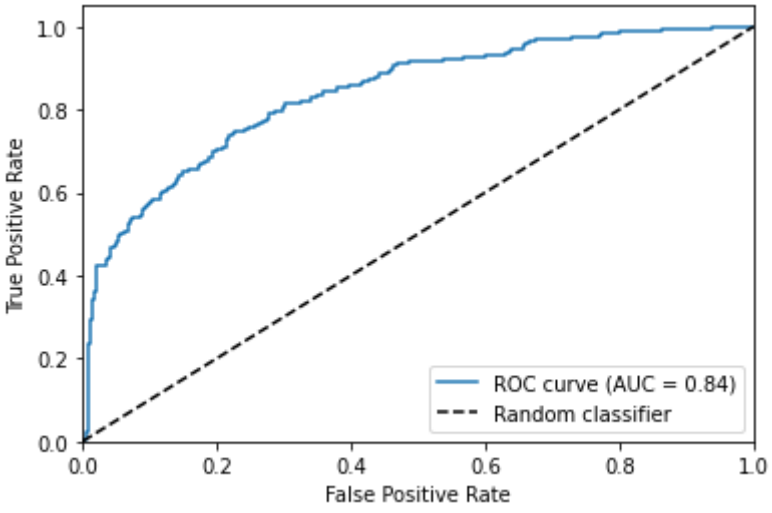
In [62]:
```
k = grid_search.best_params_['KBest__k']
bayes = GaussianNB()
pipeline_bayes = Pipeline([('KBest', SelectKBest(mutual_info_classif,
k=k)), ('MinMaxScaler', MinMaxScaler((0,1))),
                                ('model', bayes)])
```

In [63]:
```
pipeline_bayes.fit(X_train, y_train)
prediction = pipeline_bayes.predict(X_test)
prob_score = pipeline_bayes.predict_proba(X_test)[:,1]
print('Best pipeline:', str(pipeline_bayes.get_params()['steps']))
plot_metrics(y_test, prediction, prob_score, 'Test confusion matrix of
GaussianNB pipeline\nwith KBest')
```

```
Best pipeline: [('KBest', SelectKBest(k=2,
           score_func=<function mutual_info_classif at 0x000001E3AD68E8B0>)),
('MinMaxScaler', MinMaxScaler()), ('model', GaussianNB())]
```




```
Sensitivity: 0.749003984063745
Specificity: 0.7636632200886263
Accuracy score: 0.759698275862069
```

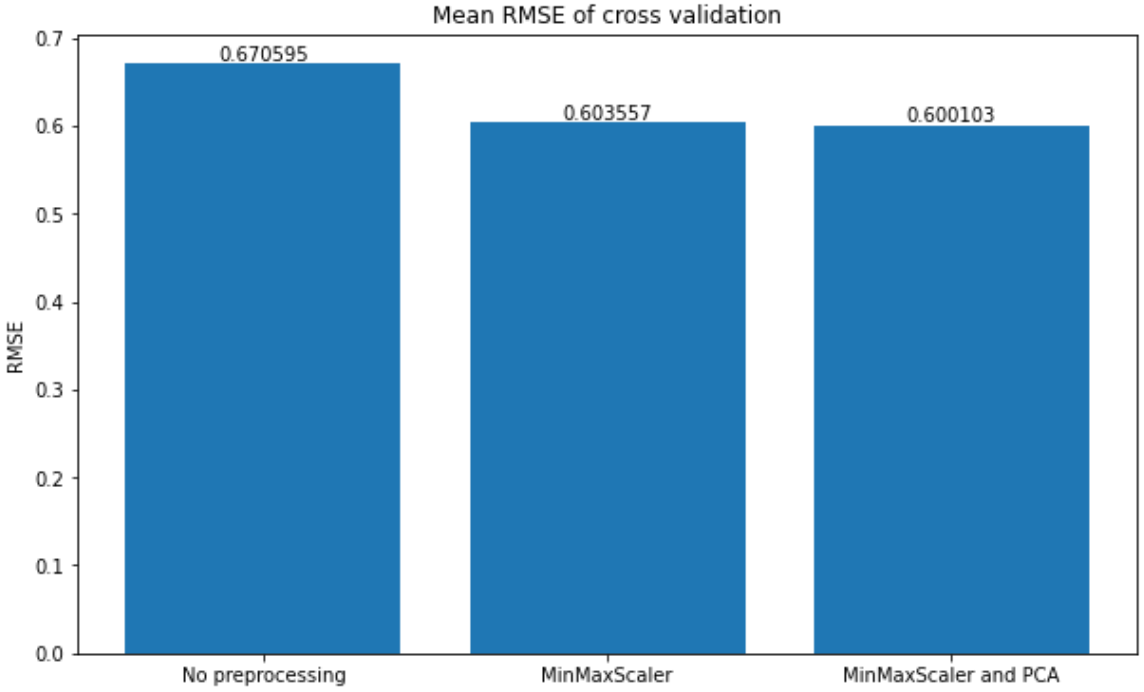


Support vector machines (C-Support Vector Classification)

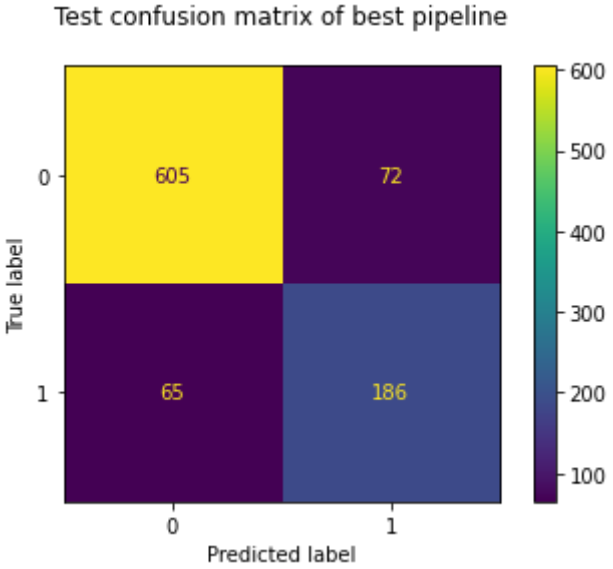
Applying preprocessing: MinMax and PCA

In [159…

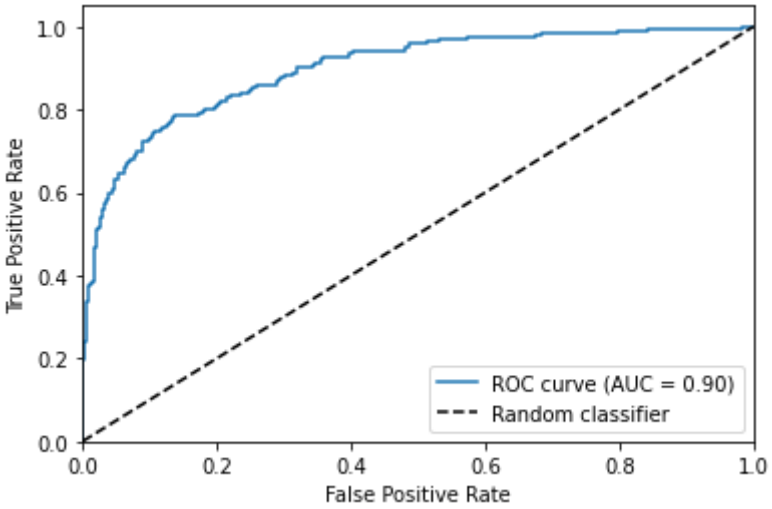
```
mean_cv_rmse = []
pipelines = []
#no preprocessing
svc = SVC(random_state=0, probability=True)
pipelines.append(Pipeline([('model', svc)]))
score = get_mean_cv_rmse(svc, X_train, y_train)
mean_cv_rmse.append(score)
#minmax
pipeline_svc = Pipeline([('MinMaxScaler', MinMaxScaler((0,1))),
('model', svc)])
pipelines.append(pipeline_svc)
score = get_mean_cv_rmse(pipeline_svc, X_train, y_train)
mean_cv_rmse.append(score)
#minmax and PCA
pipeline_svc = Pipeline([('MinMaxScaler', MinMaxScaler((0,1))),
('PCA', PCA(random_state=0)),('model', svc)])
params={'PCA__n_components': np.arange(1, 151, dtype=int)}
grid_search_pca = GridSearchCV(pipeline_svc, param_grid=params, cv=3)
grid_search_pca.fit(X_train, y_train)
pipeline_svc.set_params(**grid_search_pca.best_params_)
score = get_mean_cv_rmse(pipeline_svc, X_train, y_train)
pipelines.append(pipeline_svc)
mean_cv_rmse.append(score)

compare_RMSEs(mean_cv_rmse, labels=["No preprocessing",
"MinMaxScaler", "MinMaxScaler and PCA"])

best_svc_pipeline = pipelines[np.argmin(mean_cv_rmse)]
best_svc_pipeline.fit(X_train, y_train)
prediction = best_svc_pipeline.predict(X_test)
prob_score = best_svc_pipeline.decision_function(X_test)
print('Best pipeline:', str(best_svc_pipeline.get_params()['steps']))
plot_metrics(y_test, prediction, prob_score, 'Test confusion matrix of
best pipeline\n')
```




```
Best pipeline: [('MinMaxScaler', MinMaxScaler()), ('PCA', PCA(n_components=22, r
andom_state=0)), ('model', SVC(probability=True, random_state=0))]
```




```
Sensitivity: 0.7410358565737052
Specificity: 0.8936484490398818
Accuracy score: 0.8523706896551724
```

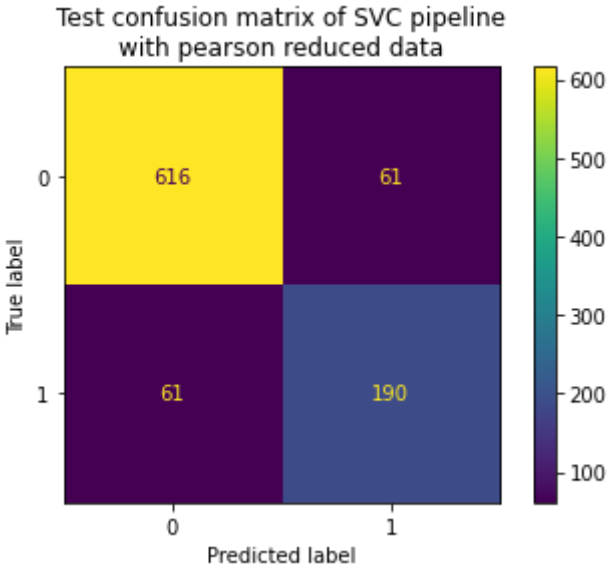


Applying preprocessing: Pearson correlation feature reduction

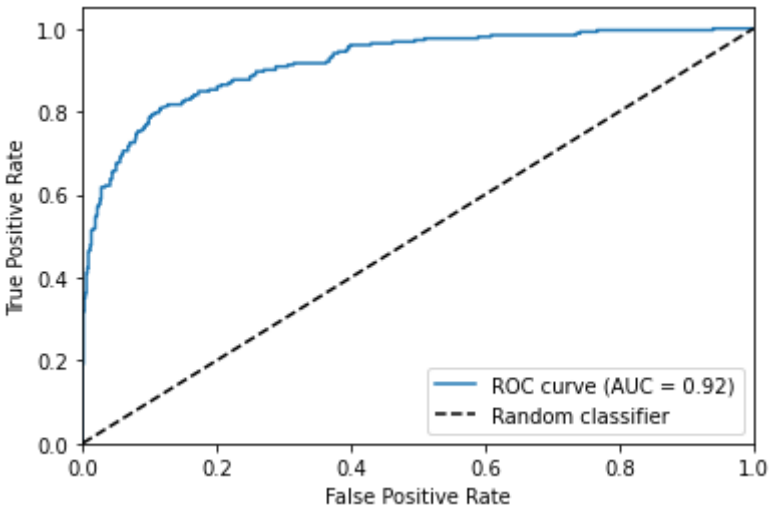
In [157…

```
svc = SVC(random_state=0, probability=True)
pipeline_svc = Pipeline([('pearson', PearsonFeatureReduction()),
('MinMaxScaler', MinMaxScaler((0,1))), ('model', svc)])
parameters = {'pearson__threshold': .01*np.arange(60, 100, 5)}
grid_search = GridSearchCV(pipeline_svc, param_grid = parameters,
verbose=1, cv=3)
grid_search.fit(X, y)
print(grid_search.best_params_)
pipeline_svc.set_params(**grid_search.best_params_)
pipeline_svc.fit(X_train, y_train)

prediction = pipeline_svc.predict(X_test)
prob_score = pipeline_svc.predict_proba(X_test)[:,1]
plot_metrics(y_test, prediction, prob_score, 'Test confusion matrix of
SVC pipeline\nwith pearson reduced data')
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
{'pearson__threshold': 0.6}
```




```
Sensitivity: 0.7569721115537849
Specificity: 0.9098966026587888
Accuracy score: 0.8685344827586207
```

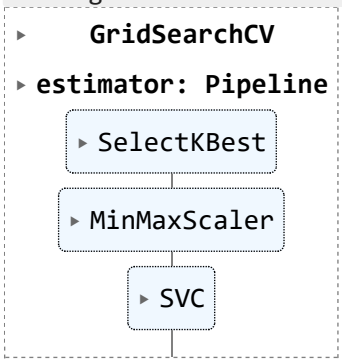


Applying preprocessing: SelectKBest

In [30]:
```
#gridsearch will be performed for the best pipeline above to further
improve the accuracy
svc = SVC(random_state=0, probability=True)
pipeline_svc = Pipeline([('KBest', SelectKBest(mutual_info_classif)),
('MinMaxScaler', MinMaxScaler((0,1))), ('model', svc)])
params = {'KBest__k': np.arange(1, 151, dtype=int)}
grid_search_svc = GridSearchCV(pipeline_svc, param_grid=params, cv=3,
verbose=2, n_jobs=-1)
grid_search_svc.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 150 candidates, totalling 450 fits
```

Out[30]:



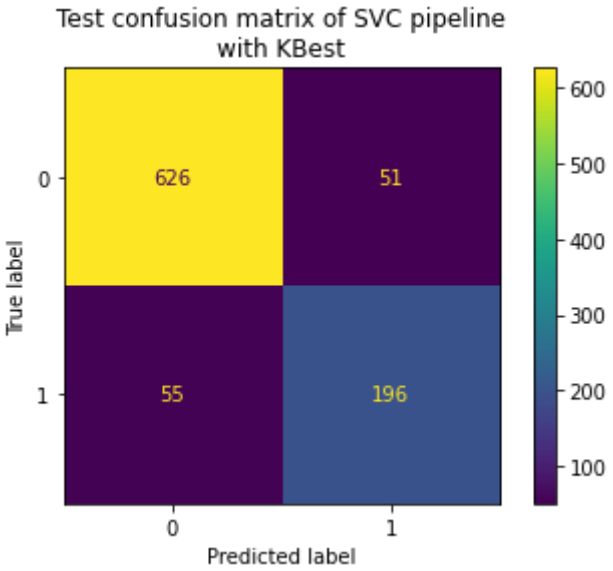
In [31]:
```
grid_search_svc.best_params_
```

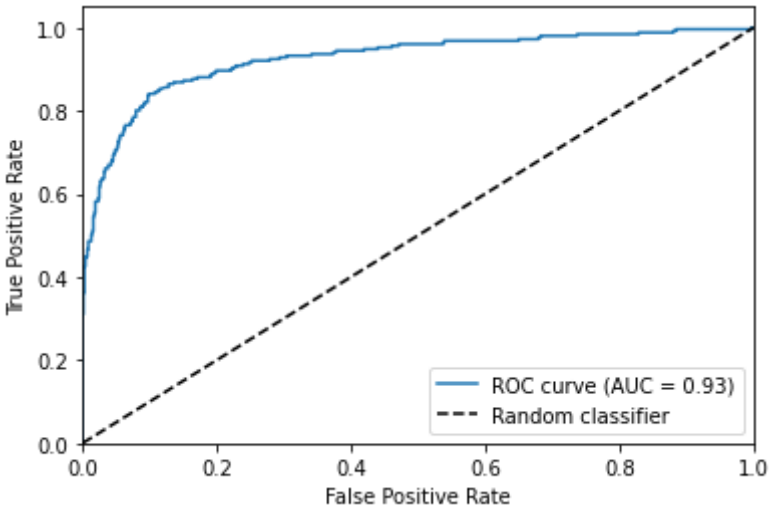
Out[31]:
```
{'KBest__k': 22}
```

In [32]:
```
k = grid_search_svc.best_params_['KBest__k']
svc = SVC(random_state=0, probability=True)
pipeline_svc = Pipeline([('KBest', SelectKBest(mutual_info_classif,
k=k)), ('MinMaxScaler', MinMaxScaler((0,1))),
                                ('model', svc)])
```

In [33]:
```
pipeline_svc.fit(X_train, y_train)
prediction = pipeline_svc.predict(X_test)
prob_score = pipeline_svc.predict_proba(X_test)[:,1]
print('Best pipeline:', str(pipeline_svc.get_params()['steps']))
plot_metrics(y_test, prediction, prob_score, 'Test confusion matrix of
SVC pipeline\nwith KBest')
```

```
Best pipeline: [('KBest', SelectKBest(k=22,
            score_func=<function mutual_info_classif at 0x000001D1ECB2B790>)),
('MinMaxScaler', MinMaxScaler()), ('model', SVC(probability=True, random_state=
0))]
```






```
Sensitivity: 0.7808764940239044
Specificity: 0.9246676514032496
Accuracy score: 0.8857758620689655
```

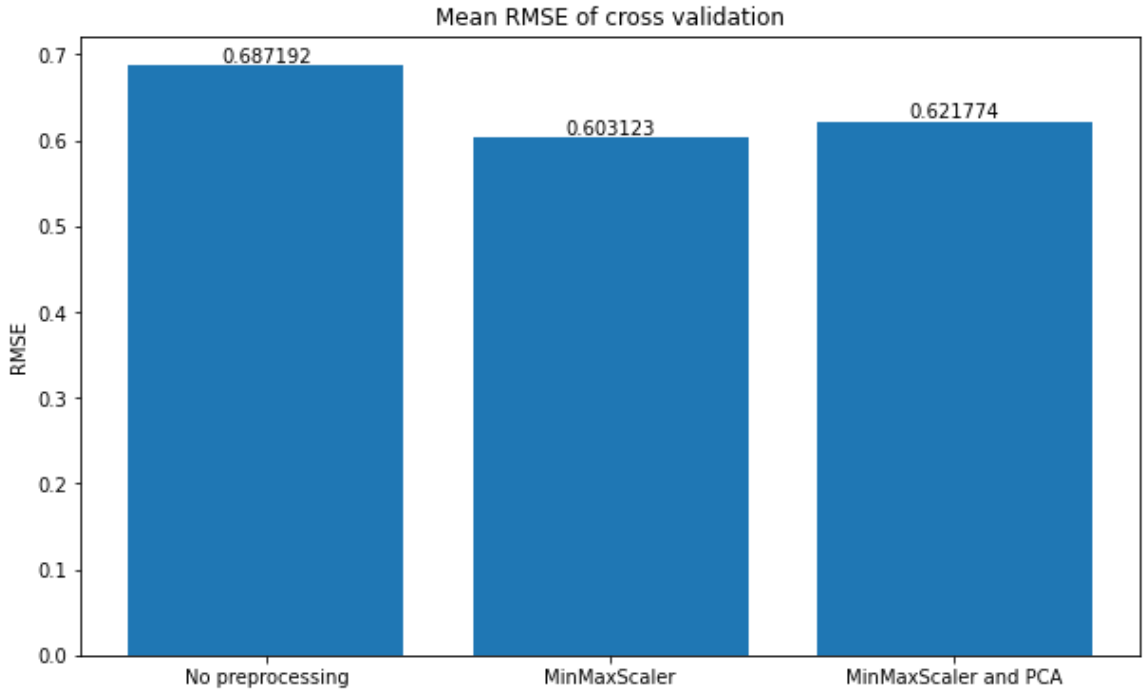
Multi-layer perceptron

Applying preprocessing: MinMax and PCA

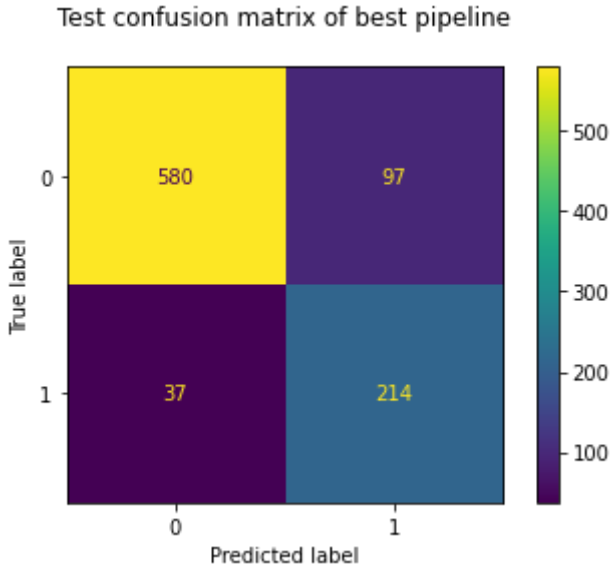
In [34]:

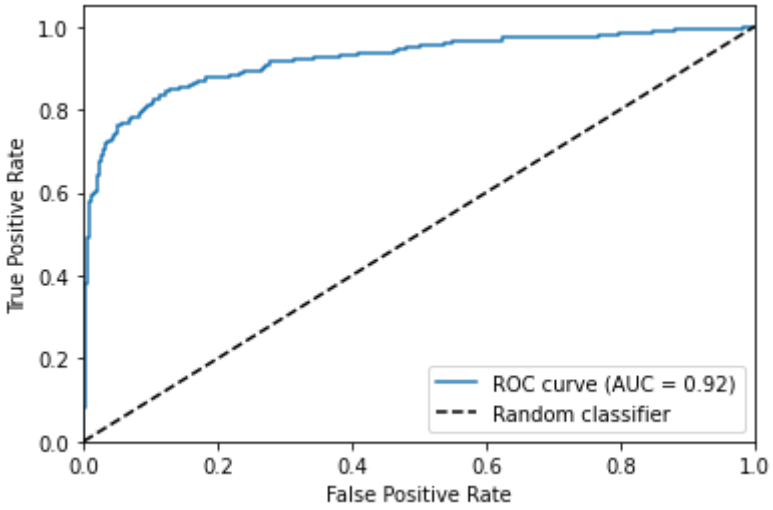
```
mean_cv_rmse = []
pipelines = []
#no preprocessing
mlp = MLPClassifier(random_state=0, max_iter = 3000)
pipelines.append(Pipeline([('model', mlp)]))
score = get_mean_cv_rmse(mlp, X_train, y_train)
mean_cv_rmse.append(score)
#minmax
pipeline_mlp = Pipeline([('MinMaxScaler', MinMaxScaler((0,1))),
('model', mlp)])
pipelines.append(pipeline_mlp)
score = get_mean_cv_rmse(pipeline_mlp, X_train, y_train)
mean_cv_rmse.append(score)
#PCA and minmax
pipeline_mlp = Pipeline([('MinMaxScaler', MinMaxScaler((0,1))),
('PCA', PCA(random_state=0)),('model', mlp)])
params={'PCA__n_components': np.arange(1, 151, dtype=int)}
grid_search_pca = GridSearchCV(pipeline_mlp, param_grid=params, cv=3)
grid_search_pca.fit(X_train, y_train)
pipeline_mlp.set_params(**grid_search_pca.best_params_)
pipelines.append(pipeline_mlp)

score = get_mean_cv_rmse(pipeline_mlp, X_train, y_train)
mean_cv_rmse.append(score)

compare_RMSEs(mean_cv_rmse, labels=["No preprocessing",
"MinMaxScaler", "MinMaxScaler and PCA"])

best_mlp_pipeline = pipelines[np.argmin(mean_cv_rmse)]
best_mlp_pipeline.fit(X_train, y_train)
prediction = best_mlp_pipeline.predict(X_test)

prob_score = best_mlp_pipeline.predict_proba(X_test)[:,1]
print('Best pipeline:', str(best_mlp_pipeline.get_params()['steps']))
plot_metrics(y_test, prediction, prob_score, 'Test confusion matrix of
best pipeline\n')
```

```
C:\Users\Anass Hamdi\AppData\Local\Programs\Python\Python39\lib\site-packages\sk
learn\neural_network\_multilayer_perceptron.py:702: ConvergenceWarning: Stochast
ic Optimizer: Maximum iterations (3000) reached and the optimization hasn't conv
erged yet.
  warnings.warn(
C:\Users\Anass Hamdi\AppData\Local\Programs\Python\Python39\lib\site-packages\sk
learn\neural_network\_multilayer_perceptron.py:702: ConvergenceWarning: Stochast
ic Optimizer: Maximum iterations (3000) reached and the optimization hasn't conv
erged yet.
  warnings.warn(
```




```
Best pipeline: [('MinMaxScaler', MinMaxScaler()), ('model', MLPClassifier(max_it
er=3000, random_state=0))]
```

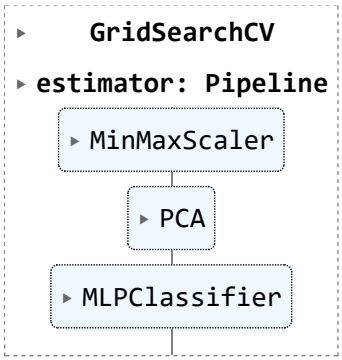





```
Sensitivity: 0.852589641434263
Specificity: 0.8567208271787297
Accuracy score: 0.8556034482758621
```

In [35]:
```
grid_search_pca
```

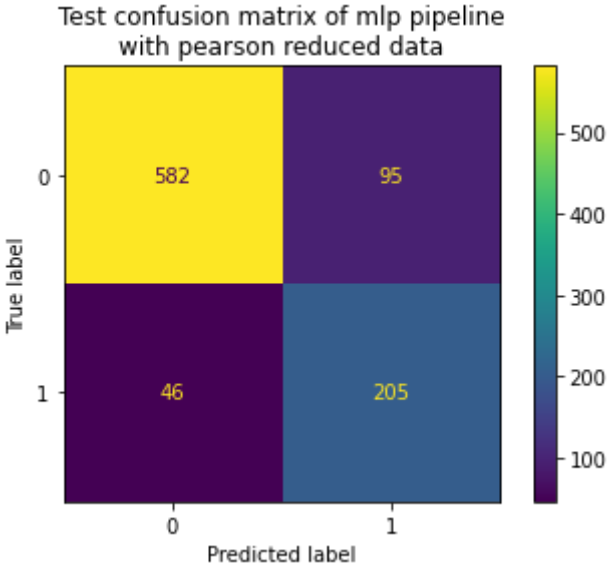
Out[35]:



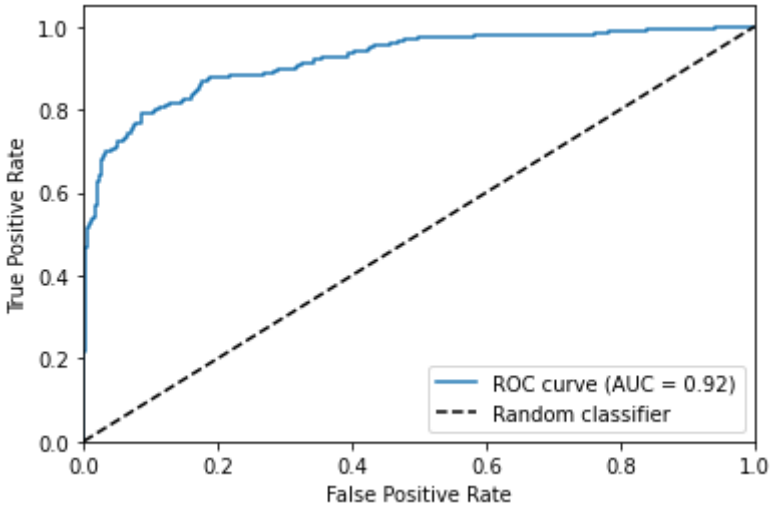
Applying preprocessing: Pearson correlation feature reduction

In [15]:
```
mlp = MLPClassifier(random_state=0, max_iter = 1000)
pipeline_mlp = Pipeline([('pearson', PearsonFeatureReduction()),
('MinMaxScaler', MinMaxScaler((0,1))), ('model', mlp)])
parameters = {'pearson__threshold': .01*np.arange(60, 100, 5),
              'model__hidden_layer_sizes': np.linspace(10, 150, 15,
dtype=int)
             }
grid_search = GridSearchCV(pipeline_mlp, param_grid = parameters,
verbose=1, cv=3)
grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
pipeline_mlp.set_params(**grid_search.best_params_)
pipeline_mlp.fit(X_train, y_train)

prediction = pipeline_mlp.predict(X_test)
prob_score = pipeline_mlp.predict_proba(X_test)[:,1]
plot_metrics(y_test, prediction, prob_score, 'Test confusion matrix of
mlp pipeline\nwith pearson reduced data')
```

```
Fitting 3 folds for each of 120 candidates, totalling 360 fits
{'model__hidden_layer_sizes': 30, 'pearson__threshold': 0.6}
```




```
Sensitivity: 0.8167330677290837
Specificity: 0.8596750369276218
Accuracy score: 0.8480603448275862
```



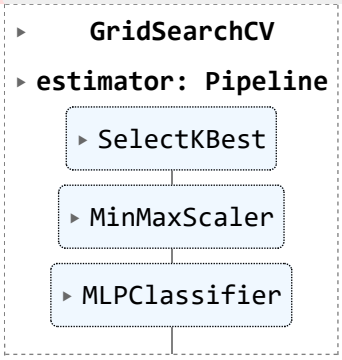
Applying preprocessing: SelectionKBest

In [48]:
```
parameters = {'KBest__k': np.linspace(10, 150, 15, dtype=int),
              'model__hidden_layer_sizes': np.linspace(10, 150, 15,
dtype=int)
             }
mlp = MLPClassifier(random_state=0, max_iter=1000)
pipeline = Pipeline([('KBest', SelectKBest()), ('minmax',
MinMaxScaler()), ('model', mlp)])
grid_search = GridSearchCV(pipeline, param_grid = parameters,
cv=3,verbose=1)
grid_search.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 225 candidates, totalling 675 fits
```

```
C:\Users\Anass Hamdi\AppData\Local\Programs\Python\Python39\lib\site-packages\sk
learn\neural_network\_multilayer_perceptron.py:702: ConvergenceWarning:

Stochastic Optimizer: Maximum iterations (1000) reached and the optimization has
n't converged yet.
```

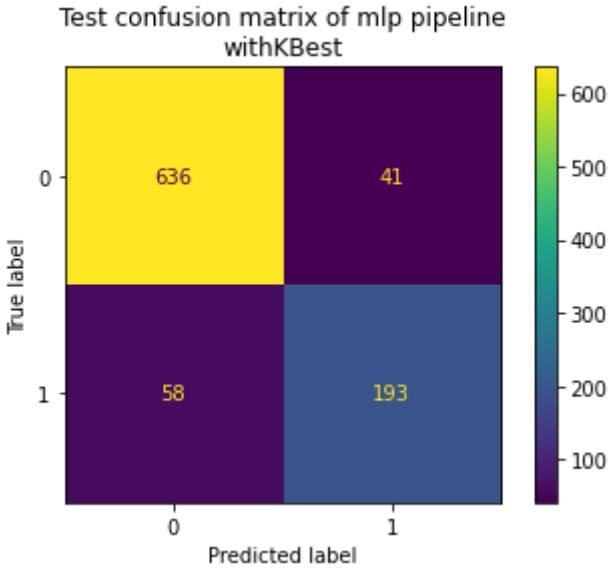
Out[48]:



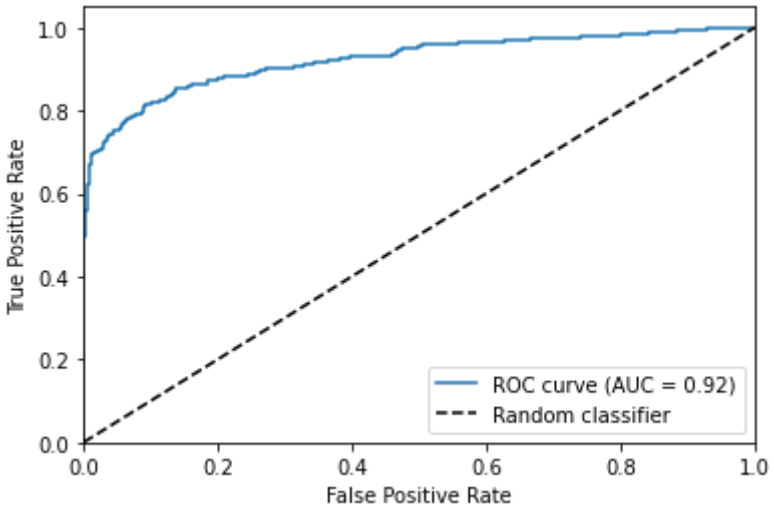
In [49]:
```
grid_search.best_params_
#{'KBest__k': 110, 'model__hidden_layer_sizes': 60}
```

Out[49]:
```
{'KBest__k': 110, 'model__hidden_layer_sizes': 60}
```

In [50]:
```
optimal_params = {'KBest__k': 110, 'model__hidden_layer_sizes': 60}
pipeline_mlp = Pipeline([('KBest', SelectKBest()), ('minmax',
MinMaxScaler()),
                         ('model', MLPClassifier(random_state=0,
max_iter=1000))])
pipeline_mlp.set_params(**optimal_params)
pipeline_mlp.fit(X_train, y_train)
prediction = pipeline_mlp.predict(X_test)
prob_score = pipeline_mlp.predict_proba(X_test)[:,1]
plot_metrics(y_test, prediction, prob_score, 'Test confusion matrix of
mlp pipeline\nwithKBest')
```




```
Sensitivity: 0.7689243027888446
Specificity: 0.9394387001477105
Accuracy score: 0.8933189655172413
```



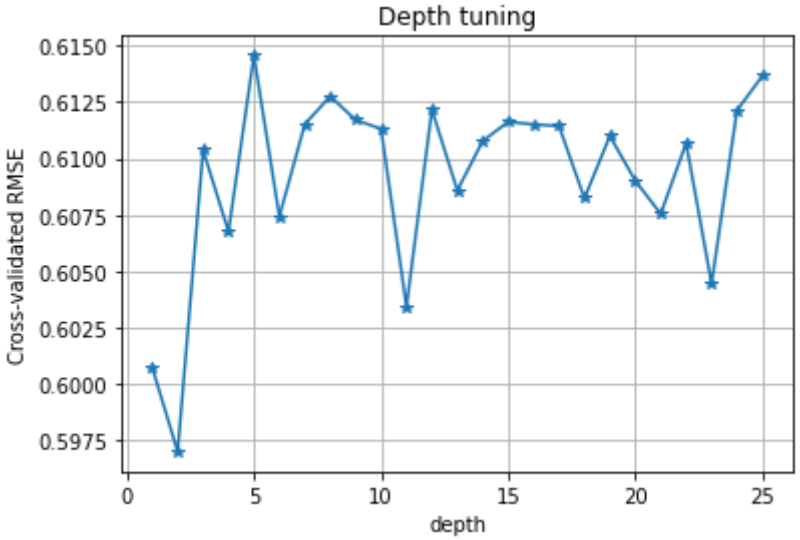
XGBoost classifier

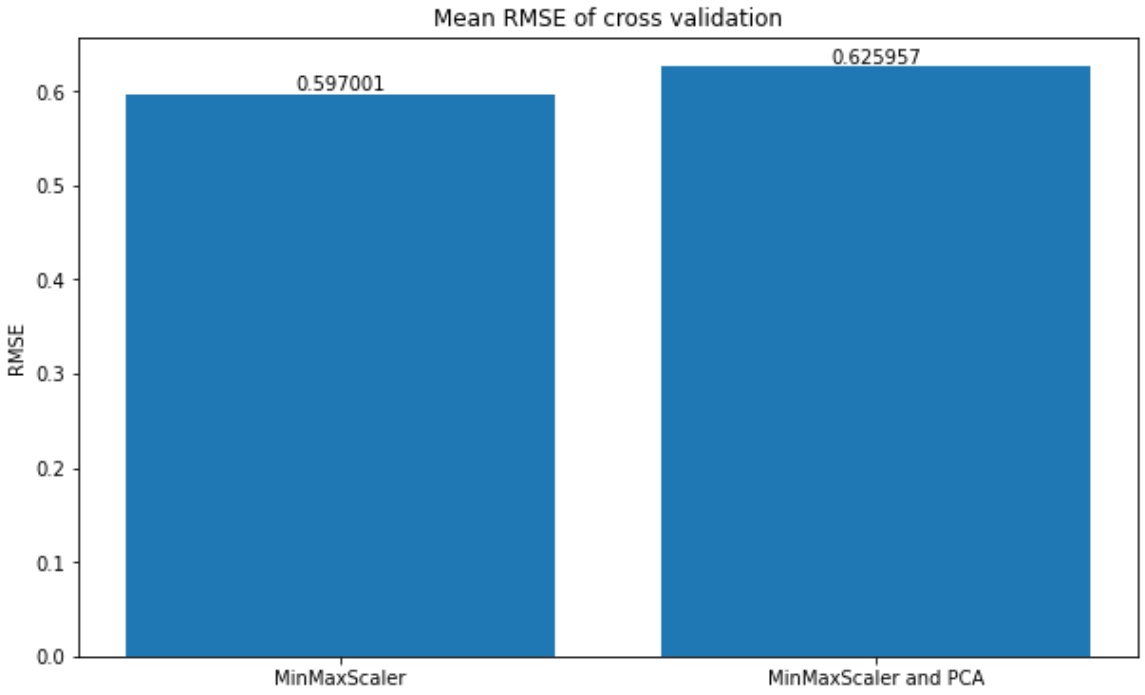
Applying preprocessing: MinMax and PCA

In [39]:

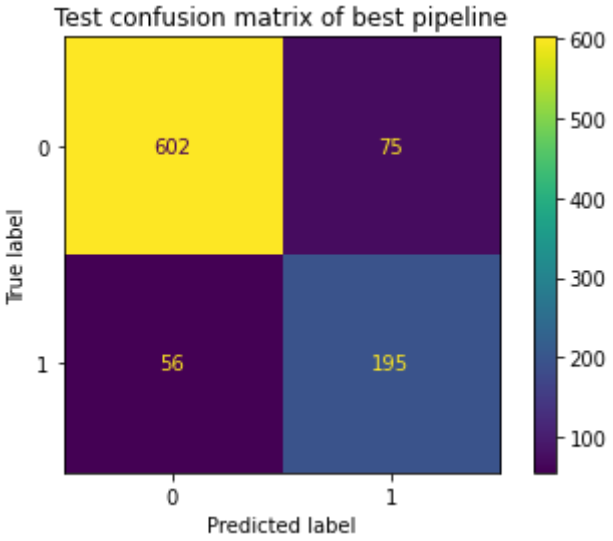
```
depth = list(range(1, 26, 1))
pipelines=[]
rmse=[]
#depth tuning with minmax
for parameter in depth:
    model = XGBClassifier(random_state=0, n_estimators=300,
learning_rate=0.05, n_jobs=4, max_depth = parameter)
    pipeline = Pipeline([('preprocessing', MinMaxScaler((0,1))),
('model', model)])
    rmse.append(get_mean_cv_rmse(pipeline, X_train, y_train))
    pipelines.append(pipeline)

#plot depth tuning
plt.figure()
plt.title("Depth tuning")
plt.plot(depth, rmse,'-*')
plt.xlabel("depth")
plt.ylabel("Cross-validated RMSE")
plt.grid()
plt.show()

#Minmax with optimal depth
pipeline_xgb = pipelines[np.argmin(rmse)]
optimal_depth = depth[np.argmin(rmse)]
score = min(rmse)

pipelines=[]
rmse=[]
pipelines.append(pipeline_xgb)
rmse.append(score)

#minmax and PCA with optimal depth
model = XGBClassifier(random_state=0, n_estimators=300,
learning_rate=0.05, n_jobs=4, max_depth = optimal_depth)
pipeline_xgb = Pipeline([('MinMaxScaler', MinMaxScaler((0,1))),
('PCA', PCA(random_state=0)),('model', model)])
params={'PCA__n_components': np.arange(1, 151, dtype=int)}
grid_search_pca = GridSearchCV(pipeline_xgb, param_grid=params, cv=3)
grid_search_pca.fit(X_train, y_train)
pipeline_xgb.set_params(**grid_search_pca.best_params_)
score = get_mean_cv_rmse(pipeline_xgb, X_train, y_train)
```

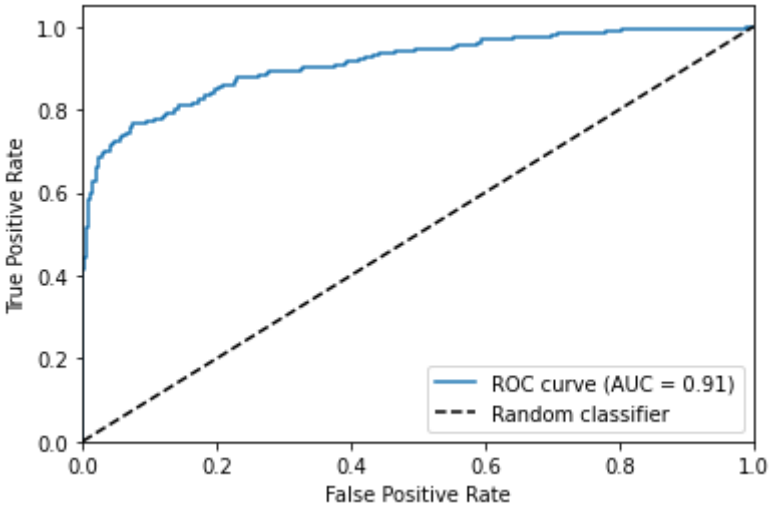
```
pipelines.append(pipeline_xgb)
rmse.append(score)
compare_RMSEs(rmse, labels=['MinMaxScaler', 'MinMaxScaler and PCA'])
#plot best pipeline
pipeline_xgb = pipelines[np.argmin(rmse)]
pipeline_xgb.fit(X_train, y_train)
prediction = pipeline_xgb.predict(X_test)
prob_score = pipeline_xgb.predict_proba(X_test)[:,1]
print('Best pipeline:', str(pipeline_xgb.get_params()['steps']))
plot_metrics(y_test, prediction, prob_score, f'Test confusion matrix
of best pipeline')
```





```
Best pipeline: [('preprocessing', MinMaxScaler()), ('model', XGBClassifier(base_
score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.05, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=2, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=300, n_jobs=4, num_parallel_tree=None,
              predictor=None, random_state=0, ...))]
```



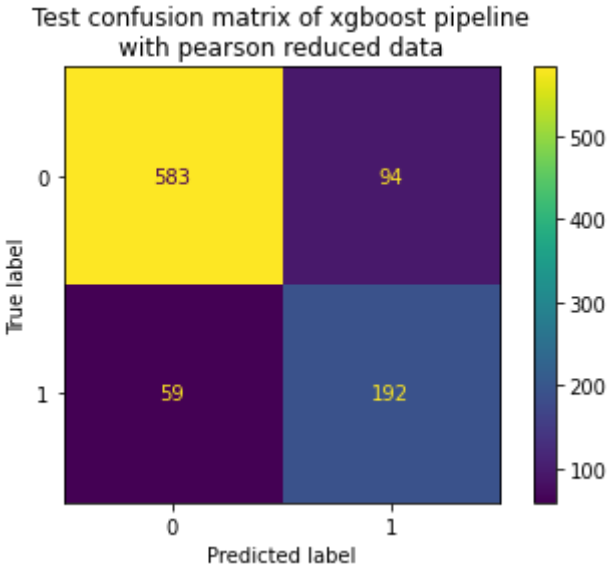


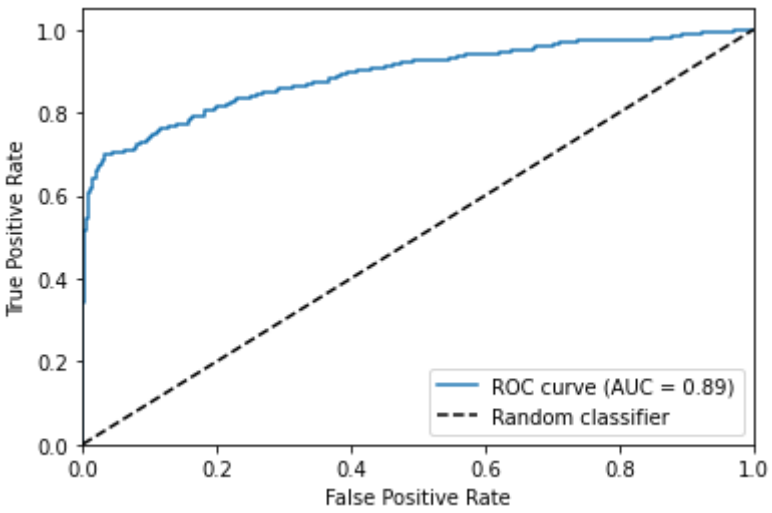

```
Sensitivity: 0.7768924302788844
Specificity: 0.8892171344165436
Accuracy score: 0.8588362068965517
```

Applying preprocessing: Pearson correlation feature reduction

In [40]:
```
model = XGBClassifier(random_state=0, n_estimators=300,
learning_rate=0.05, n_jobs=4, max_depth = 18)
pipeline_xgb = Pipeline([('pearson', PearsonFeatureReduction()),
('minmax', MinMaxScaler((0,1))), ('model', model)])
parameters = {'pearson__threshold': .01*np.arange(60, 100, 5)}
grid_search = GridSearchCV(pipeline_xgb, param_grid = parameters,
verbose=1, cv=3)
grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
pipeline_xgb.set_params(**grid_search.best_params_)
pipeline_xgb.fit(X_train, y_train)

prediction = pipeline_xgb.predict(X_test)
prob_score = pipeline_xgb.predict_proba(X_test)[:,1]
plot_metrics(y_test, prediction, prob_score, 'Test confusion matrix of
xgboost pipeline\nwith pearson reduced data')
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
{'pearson__threshold': 0.6}
```





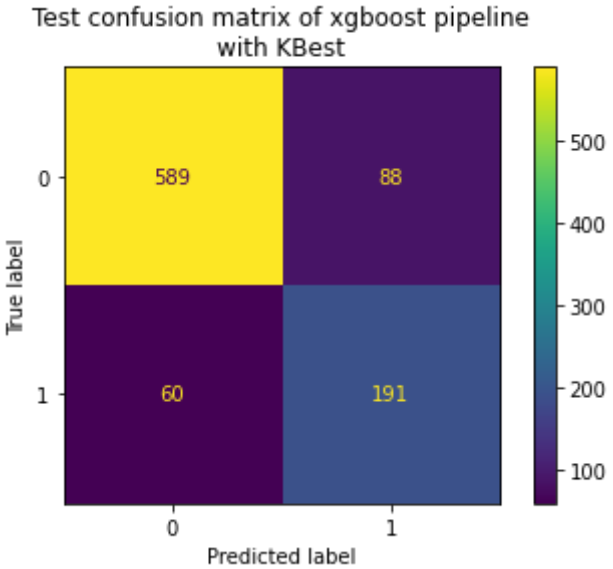
```
Sensitivity: 0.7649402390438247
Specificity: 0.8611521418020679
Accuracy score: 0.8351293103448276
```

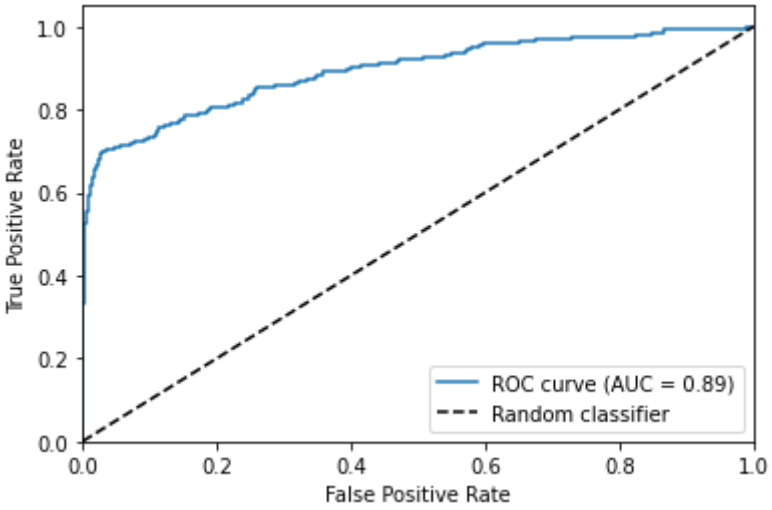
Applying preprocessing: SelectKBest

In [41]:

```
model = XGBClassifier(random_state=0, n_estimators=300,
learning_rate=0.05, n_jobs=4, max_depth = 18)
pipeline_xgb = Pipeline([('KBest', SelectKBest(mutual_info_classif)),
('minmax', MinMaxScaler()), ('model', model)])

parameters = {'KBest__k': np.arange(1, 151, 1, dtype=int)}
grid_search = GridSearchCV(pipeline_xgb, param_grid = parameters,
verbose=1, cv=3)
grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
pipeline_xgb.set_params(**grid_search.best_params_)

pipeline_xgb.fit(X_train, y_train)

prediction = pipeline_xgb.predict(X_test)
prob_score = pipeline_xgb.predict_proba(X_test)[:,1]
plot_metrics(y_test, prediction, prob_score, 'Test confusion matrix of
xgboost pipeline\nwith KBest')
```

```
Fitting 3 folds for each of 150 candidates, totalling 450 fits
{'KBest__k': 140}
```



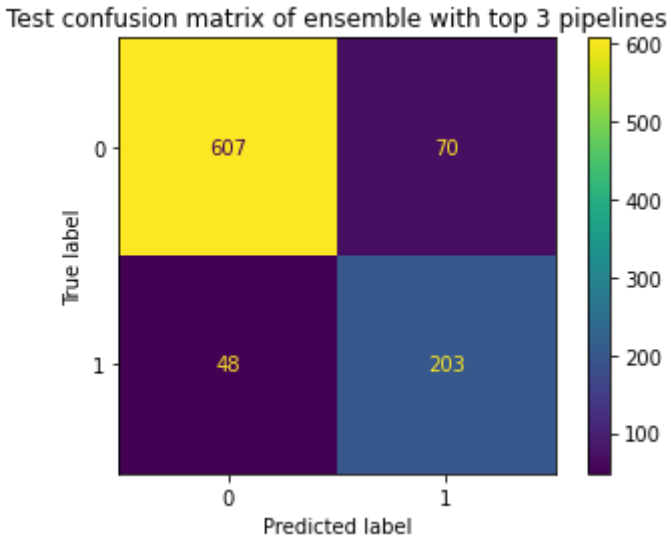



```
Sensitivity: 0.7609561752988048
Specificity: 0.8700147710487445
Accuracy score: 0.8405172413793104
```

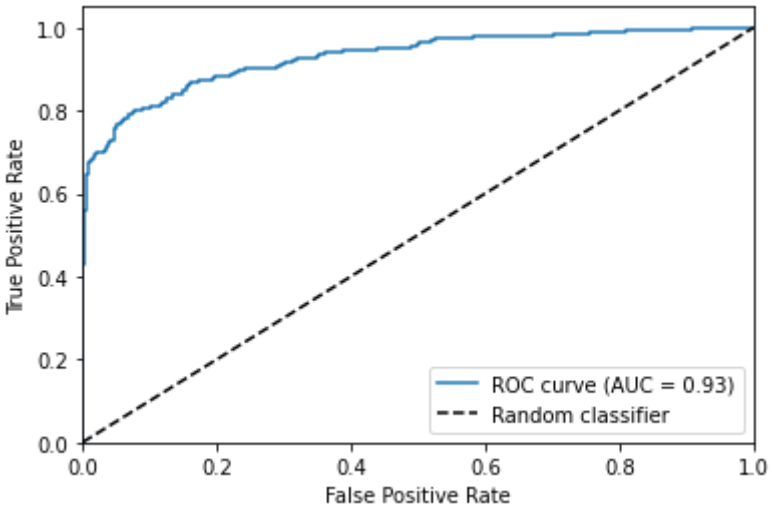
Creating ensemble model

In [17]:

```
######## INITIAL ENSEMBLE MODEL ########
#top 3: MLP, XGBoost classifier, c-support vector classifier
#redefine top 3 pipelines
#mlp pipeline
#best_mlp_pipeline = Pipeline([('minmax', MinMaxScaler()), ('model',
MLPClassifier(random_state=0, max_iter=3000))])
#xgboost pipeline
#model = XGBClassifier(random_state=0, n_estimators=300,
learning_rate=0.05, n_jobs=-1, max_depth = 18)
#best_xgb_pipeline = Pipeline([('minmax', MinMaxScaler((0,1))),
('model', model)])
#SVC pipeline
#svc = SVC(random_state=0, probability=True)
#k=22
#best_svc_pipeline = Pipeline([('KBest',
SelectKBest(mutual_info_classif, k=k)), ('MinMaxScaler',
MinMaxScaler((0,1))),
#                              ('model', svc)])
#ensemble is created
#estimators=[('mlp', best_mlp_pipeline), ('xgboost',
best_xgb_pipeline), ('SVC', best_svc_pipeline)]
#ensemble = VotingClassifier(estimators, voting='soft')

#ensemble.fit(X_train, y_train)
#prediction = ensemble.predict(X_test)
#prob_score = ensemble.predict_proba(X_test)[:,1]
#plot_metrics(y_test, prediction, prob_score, title= 'Test confusion
matrix of ensemble with top 3 pipelines')
```



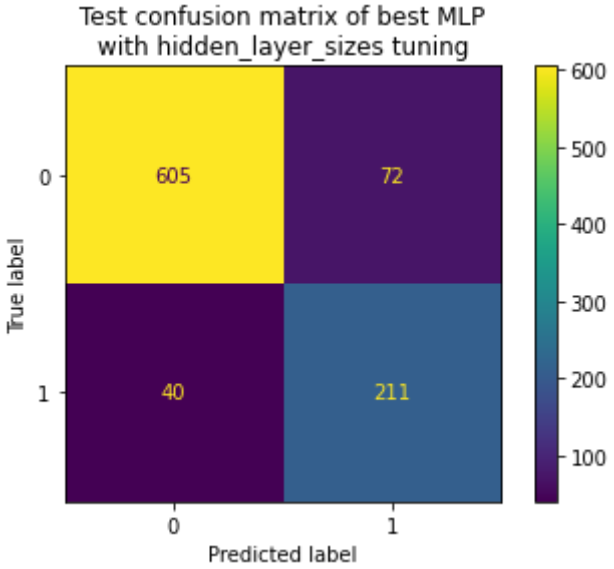
```
Sensitivity: 0.8087649402390438
Specificity: 0.896602658788774
Accuracy score: 0.8728448275862069
```



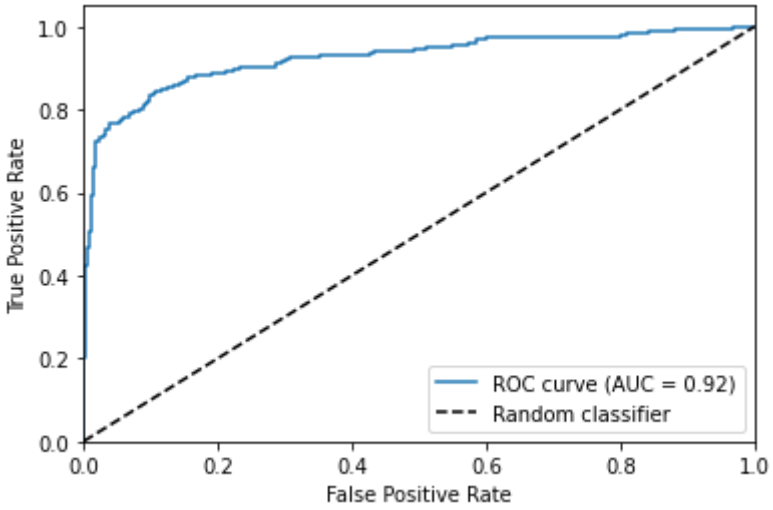
In [80]:

```
######## hidden_layer_sizes tuning #######
best_mlp_pipeline = Pipeline([('minmax', MinMaxScaler()),
                              ('model', MLPClassifier(random_state=0,
max_iter=1000))])
parameters = {'model__hidden_layer_sizes': np.linspace(10, 150, 15,
dtype=int)}
grid_search = GridSearchCV(best_mlp_pipeline, param_grid = parameters,
cv=3,verbose=1)
grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
best_mlp_pipeline.set_params(**grid_search.best_params_)
best_mlp_pipeline.fit(X_train, y_train)
prediction = best_mlp_pipeline.predict(X_test)
prob_score = best_mlp_pipeline.predict_proba(X_test)[:,1]
plot_metrics(y_test, prediction, prob_score, title= 'Test confusion
matrix of best MLP\nwith hidden_layer_sizes tuning')
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
{'model__hidden_layer_sizes': 70}
```

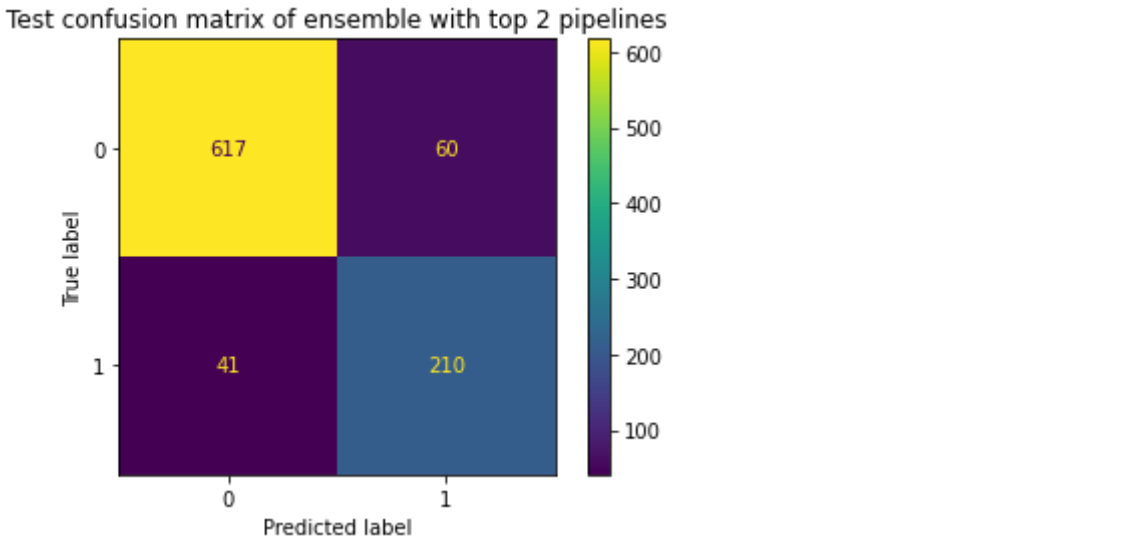



```
Sensitivity: 0.8406374501992032
Specificity: 0.8936484490398818
Accuracy score: 0.8793103448275862
```

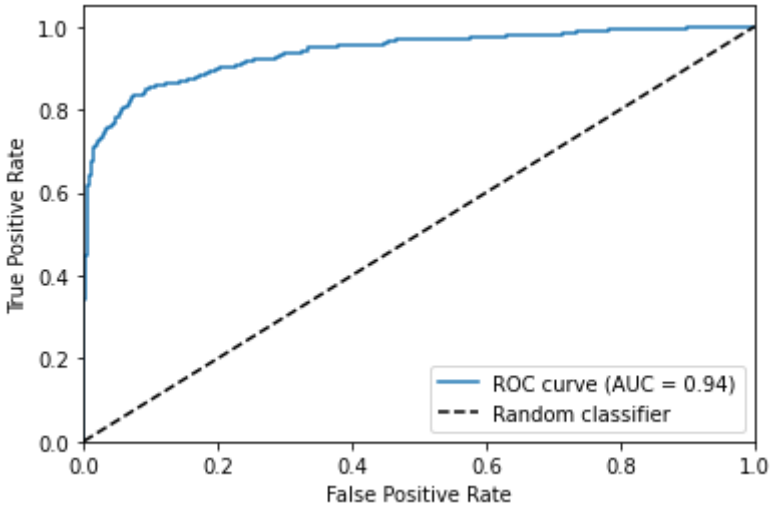


In [19]:

```
######## FINAL ENSEMBLE MODEL ########
#top 2: MLP, c-support vector classifier
#redefine top 2 pipelines
#mlp pipeline
best_mlp_pipeline = Pipeline([('minmax', MinMaxScaler()),
                              ('model', MLPClassifier(random_state=0,
hidden_layer_sizes = (70,), max_iter=1000))
                             ])
#SVC pipeline
svc = SVC(random_state=0, probability=True)
k=22
best_svc_pipeline = Pipeline([('KBest',
SelectKBest(mutual_info_classif, k=k)), ('MinMaxScaler',
MinMaxScaler((0,1))),
                              ('model', svc)])
#ensemble is created
estimators=[('mlp', best_mlp_pipeline), ('SVC', best_svc_pipeline)]
ensemble = VotingClassifier(estimators, weights=[1, 1.5],
voting='soft')

ensemble.fit(X_train, y_train)
prediction = ensemble.predict(X_test)
prob_score = ensemble.predict_proba(X_test)[:,1]
plot_metrics(y_test, prediction, prob_score, title= 'Test confusion
matrix of ensemble with top 2 pipelines')
```




```
Sensitivity: 0.8366533864541833
Specificity: 0.9113737075332349
Accuracy score: 0.8911637931034483
```



In [20]:
```
#https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/
#Save model
#model_micros = ensemble
#filename = "model_micros.joblib"
#joblib.dump(model_micros, filename)
```

Analyzing train-test split and results with deepchecks

In [11]:
```
#SOURCE: https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/
#load model
filename = "model_micros.joblib"
model_micros = joblib.load(filename)
```

In [21]:
```
#SOURCE: https://docs.deepchecks.com/stable/tabular/auto_tutorials/quickstarts/plot_quick_train_test_validation.html
train_ds = Dataset(X_train.reset_index(drop=True),
label=y_train.reset_index(drop=True), cat_features=[])
test_ds =  Dataset(X_test.reset_index(drop=True),
label=y_test.reset_index(drop=True), cat_features=[])
```

In [22]:
```
#SOURCE: https://docs.deepchecks.com/stable/tabular/auto_tutorials
/quickstarts/plot_quick_train_test_validation.html
#NOTE: the results don't show up on a widget as it supposed to in
JupyterLab, it will work in JupyterNotebook and GoogleColab
#train_test validation
validation_suite = train_test_validation()
suite_result = validation_suite.run(train_ds, test_ds, model_micros)
# Note: the result can be saved as html using
suite_result.save_as_html()
# or exported to json using suite_result.to_json()
suite_result.show_in_iframe()
```

```
deepchecks - WARNING - Could not find built-in feature importance on the model,
using permutation feature importance calculation instead
deepchecks - WARNING - Features importance was not calculated:
Skipping permutation importance calculation: calculation was projected to finish
in 1211 seconds, but timeout was configured to 120 seconds
```

Full Screen

▼ Train Test Validation Suite

Train Test Validation Suite

The suite is composed of various checks such as: Feature Drift, Train Test Samples Mix, Feature Label Correlation Change, etc...

Each check may contain conditions (which will result in pass ✓ / fail ✗ / warning ! / error ?!) as well as other outputs such as plots or tables.

Suites, checks and conditions can all be modified. Read more about custom suites.

▸ Didn't Pass

▸ Passed

▸ Other

▸ Didn't Run

In [23]:
```
#SOURCE: https://docs.deepchecks.com/stable/api/generated
/deepchecks.tabular.suites.model_evaluation.html
#model evauation
suite = model_evaluation()
result = suite.run(train_ds, test_ds, model_micros)

result.show_in_iframe()
```

```
deepchecks - WARNING - Could not find built-in feature importance on the model,
using permutation feature importance calculation instead
deepchecks - WARNING - Features importance was not calculated:
Skipping permutation importance calculation: calculation was projected to finish
in 1137 seconds, but timeout was configured to 120 seconds
```

Full Screen

▼ Model Evaluation Suite

Model Evaluation Suite

The suite is composed of various checks such as: Confusion Matrix Report, Simple Model Comparison, Regression Error Distribution, etc...

Each check may contain conditions (which will result in pass ✓ / fail ✗ / warning ! / error ?!) as well as other outputs such as plots or tables.

Suites, checks and conditions can all be modified. Read more about custom suites.

▸ Didn't Pass

▸ Passed

▸ Other

▸ Didn't Run

Part 2: Cancer classification based on classification of multiple micros per subject

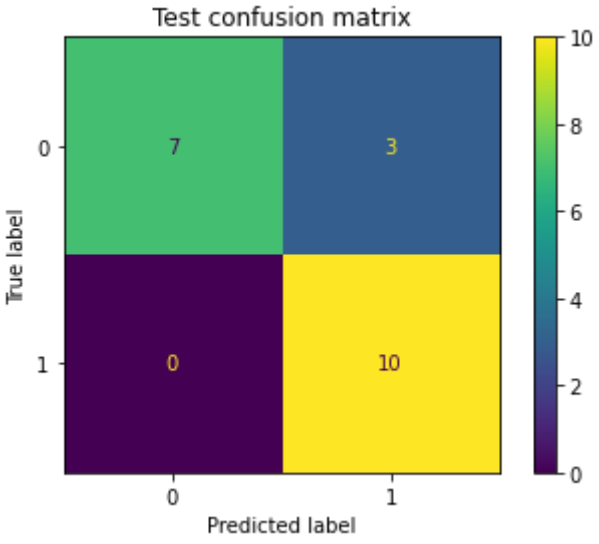
Classifying based on thresholding

In [43]:
```
#Load model
filename = "model_micros.joblib"
model_micros = joblib.load(filename)
```

In [17]:

```
prediction = model_micros.predict(X_test)
prediction = pd.Series(prediction, index=y_test.index)

l_threshold = 0.01*np.arange(5, 60)
score = []
y_test_cancer = y_test.groupby('Patient ID').max()      #reduce the
rows to one per patient id by using max as
                                                        #all labels
are the same per patient, thus we can get one label that
                                                        #represents
whether the patient has breast cancer
for threshold in l_threshold:
    prediction_cancer = prediction.groupby('Patient
ID').mean().apply(lambda x: 1 if x>=threshold else 0)
    acc = accuracy_score(y_test_cancer,
                         prediction_cancer)
    score.append(acc)
th_optimal = l_threshold[np.argmax(score)]
prediction_cancer = prediction.groupby('Patient
ID').mean().apply(lambda x: 1 if x>=th_optimal else 0)
y_test_cancer = y_test.groupby('Patient ID').max()

display(Markdown(f'optimal threshold: {round(th_optimal, 2)}'))
plot_metrics(y_test_cancer, prediction_cancer, title='Test confusion
matrix')
```

optimal threshold: 0.14




```
Sensitivity: 1.0
Specificity: 0.7
Accuracy score: 0.85
```