Data Space Report (Official) - Decision Trees-v1.0.0

June 11, 2020

1 Data Space Report

1.1 Pittsburgh Bridges Data Set

`Andy Warhol Bridge - Pittsburgh.`

Report created by Student Francesco Maria Chiarlo s253666, for A.A 2019/2020.

Abstract:The aim of this report is to evaluate the effectiveness of distinct, different statistical learning approaches, in particular focusing on their characteristics as well as on their advantages and backwards when applied onto a relatively small dataset as the one employed within this report, that is Pittsburgh Bridgesdataset.

Key words:Statistical Learning, Machine Learning, Bridge Design.

1.1.1 Imports Section

[11]:
```
from utils.all_imports import *;
%matplotlib inline
```

[12]:
```
# Set seed for notebook repeatability
np.random.seed(0)

# READ INPUT DATASET
# =========================================================================== #
dataset_path, dataset_name, column_names, TARGET_COL = get_dataset_location()
estimators_list, estimators_names = get_estimators()

dataset, feature_vs_values = load_brdiges_dataset(dataset_path, dataset_name)
```

[13]:
```
columns_2_avoid = ['ERECTED', 'LENGTH', 'LOCATION']
```

[14]:
```
# Make distinction between Target Variable and Predictors
# --------------------------------------------------------------------------- #
rescaledX, y, columns = prepare_data_for_train(dataset, target_col=TARGET_COL)
```

```
Summary about Target Variable {target_col}
--------------------------------------------------
2    57
```

```
1    13
Name: T-OR-D, dtype: int64
shape features matrix X, after normalizing:  (70, 11)
```

1.2 Pricipal Component Analysis

[15]:
```
show_table_pc_analysis(X=rescaledX)
```

```
Cumulative varation explained(percentage) up to given number of pcs:
```

[15]:
```
   # PCS  Cumulative Varation Explained (percentage)
0      2                                    47.738342
1      5                                    75.856460
2      6                                    82.615768
3      7                                    88.413903
4      8                                    92.661938
5      9                                    95.976841
6     10                                    98.432807
```

Major Pros & Cons of PCA

1.3 Learning Models

[16]:
```
# Parameters to be tested for Cross-Validation Approach
# -----------------------------------------------------

# Array used for storing graphs
plots_names = list(map(lambda xi: f"{xi}_learning_curve.png", estimators_names))
pca_kernels_list = ['linear', 'poly', 'rbf', 'cosine', 'sigmoid']
cv_list = list(range(10, 1, -1))

param_grids = []
parmas_logreg = {
    'penalty': ('l1', 'l2', 'elastic', None),
    'solver': ('newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'),
    'fit_intercept': (True, False),
    'tol': (1e-4, 1e-3, 1e-2),
    'class_weight': (None, 'balanced'),
    'C': (10.0, 1.0, .1, .01, .001, .0001),
    # 'random_state': (0,),
}; param_grids.append(parmas_logreg)

parmas_knn_clf = {
    'n_neighbors': (2,3,4,5,6,7,8,9,10),
    'weights': ('uniform', 'distance'),
    'metric': ('euclidean', 'minkowski', 'manhattan'),
    'leaf_size': (5, 10, 15, 30),
    'algorithm': ('ball_tree', 'kd_tree', 'brute'),
}; param_grids.append(parmas_knn_clf)

params_sgd_clf = {
    'loss': ('log', 'modified_huber'), # ('hinge', 'log', 'modified_huber',␣
 ↪'squared_hinge', 'perceptron')
    'penalty': ('l2', 'l1', 'elasticnet'),
    'alpha': (1e-1, 1e-2, 1e-3, 1e-4),
    'max_iter': (50, 100, 150, 200, 500, 1000, 1500, 2000, 2500),
    'class_weight': (None, 'balanced'),
    'learning_rate': ('optimal',),
    'tol': (None, 1e-2, 1e-4, 1e-5, 1e-6),
    # 'random_state': (0,),
}; param_grids.append(params_sgd_clf)

kernel_type = 'svm-rbf-kernel'
params_svm_clf = {
    # 'gamma': (1e-7, 1e-4, 1e-3, 1e-2, 0.1, 1.0, 10, 1e+2, 1e+3, 1e+5, 1e+7),
    'gamma': (1e-5, 1e-3, 1e-2, 0.1, 1.0, 10, 1e+2, 1e+3, 1e+5),
    'max_iter':(1e+2, 1e+3, 2 * 1e+3, 5 * 1e+3, 1e+4, 1.5 * 1e+3),
    'degree': (1,2,4,8),
    'coef0': (.001, .01, .1, 0.0, 1.0, 10.0),
    'shrinking': (True, False),
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid',],
    'class_weight': (None, 'balanced'),
    'C': (1e-4, 1e-3, 1e-2, 0.1, 1.0, 10, 1e+2, 1e+3),
    'probability': (True,),
}; param_grids.append(params_svm_clf)

parmas_tree = {
    'splitter': ('random', 'best'),
    'criterion':('gini', 'entropy'),
    'max_features': (None, 'sqrt', 'log2'),
    'max_depth': (None, 3, 5, 7, 10,),
    'splitter': ('best', 'random',),
    'class_weight': (None, 'balanced'),
}; param_grids.append(parmas_tree)

parmas_random_forest = {
    'n_estimators': (3, 5, 7, 10, 30, 50, 70, 100, 150, 200),
    'criterion':('gini', 'entropy'),
    'bootstrap': (True, False),
    'min_samples_leaf': (1,2,3,4,5),
    'max_features': (None, 'sqrt', 'log2'),
    'max_depth': (None, 3, 5, 7, 10,),
    'class_weight': (None, 'balanced', 'balanced_subsample'),
}; param_grids.append(parmas_random_forest)
```

```
# Some variables to perform different tasks
# -----------------------------------------------------
N_CV, N_KERNEL, N_GS = 9, 5, 6;
nrows = N_KERNEL // 2 if N_KERNEL % 2 == 0 else N_KERNEL // 2 + 1;
ncols = 2; grid_size = [nrows, ncols]
```

[17]:
```
n_components=9
learning_curves_by_kernels(
# learning_curves_by_components(
    estimators_list[:], estimators_names[:],
    rescaledX, y,
    train_sizes=np.linspace(.1, 1.0, 10),
    n_components=9,
    pca_kernels_list=pca_kernels_list[0],
    verbose=0,
    by_pairs=True,
    savefigs=True,
    scoring='accuracy',
    figs_dest=os.path.join('figures', 'learning_curve', f"Pcs_{n_components}"),␣
 ↪ignore_func=True,
    # figsize=(20,5)
)
```

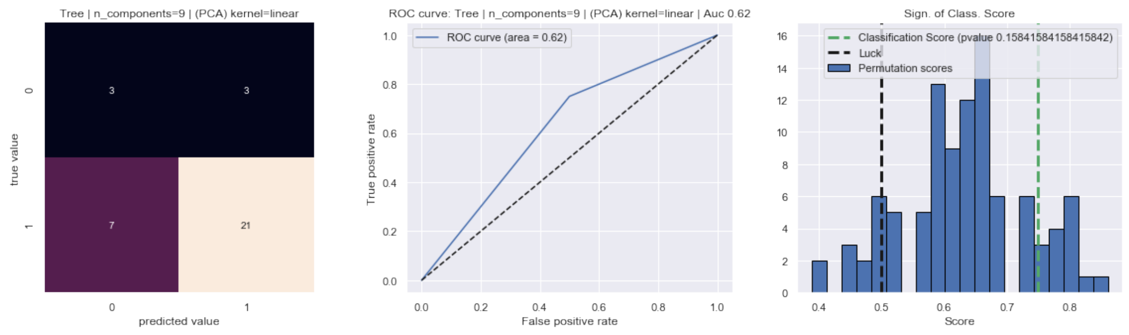
[18]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

```
<IPython.core.display.Javascript object>
```

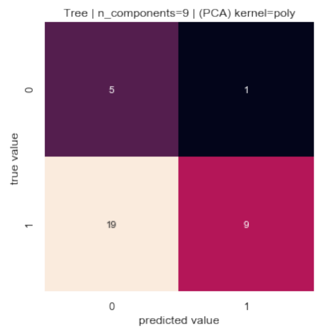
Learning Technique	Type of Learner	Type of Learning	Classification	Regression	Clustering	Outlier Detection
Decision Trees	*Non-parametric Model*	*Supervised Learning*	*Supported*	*Supported*	*Not-Supported*	*Not-Supported*

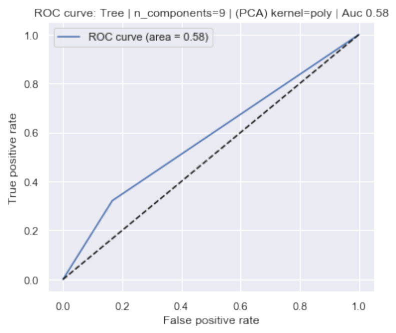
[19]:
```
plot_dest = os.path.join("figures", "n_comp_9_analysis", "grid_search")
X = rescaledX

df_gs, df_auc_gs, df_pvalue = grid_search_all_by_n_components(
    estimators_list=estimators_list[5], \
    param_grids=param_grids[4],
    estimators_names=estimators_names[5], \
    X=X, y=y,
    n_components=9,
    random_state=0, show_plots=False, show_errors=False, verbose=1,␣
 ↪plot_dest=plot_dest, debug_var=False)
df_9, df_9_auc = df_gs, df_auc_gs
```

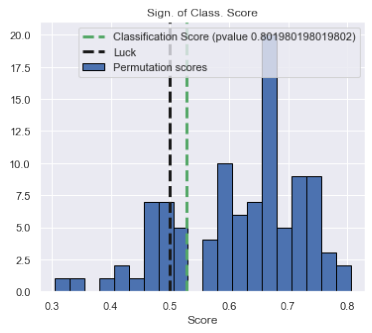
```
Kernel PCA: Linear | Tree
================================================================================
====================
              precision    recall  f1-score   support

     class 0       0.30      0.50      0.37         6
     class 1       0.88      0.75      0.81        28

    accuracy                           0.71        34
   macro avg       0.59      0.62      0.59        34
weighted avg       0.77      0.71      0.73        34

Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.84                0.71  0.62  0.15842
```



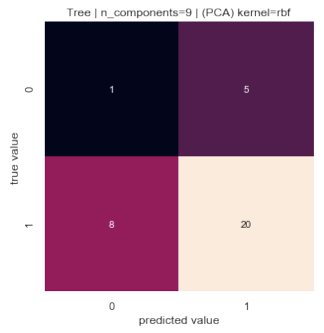
```
Kernel PCA: Poly | Tree
================================================================================
====================
              precision    recall  f1-score   support

     class 0       0.21      0.83      0.33         6
     class 1       0.90      0.32      0.47        28

    accuracy                           0.41        34
   macro avg       0.55      0.58      0.40        34
weighted avg       0.78      0.41      0.45        34

Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.92                0.41  0.58  0.80198
```

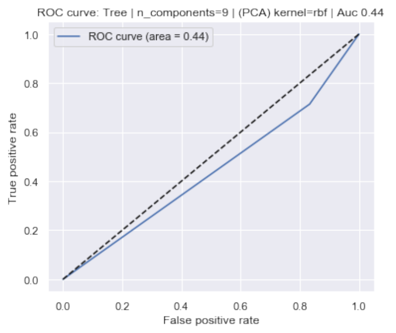


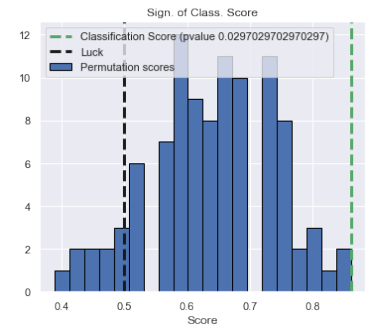




```
Kernel PCA: Rbf | Tree
================================================================================
====================
              precision    recall  f1-score   support

     class 0       0.11      0.17      0.13         6
     class 1       0.80      0.71      0.75        28

    accuracy                           0.62        34
   macro avg       0.46      0.44      0.44        34
weighted avg       0.68      0.62      0.65        34

Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.91              0.62  0.44  0.02970
```

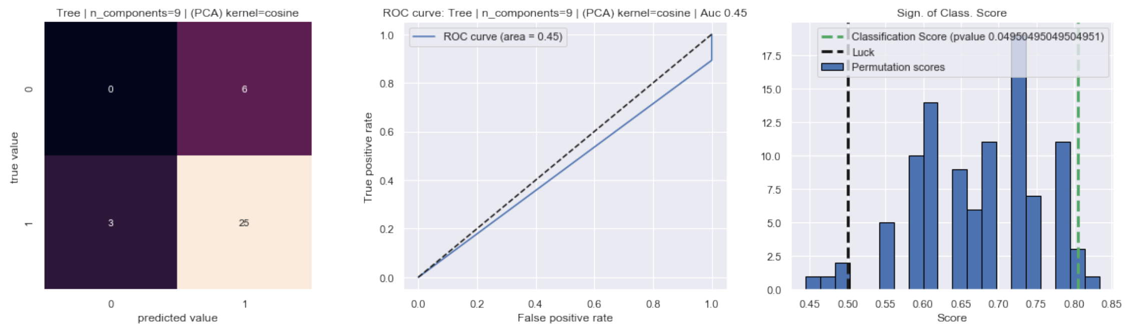




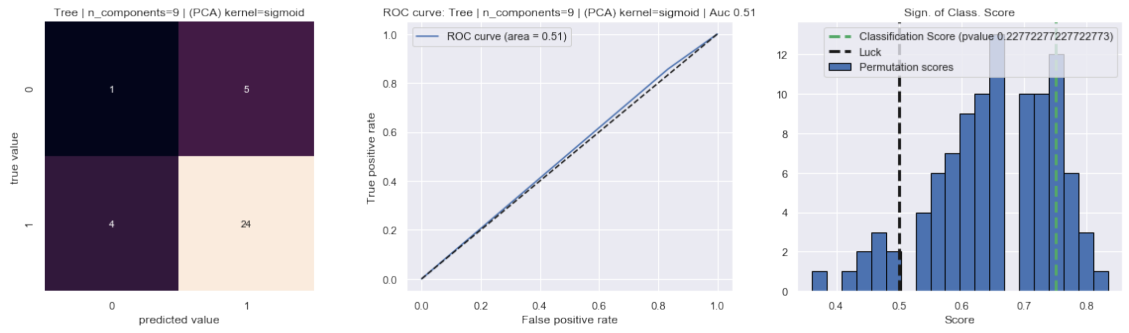


```
Kernel PCA: Cosine | Tree
================================================================================
====================
              precision    recall  f1-score   support

     class 0       0.00      0.00      0.00         6
     class 1       0.81      0.89      0.85        28
```

```
    accuracy                           0.74        34
   macro avg       0.40      0.45      0.42        34
weighted avg       0.66      0.74      0.70        34

Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.86              0.74  0.45  0.04950
```



```
Kernel PCA: Sigmoid | Tree
================================================================================
====================
              precision    recall  f1-score   support

     class 0       0.20      0.17      0.18         6
     class 1       0.83      0.86      0.84        28

    accuracy                           0.74        34
   macro avg       0.51      0.51      0.51        34
weighted avg       0.72      0.74      0.73        34

Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.81              0.74  0.51  0.22772
```



Looking at the results obtained running grid-search algorithm applied to Decision Tree Classifier we can rougly saying that the different models will obtain performances with default classification thresholds that are not as good as the results obtained from previous models, also because the models do not show Roc Curve along with their corresponding Auc scores that enable us saying that such models can perform well during inference also varying the default threshold. In particular we can say what follows: - looking at **Linear kernel-Pca based Decision Tree Classifier**, we notice that with the default threshold the model obtain high precision and recall for class 1 examples, meaning it is able to correctly classify most of the samples from class 1 as well as few examples from class 0 are exchanged as belonging to class 1. However speaking about class 0 we notice that we have obtained a very low precision and a 50 percent of recall that means that the model with default threshold misclassifyes half of the samples from class 0 and we are not really sure that what we have classified as the class 0 instance it really belong to class 0. Finally looking at Roc Curve we can observe that even from the very beginning the Sensitivity and 1-Specificity grow linearly with a slope value sligthly bigger than the slope of referring curve represented by Random Classifier, however at a given point the slope changes and as the thresholds approaches to higher values and the curve approaches to the top the slope decreases importantly, instead the value of Auc score account for 0.62.

- while looking at **Poly kernel-Pca based Decision Tree Model**, we can clearly understand that such a classifier is not good enough to be exploited for further inferences since it accounts for just .58 value of Auc score and observing Roc Curve we can conclude that it goes slightly better than the curve provided by random classifier. Moreover, the model when we adopt the default threshold seems to correctly classify most of the istances from class 0, but wrongly predict the class for instances of the opposite categories, in fact it is characterized from law precision referred to class 0, and since we want to correctly predict labels for both categories, here with such a classifier we are not able to satisfy such a constraint.
- the classifier corresponding to a **Rbf kernel-Pca based Decision Tree**, here is the model which leads to the worst performances, since the roc curve graphics is even worst than the random classifier and the roc curve accounts for a Auc score that is less than .5, more precisely just .44. So this result will be discarded, even if the model seems to correctly recognize samples from class 1 but wrongly predict labels for the class 0 samples, and again also here we have to state that we are not able to meet the constraint of correctly classify most of the data examples as we expect from a well defined classifier.
- referring to **Cosine kernel-Pca based Decision Tree Classifier**, when adopting a default threshold we notice that the model even if correctly classifyes all samples from class 0 leading to high recall for such a category, we can say also that the model has a low precision for class 0, meaning that the model confuses many samples from class 1 in fact is characterized from low value of recalll for the class 1, however when predicts a label equals to category one it is almost always sure about the choice. Thus, looking at Roc Curve and Auc score we can note that the model is characterized from a firt phase in which the Sensitivityu and 1-Specificity are not growing linearly following a line with a slope even lower than the one of Random Classifier as the preivous models, accounting for a AUC score equals to *45%*, we cannot neither decided to switch the labels for trying to train another classifier with such a new configuration, since the AUC does not suggest us to follow this new available and well-knwon strategy.
- Lastly the **Sigmoid kernel-Pca based Decision Tree Classifier**, with a default threshold of .5 for classification shows peformance scores that are more or less analog to those seen previously for *Rbf and Cosine kernel-Pca based Decision Tree Models*. The only difference is that the model get a AUC score much closer to that of Random Classifier. Again, also this

model is able to predict with high recall the samples belonging to class 1, instead wrongly predict class labels for those samples whcih come from class 0.

Significance Analysis: finally, when looking at the different graphics related to the test which aims at investigating the diagnostic power of our different models we have fine tuned for *SGD Classifier*, picking the best one for such a test we can notice that beacues of the *signficance level* α set equal to *0.05 that is 5% of chance to reject the Null-Hypothesis* H_0, we have obtained following results. Adopting the Decision Trees statistical learning technique for classification fine tuned as above with hyper-params selectd also depending on the kind of *kernel-trick adopted for kernel-Pca unsupervised technique*, we can claim that only two out of five trials lead to a *p-vlaue* worst than *selected significance level equal to 5%*, which are *Rbf- and Cosine-kernel Pca based Sgd Classifier*, so rejecting the *Null-Hypotesis* for those two cases will results into a *Type I Error*. While the remaining three cases, that are *Linear Poly-, and Sigmoid-kernel Pca based Sgd Classifier* have obtained a p-value over the range $[15, 90] \in R$ *in percet points*. Holthough we have at least two out of five classifier fine tuned that seem to allow us reject the *Null-Hypothesis* in order to justify the usae of such a models so fine-tuned and that accept the weights and hyper-parameters values we have discovered we end up saying that there is no one of the previous models that for sure we will accept to employ for inference and classification tasks due to their worst performance, rougly speaking. In other words, it seems that *Decision Tree Classification technique* does not work fine with such a small, unbalanced dataset.

Table Fine Tuned Hyper-Params(Decision Trees Classifier)

```
[20]: # create_widget_list_df([df_gs, df_auc_gs]) #print(df_gs); print(df_auc_gs)
      show_table_summary_grid_search(df_gs, df_auc_gs, df_pvalue)
```

```
[20]:              AUC(%)  P-Value(%)  Acc Train(%)  Acc Test(%) class_weight  \
      Tree linear    0.62       15.84          0.84         0.71         None
      Tree poly      0.58       80.20          0.92         0.41         None
      Tree rbf       0.44        2.97          0.91         0.62     balanced
      Tree cosine    0.45        4.95          0.86         0.74         None
      Tree sigmoid   0.51       22.77          0.81         0.74     balanced

                   criterion max_depth max_features splitter
      Tree linear       gini      None         None     best
      Tree poly      entropy      None         None     best
      Tree rbf          gini      None         sqrt     best
      Tree cosine    entropy         3         None   random
      Tree sigmoid      gini      None         None     best
```

Looking at the table shown above and obtained running grid-search algorithm applied to Decision Tree Classifier, only three out of five classifier get significant accuracy values, and those are Decision Tree created by means of *Linear, Cosine Sigmoid tricks adopted for kernel-Pca*, since we reaches accuracy up to *71%, 74%, and 74%* respectively. However, the worst classifier reveals to be *Polynomial based kernel Pca Decision Tree* with an accuracy on test set that reaches just *41%*, lastly the *Sigmoid based kernel Pca Decision Tree* was slightly worst than the previous three classifier that we have told were the best models, since the latter *Sigmoid based kernel Pca Decision Tree* has reached an accuracy score just 10 percent points less than the previous three, that is *62%*, however, not enough to consider it a good enough classifier. Looking at the hyper-parameters results shown from the summary table included in the report just above for DecisionTree Classification algorithm we can explain those results as follows:

- referring to the **class_weight hyper-param**, which represents weights associated with classes and if we disable such a setting so that it is valued to None, all classes are supposed to have weight one. Instead the *"balanced"* mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y)). We can notice that the choice of whether adopting a weighted strategy as balanced for performing training phase was chosen as best hyper-param value for *class_weight param* from just two out of five resulting fine tuned classifiers which were *Rbf, Sigmoid kernel-Pca based Decision Tree Classifier*, while the remaining adopted a uniform strategy. However discarding the worst classifier such a parameter half of the time was chose to be balanced and the remaining time to be uniform. In other words the choice of the kernel-trick at preprocessing time affected later the choice at grid-search training for *class_weight hyper-param.*

- reviewing **criterion decision trees' param**, which stands for the function to measure the quality of a split, where supported criteria are *"gini" for the Gini impurity and "entropy" for the information gain.* The most chosen *measure the quality* was the *gini for the Gini impurity*, in fact *Linear, Rbf, Sigmoid kernel-Pca based Decision Tree Classifier* selected this technique, while the remaining fine tuned classifiers *Poly and Cosine kernel-Pca based Decision Tree Classifiers* take advantages from exploiting *"entropy" for the information gain.* Again the choice at pre-processing time of a specific kernel-trick amongst those available for kernel-Pca unsupervised procedure leads to a particular *criterion* adopted from the trees based estimator when building the trees strucuture.

- looking at **max_depth trees' param**, which stands for the maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than *min_samples_split samples*, we clearly understand that for such a unbalanced and small dataset the best strategy while building the trees data strucutre was to keep growing the trees until all leaves are pure or until all leaves contain less than *min_samples_split samples.* In other words the hyper-param was set with a None value suggesting not to stop growing the trees at a given point, but rather to expand as much as possible. Only for *Cosine kernel-Pca based Decision Tree Classifiers* the best max_depth value for the hyper-parameter involved in this analysis was set to a really small size, just three nodes, which means that the preprocessed data via Cosine kernel-trick, does need to* much attributes to be taken into account before arriving into a leaf node.

- also describing **max_features hyper-parameter**, that is, the number of features to consider when looking for the best split, where if *"sqrt", then max_features=sqrt(n_features)*, *if "log2", then max_features=log2(n_features)*, and *if None, then max_features=n_features,* we can notice that the initial choice of the kernel-trick for pre-processing the data points within the dataset do not affect the final selection about the right technique to be adopted for calculating the number of features to be considered when looking for the best split. Moreover, since we are dealing with a small dataset, unbalanced with respect the class labels and also with a small number of features, we can reasonable understand that in most of the cases the classifier get better performance scores just considering a number of features equal to the available features after having performed the kernel-Pca algorithm. Only *Rbf kernel-Pca based Decision Tree Classifiers* selected a strategy tgat corresponds to *"sqrt", then max_features=sqrt(n_features)* for deciding the features to be identifyed for the best split.

- lastly, when speaking about the strategy used to choose the split at each node, where supported strategies are *"best" to choose the best split and "random" to choose the best random split*, we are saying that we are referring to **splitter hyper-parameter**. Here, for the trails we have carryed out, what we can say about such a hyper-param, is that since we are dealing with a small dataset, unbalanced and with not so much large number of features, also after having preprocessed it and discarded some useless features, is that the diverese fine tuned models in most of the cases adopted a best strategy, which requires more trainign time, while just for a single case corresponding to *Cosine kernel-Pca based Decision Tree Classifier*, the retrieved model seems to go better when adopint a random strategy.

The choice of kernel-Pca with a specific kernel-trick was decisive and affects the hyper-partameters set for building the If we imagine to build up an *Ensemble Classifier* from the family of *Average Methods*, which state that the underlying principle leading their creation requires to build separate and single classifiers than averaging their prediction in regression context or adopting a majority vote strategy for the classification context, we can claim that amongst the purposed decision trees classifier, for sure, we could employ the classifiers found from the **Linear, Rbf, Cosine and Sigmoid kernel-Pca based Decision Tree Classifier** because of their performance metrics and also because Ensemble Methods such as Bagging Classifier, usually work fine exploiting an ensemble of independent and fine tuned classifier differently from Boosting Methods which instead are based on weak learners.

1.3.1 Improvements and Conclusions

Extension that we can think of to better improve the analyses we can perform on such a relative tiny dataset many include, for preprocessing phases: - Selecting different *Feature Extraction ant Dimensionality Reduction Techniques* other than Pca or kernel Pca such as: *linear discriminant analysis (LDA)*, or *canonical correlation analysis (CCA) techniques* as a pre-processing step.

Extension that we can think of to better improve the analyses we can perform on such a relative tiny dataset many include, for training phases:

- Selecting different *Ensemble Methods, investigating both Average based and Boosting based Statistical Learning Methods.*

Extension that we can think of to better improve the analyses we can perform on such a relative tiny dataset many include, for diagnostic analyses after having performed train and test phases:

- Using other measures, indicators and ghraphical plots such as the *Total Operating Characteristic (TOC)*, since also such a measure characterizes diagnostic ability while revealing more information than the ROC. In fact for each threshold, ROC reveals two ratios, TP/(TP + FN) and FP/(FP + TN). In other words, ROC reveals hits/(hits + misses) and false alarms/(false alarms + correct rejections). On the other hand, TOC shows the total information in the contingency table for each threshold. Lastly, the TOC method reveals all of the information that the ROC method provides, plus additional important information that ROC does not reveal, i.e. the size of every entry in the contingency table for each threshold.

1.3.2 References

- Data Domain Information part:
  - (Deck) https://en.wikipedia.org/wiki/Deck_(bridge)
  - (Cantilever bridge) https://en.wikipedia.org/wiki/Cantilever_bridge

  - (Arch bridge) https://en.wikipedia.org/wiki/Deck_(bridge)
- Machine Learning part:
  - (Theory Book) https://jakevdp.github.io/PythonDataScienceHandbook/
  - (Decsion Trees) https://scikit-learn.org/stable/modules/tree.html#tree
  - (SVM) https://scikit-learn.org/stable/modules/svm.html
  - (PCA) https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
- Chart part:
  - (Seaborn Charts) https://acadgild.com/blog/data-visualization-using-matplotlib-and-seaborn
- Markdown Math part:
  - https://share.cocalc.com/share/b4a30ed038ee41d868dad094193ac462ccd228e2/Homework%20/HW%2
%20Markdown%20and%20LaTeX%20Cheatsheet.ipynb?viewer=share
  - https://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Typesetting%20Equations.htn

others

- Plots:
  - (Python Plot) https://www.datacamp.com/community/tutorials/matplotlib-tutorial-python?utm_source=adwords_ppc&utm_campaignid=898687156&utm_adgroupid=48947256715&ut
299261629574:dsa-473406587955&utm_loc_interest_ms=&utm_loc_physical_ms=1008025&gclid=C
_j1BRDkARIsAJcfmTFu4LAUDhRGK2D027PHiqIPSlxK3ud87Ek_lwOu8rt8A8YLrjFiHqsaAoLDEA
- Third Party Library:
  - (statsmodels) https://www.statsmodels.org/stable/index.html#
- KDE:
  - (TUTORIAL) https://jakevdp.github.io/blog/2013/12/01/kernel-density-estimation/
- Metrics:
  - (F1-Accuracy-Precision-Recall) https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c