Data Space Report (Official) - Knn Analysis.v1.0.1

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

[1]:
```
from utils.all_imports import *;
%matplotlib inline
```

```
None
```

[2]:
```
# Set seed for notebook repeatability
np.random.seed(0)
```

[3]:
```
# 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)
```

[4]:
```
columns_2_avoid = ['ERECTED', 'LENGTH', 'LOCATION']
```

[5]:
```
# 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

[6]:
```
show_table_pc_analysis(X=rescaledX)
```

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

[6]:
```
   # 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

[7]:
```
# Parameters to be tested for Cross-Validation Approach
# --------------------------------------------------------
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),
    'max_iter':(1e+2, 1e+3, 2 * 1e+3, 5 * 1e+3, 1e+4, 1.5 * 1e+3),
    'degree': (1,2,3,4,5),
    '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]
```

Learning Technique	Type of Learner	Type of Learning	Classification	Regression	Clustering
K-Nearest Neighbor	*Instance-based or Non-generalizing*	*Supervised and Usupervised Learning*	*Supported*	*Supported*	*Supported*

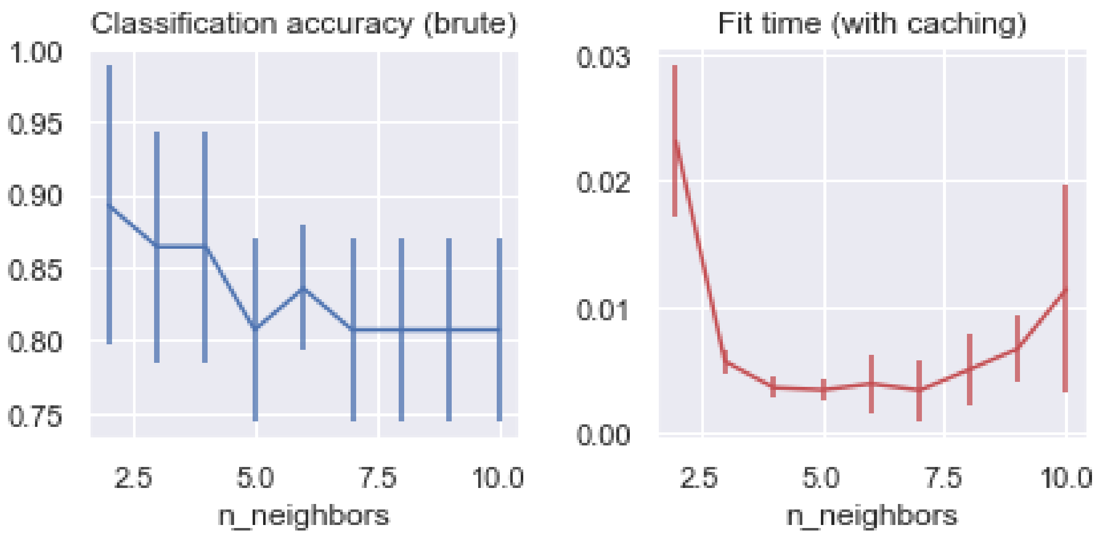
```
[8]: n_components=9
     learning_curves_by_kernels(
     # learning_curves_by_components(
         estimators_list[:], estimators_names[:],
         rescaledX, y,
         train_sizes=np.linspace(.1, 1.0, 10),
         n_components=9,
         pca_kernels_list=pca_kernels_list[0],
         verbose=0,
         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)
     )
```

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

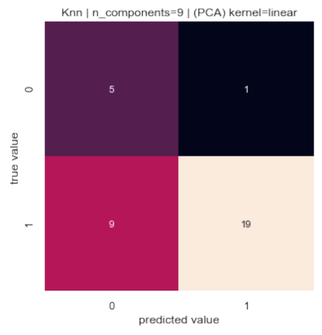
```
<IPython.core.display.Javascript object>
```

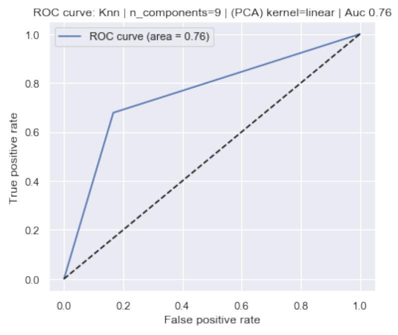
```
[10]: plot_dest = os.path.join("figures", "n_comp_9_analysis", "grid_search")
      X = rescaledX; pos = 2

      df_gs, df_auc_gs, df_pvalue = grid_search_all_by_n_components(
          estimators_list=estimators_list[pos], \
          param_grids=param_grids[pos-1],
          estimators_names=estimators_names[pos], \
          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
```

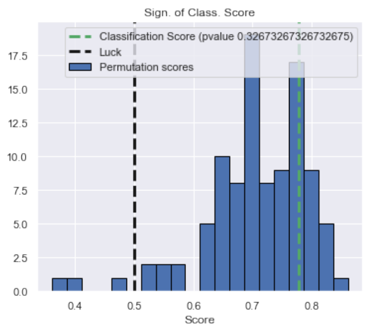
```
Kernel PCA: Linear | Knn
================================================================================
====================
              precision    recall  f1-score   support

     class 0       0.36      0.83      0.50         6
     class 1       0.95      0.68      0.79        28

    accuracy                           0.71        34
   macro avg       0.65      0.76      0.65        34
weighted avg       0.85      0.71      0.74        34
```



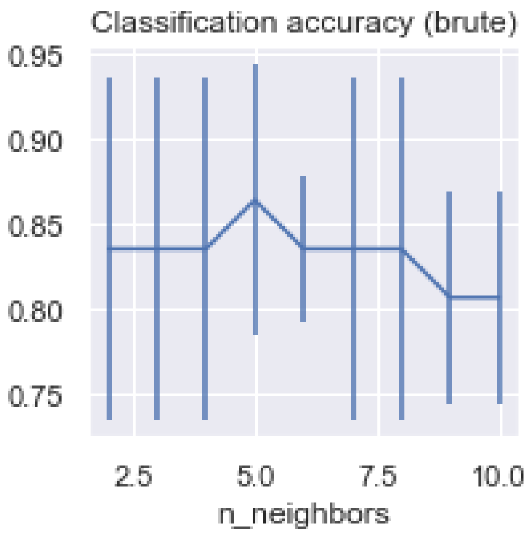
```
Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.92              0.71  0.76  0.32673
```

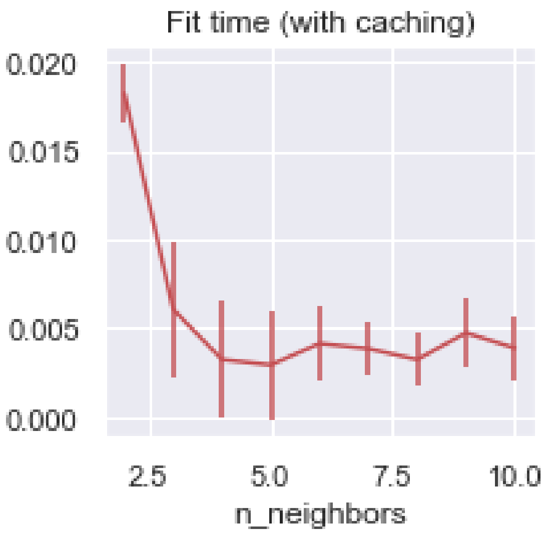




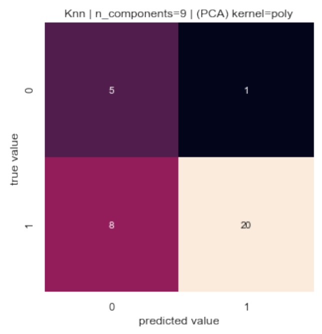


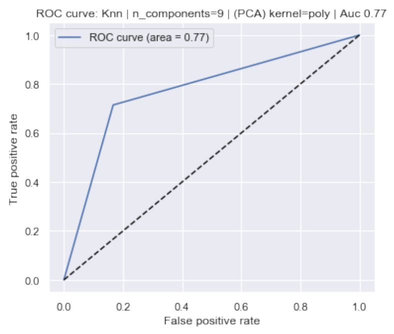
```
Kernel PCA: Poly | Knn
================================================================================
====================
              precision    recall  f1-score   support

     class 0       0.38      0.83      0.53         6
     class 1       0.95      0.71      0.82        28

    accuracy                           0.74        34
   macro avg       0.67      0.77      0.67        34
weighted avg       0.85      0.74      0.77        34
```

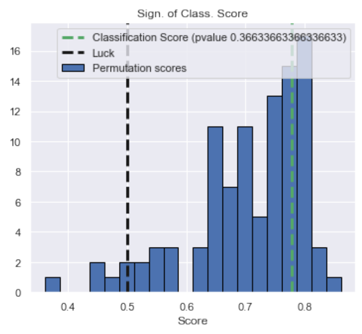




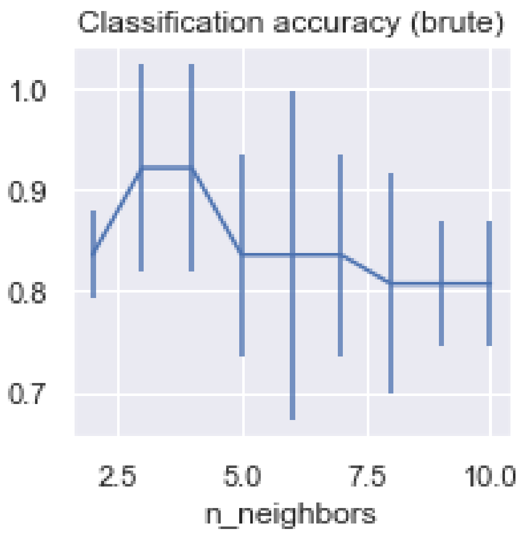
```
Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.89              0.74  0.77  0.36634
```

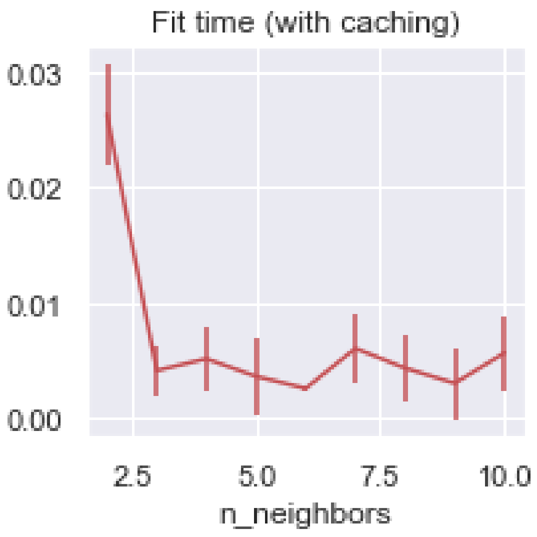




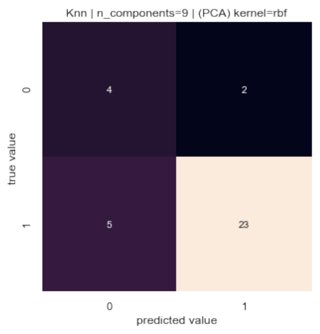


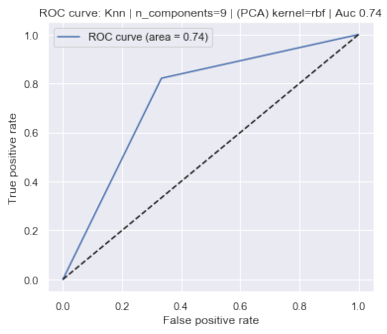
```
Kernel PCA: Rbf | Knn
================================================================================
====================
              precision    recall  f1-score   support

     class 0       0.44      0.67      0.53         6
     class 1       0.92      0.82      0.87        28

    accuracy                           0.79        34
   macro avg       0.68      0.74      0.70        34
weighted avg       0.84      0.79      0.81        34
```

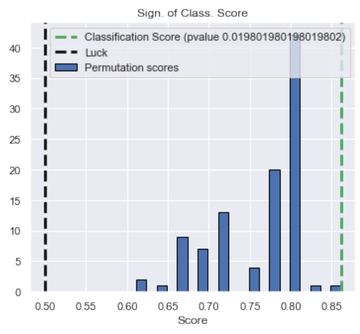




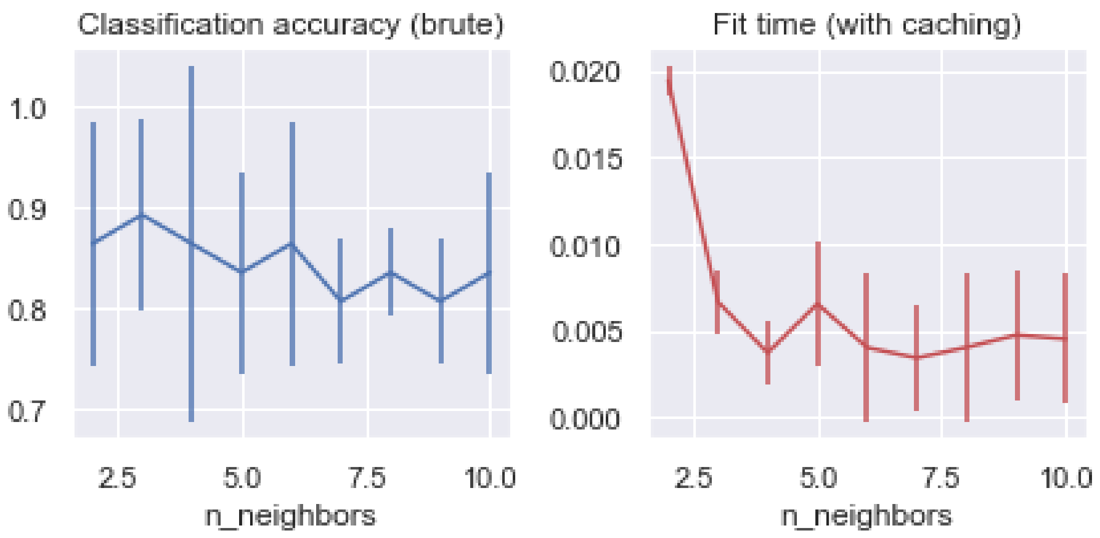
```
Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.95              0.79  0.74  0.01980
```



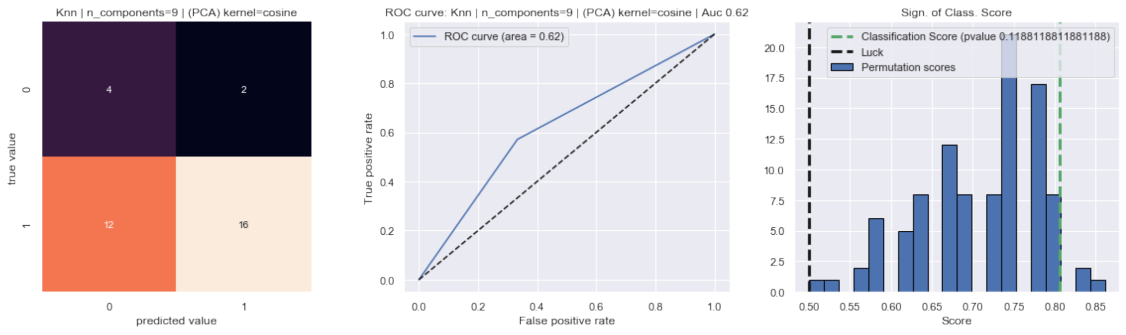




```
Kernel PCA: Cosine | Knn
================================================================================
====================
              precision    recall  f1-score   support

     class 0       0.25      0.67      0.36         6
     class 1       0.89      0.57      0.70        28

    accuracy                           0.59        34
   macro avg       0.57      0.62      0.53        34
weighted avg       0.78      0.59      0.64        34
```

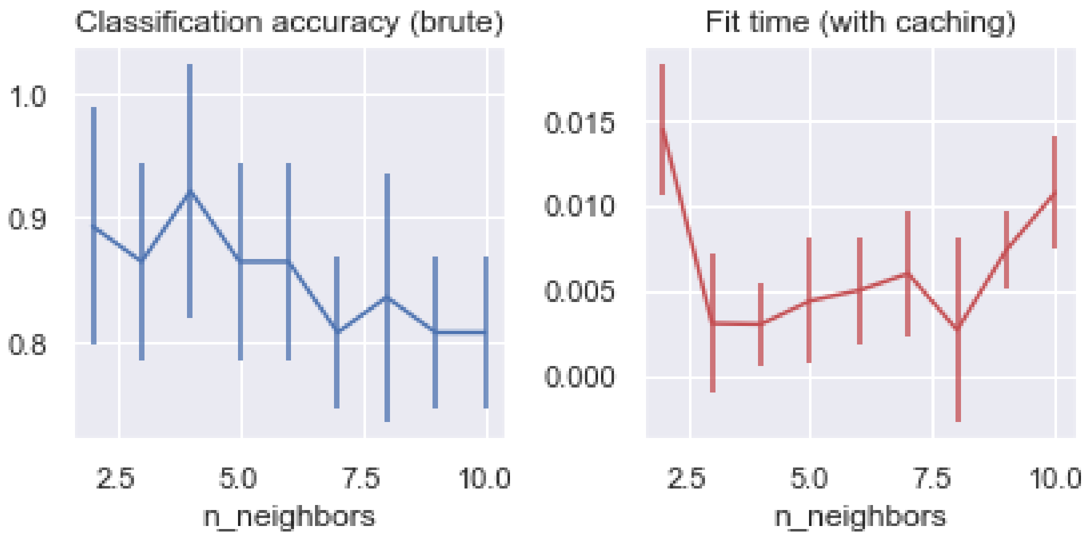


```
Best Score (CV-Train) Best Score (Test)    AUC  P-value
                 0.92              0.59   0.62  0.11881
```

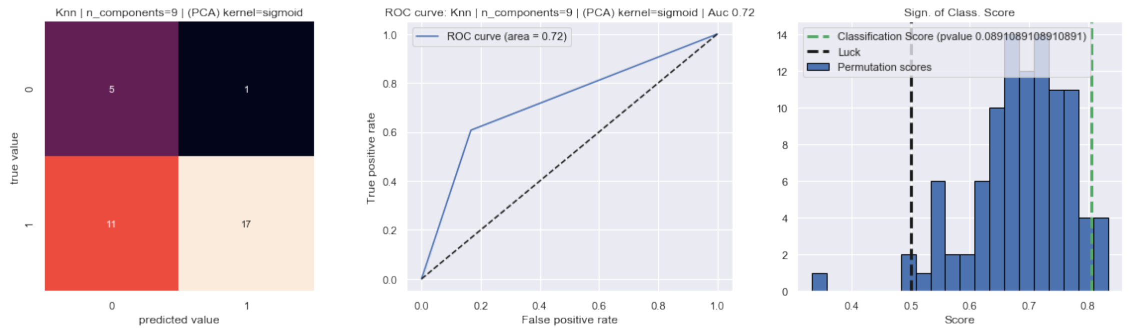


```
Kernel PCA: Sigmoid | Knn
```

```
================================================================================
====================
              precision    recall  f1-score   support

     class 0       0.31      0.83      0.45         6
     class 1       0.94      0.61      0.74        28

    accuracy                           0.65        34
   macro avg       0.63      0.72      0.60        34
weighted avg       0.83      0.65      0.69        34
```



```
Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.92                0.65  0.72  0.08911
```



Looking at the results obtained running *Knn Classifier* against our dataset splitted into training set and test set and adopting a different kernel trick applied to *kernel-Pca* unsupervised preprocessing method we can state generally speaking that all the such a *Statistical Learning technique* leads to a sequence of results that on average are more than appreciable beacues of the accuracy scores obtained at test time which compared against the same score but related to train phase allows us to understand that during the model creation the resulting classifiers do not overfit to the data, and even when the training score was high it was still lower than the scores in terms of accuracy obtained from the Logisti Regression Models which overfit to the data. Moreover, looking at the weighted values of *Recall, Precision, and F1-Scores* we can notably claim that the classifiers based on Knn obtained good performance and and except for one trial where we got lower and worst results, when *Sigmoid Trick* is selected, in the remaning cases have gotten remarkable results. More precisely we can say what follows: - speaking about **Linear kernel Pca based Knn Classifier**, when adoping the default threshold of *.5* for classification purposes we have a model that reaches an accuracy of *71%* at test time against an accuracy of *92%* at train step, while the Auc score reaches a value of *76%* with a Roc Curve that shows a behavior for which the model for a first set of thresholds let *TPR* grows faster than *FPR*, and only when we take into account larger thresholds we can understand that the trend is reversed. Looking at classification report we can see that the model has high precision and recall for class 1, so this means that the classifier has high confidence when predicting class 1 labels, instead it is less certain when predicting class 0 instances because has low precision, even if the model was able to predict correctly all the samples from class 0, leading to high recall. - observing **Polynomial kernel Pca based Knn Estimator**, we can notice that such a model exploiting a default threshold of *.5* reaches an accuracy of *74%* at test time against an accuracy of *89%* at train step, while the Auc score reaches a value of *77%*. What we can immediately understand is that the second model we have trained for Knn classifier is able to better generalize because obtained a higher accuracy score for test set which is also less far from train accuracy score, moreover the model has a slightly greater precision and recall when referring to class 1, while the precision and recall about class 0 seem to be more or less the same. - review **Rbf kernel Pca based Knn Classifier**, we can notice that such a model exploiting a default threshold of *.5* reaches an accuracy of *79%* at test time against an accuracy of *95%* at train step, while the Auc score reaches a value of *74%*. We can understand that even if this model, having selected *Rbf kernel Trick for kernel-Pca*, corresponds to the estimator that among Knn classifiers is the one with the best performance in terms of accuracy we notice that the corresponding auc score is less than the other first two analyzed trials where we end up saying that such classifiers lead to acceptable results. However, this method set with the hyper-params found by the grid-search algorithm reveals a higher value of precision related to class 0, meaning that *Rbf kernel Pca based Logisti Classifier* has a higher precision than previous models when classifying instances as belonging to class 0, while precisin and recall metrics for class 1 was more or less the same. This classifier is the one that we should select since it has higher precision values for better classifyng new instances. - looking at **Cosine kernel Pca based Knn Classifier**, we can notice that such a model exploiting a default threshold of *.5* reaches an accuracy of *59%* at test time against an accuracy of *92%* at train step, while the Auc score reaches a value of *62%*. We can clearly see that that such a model corresponds to the worst solution amongst the models we have trained exploiting *Knn classifier*, because we can state that due to a mostly lower accuracy score obtained at test time than the accuracy score referred to training time the classifier seems to have overfit to the data. In particular speaking about Precision and Recall scores about class 1, from classification report, the model seems to be mostly precise when predicting class 1 as label for the instances to be classified, however was misclassifying nearly half of the samples from class 1. Furthermore, the model also do not obtain fine results in terms of metrics when we focus on class 0 precision and recall. This model is the oen we should avoid, and do not exploit. - finally, referring to **Sigmoid kernel Pca based Knn Model**, we can notice that such a model exploiting a default threshold of *.5* reaches an accuracy of *65%* at test time against an accuracy of *92%* at train step, while the Auc score reaches a value of *72%*. It has values for performance metrics such as *precisio, recall, and F1-Score* which are more or less quyite similar to those of the first models trained for Knn classifier, that are those which are exploiting lienar and polynomial tricks, however this model misclassifyes larger number of class 1 instances lowering down the precision related to class 0, as well as lowering down recall from class 1. Also such a classifier with respect the first three trial is not sufficiently fine to be accepted so that we can exclude it from our choice.

```
[11]: # 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)
```

```
[11]:             AUC(%) P-Value(%) Acc Train(%) Acc Test(%)  algorithm leaf_size  \
Knn linear    0.76      32.67         0.92        0.71  ball_tree         5
Knn poly      0.77      36.63         0.89        0.74  ball_tree         5
Knn rbf       0.74       1.98         0.95        0.79  ball_tree         5
Knn cosine    0.62      11.88         0.92        0.59  ball_tree         5
Knn sigmoid   0.72       8.91         0.92        0.65  ball_tree         5

                metric n_neighbors   weights
Knn linear   euclidean           3  distance
Knn poly     euclidean           3  distance
Knn rbf      euclidean           7  distance
Knn cosine   euclidean           3  distance
Knn sigmoid  euclidean           4   uniform
```

Looking at the table dispalyed just above that shows the details about the selected values for hyper-parameters specified during grid search, in the different situations accordingly to the fixed kernel-trick for kernel Pca unsupervised method we can state that, referring to the first two columns of *Train and Test Accuracy*, we can recognize which trials lead to more overfit results such as for *Cosine, and Sigmoid Tricks* or less overfit solution such as in the case of *Linear, Polynomial, and Rbf Trick*. Speaking about the hyper-parameters, we can say what follows: - looking at the *algorithm paramter*, which can be set alternatively as *brute, kd-tree, and ball-tree* where each algorithm represents an differen strategy for implementing neighbor-based learning with pros and cons in terms of requested training time, memory usage and inference performance in terms of elapsed time, we can clearly understand that the choice of the kind of kernel trick for performing kernel-Pca does not care since all the trials preferred and selectd *ball-tree* strategy to solve the problem. It means that the grid search algorithm, when forced to try all the possible combination recorded such a choice as the best hyper-param which leads to building an expensive data strucuture which aims at integrating somehow distance information to achieve better performance scores. So, this should make us reason about the fact that it is still a good choice or we should re-run the procedure excluding such algorithm. In fact the answer to such a issue depend on the forecast about the number of queryes we aim to solve. If it will be huge in future than ball-tree algorthm was a good choice and a goo parameter included amongst the hyper-params grid of values, otherwise we should get rid of it. - referring to *leaf_size parameter*, we can notice that also here the choice of a specific kernel trick for performing kernel-Pca algorithm does not affect the value tha such a parameter has assumed amongst those proposed. However, recalling that leaf size hyper-param is used to monitor and control the tree-like structure of our solutions we can understand that since the value is pretty low the obtained trees were allowed to grow toward maximum depth. - speaking about *distance parameter*, the best solution through different trials was *Euclidean distance*, which also corresponds to the default choice, furthermore the choice of a kernel trick in the context of the other grid values was not affecting the choice of *distance parameter*. - *n_neighbors parameter* is the one which is most affected and influenced by the choice of kernel trick for performing the kernel-Pca preprocessing method, since three out of five trials found 3 as the best value which are *Linear, Poly and Cosine tricks*, however only the first two examples using such a low number of neighbors still obtained fine results, instead the best trial which is the classifier characterized from Rbf kernel trick for kernel-Pca has selected 7 as the best value for the number of neighbors meanning that such a classifier required a greater number of neighbor before estimating class label and also that the query time t solve the inference would be longer. - lastly, the *weights param* is involved when we want to assign a certain weight to examples used during classification, where usually farway points will have less effect and nearby point grow their importance. The most frequent choice was represented by the *distance strategy*, which assign a weigh value to each sample of train set involved during classification a value proportional to the inverse of the distance of that sample from the query point. Only the Sigmoid kernel trick case instead adopted a weights strategy which corresponds to the default choice which is the uniform strategy.

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 Knn classifier, for sure, we could employ the classifier foudn from the first three trials 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.

```
[12]: # show_histogram_first_sample(Xtrain_transformed, ytrain_, estimators_)
```

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.

1.4 References section

1.4.1 Main 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/
  - (Feature Extraction: PCA) https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.P
  - (Linear Model: Logistic Regression) https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  - (Neighbor-based Learning: Knn) https://scikit-learn.org/stable/modules/neighbors.html
  - (Stochastc Learning: SGD Classifier) https://scikit-learn.org/stable/modules/sgd.html#sgd
  - (Discriminative Model: SVM) https://scikit-learn.org/stable/modules/svm.html
  - (Non-Parametric Learning: Decsion Trees) https://scikit-learn.org/stable/modules/tree.html#tree
  - (Ensemble, Non-Parametric Learning: RandomForest) https://scikit-learn.org/stable/modules/ensemble.html#forest
- Metrics:
  - (F1-Accuracy-Precision-Recall) https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c
- Statistics:
  - (Correlation and dependence) https://en.wikipedia.org/wiki/Correlation_and_dependence
  - (KDE) https://jakevdp.github.io/blog/2013/12/01/kernel-density-estimation/
- Chart part:
  - (Seaborn Charts) https://acadgild.com/blog/data-visualization-using-matplotlib-and-seaborn
- Third Party Library:
  - (sklearn) https://scikit-learn.org/stable/index.html
  - (statsmodels) https://www.statsmodels.org/stable/index.html#

1.4.2 Others References

- 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
- Markdown Math part:
  - (Math Symbols Latex) https://oeis.org/wiki/List_of_LaTeX_mathematical_symbols
  - (Tutorial 1) https://share.cocalc.com/share/b4a30ed038ee41d868dad094193ac462ccd228e2/Homework%
%20Markdown%20and%20LaTeX%20Cheatsheet.ipynb?viewer=share
  - (Tutorial 2) https://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Typesetting%20E

```
[ ]:
```