Data Space Report - Pittsburgh's Bridge Dataset

Francesco Maria Chiarlo, matricola s253666

2020-June-16

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 on a relatively small dataset as the one employed within this report, that is Pittsburgh Bridgesdataset.

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

Contents

1 Data Space Report

1.1 Imports Section

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

# Set seed for notebook repeatability
np.random.seed(0)
```

```
None
```

1.1.1 Dataset's Attributes Description

The analyses that I aim at accomplishing while using as means the methods or approaches provided by both Statistical Learning and Machine Learning fields, concern the dataset Pittsburgh Bridges, and what follows is a overview and brief description of the main characteristics, as well as, basic information about this precise dataset.

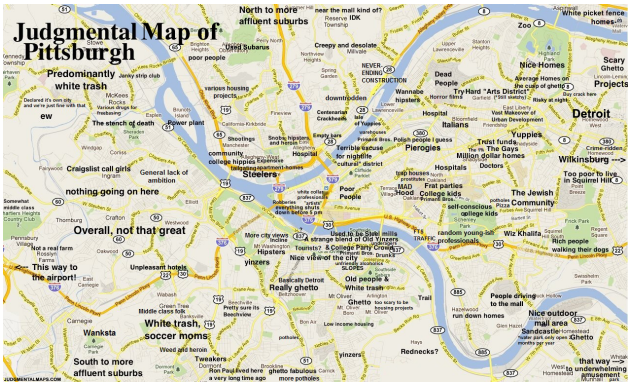
The Pittsburgh Bridges dataset is a dataset available from the web site called mainly *"UCI Machine Learing Repository"*, which is one of the well known web site that let a large amount of different datasets, from different domains or fields, to be used for machine-learning research and which have been cited in peer-reviewed academic journals.

In particular, the dataset I'm going to treat and analyze, which is Pittsburgh Bridges dataset, has been made freely available from the Western Pennsylvania Regional Data Center (WPRDC), which is a project led by the University Center of Social and Urban Research (UCSUR) at the University of Pittsburgh ("University") in collaboration with City of Pittsburgh and The County of Allegheny in Pennsylvania. The WPRDC and the WPRDC Project is supported by a grant from the Richard King Mellon Foundation.

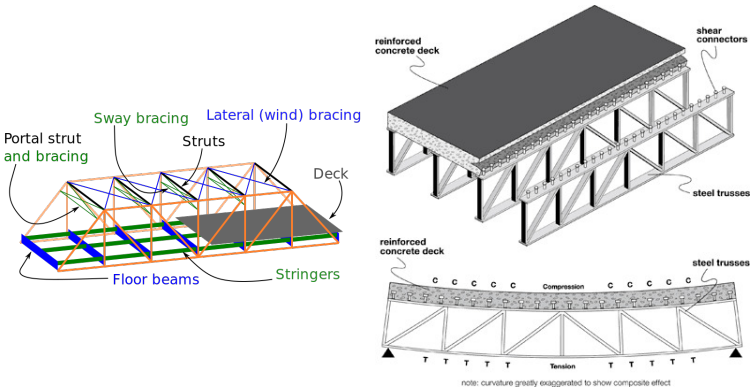
In order to be more precise, from the official and dedicated web page, within UCI Machine Learning cite, Pittsburgh Bridges dataset is a dataset that has been created after the works of some co-authors which are: - Yoram Reich & Steven J. Fenves from Department of Civil Engineering and Engineering Design Research Center Carnegie Mellon University Pittsburgh, PA 15213

The Pittsburgh Bridges dataset is made of up to 108 distinct observations and each of that data sample is made of 12 attributes or features where some of them are considered to be continuous properties and other to be categorical or nominal properties. Those variables are the following:

- **RIVER**: which is a nominal type variable that can assume the subsequent possible discrete values which are: A, M, O. Where A stands for Allegheny river, while M stands for Monongahela river and lastly O stands for Ohio river.
- **LOCATION**: which represents a nominal type variable too, and assume a positive integer value from 1 up to 52 used as categorical attribute.

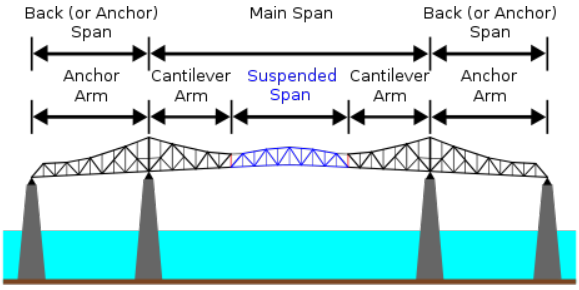


- **ERECTED**: which might be either a numerical or categorical variable, depending on the fact that we want to aggregate a bunch of value under a categorical quantity. What this means is that, basically such attribute is made of date starting from 1818 up to 1986, but we may imagine to aggregate somehow these data within a given category among those suggested, that are CRAFTS, EMERGENING, MATURE, MODERN.
- **PURPOSE**: which is a categorical attribute and represents the reason why a particular bridge has been built, which means that this attribute represents what kind of vehicle can cross the bridge or if the bridge has been made just for people. For this reasons the allowd values for this attributes are the following: WALK, AQUEDUCT, RR, HIGHWAY. Three out of four are self explained values, while RR value that might be tricky at first glance, it just stands for railroad.
- **LENGTH**: which represents the bridge's length, is a numerical attribute if we just look at the real number values that go from 804 up to 4558, but we can again decide to handle or arrange such values so that they can be grouped into range of values mapped into SHORT, MEDIUM, LONG so that we can refer to a bridge's length by means of these new categorical values.
- **LANES**: which is a categorical variable which is represented by numerical values, that are 1, 2, 4, 6 which indicate the number of distinct lanes that a bridge in Pittsburgh city may have. The larger the value the wider the bridge.
- **CLEAR-G**: specifies whether a vertical navigation clearance requirement was enforced in the design or not.
- **T-OR-D**: which is a nominal attribute, in other words, a categorical attribute that can assume THROUGH, DECK values. In order to be more precise, this samples attribute deals with structural elements of a bridge. In fact, a deck is the surface of a bridge and this structural element, of bridge's superstructure, may be constructed of concrete, steel, open grating, or wood. On the other hand, a through arch bridge, also known as a half-through arch bridge or a through-type arch bridge, is a bridge that is made from materials such as steel or reinforced concrete, in which the base of an arch structure is below the deck but the top rises above it.
- **MATERIAL**: which is a categorical or nominal variable and is used to describe the bridge telling which is the main or core material used to build it. This attribute can assume one of the possible, following values which are: WOOD, IRON, STEEL. Furthermore, we expect to see somehow a bit of correlation between the values assumed by the pairs



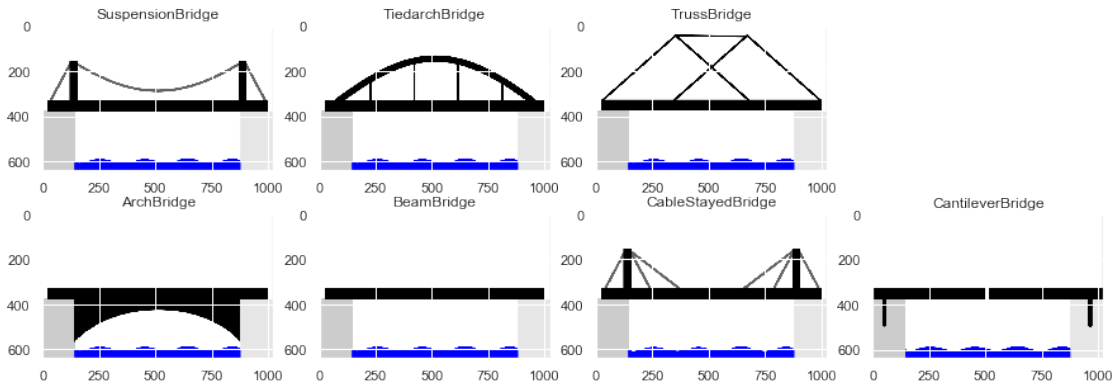
represented by T-OR-D and MATERIAL columns, when looking just to them.

- **SPAN**: which is a categorical or nominal value and has been recorded by means of three possible values for each sample, that are SHORT, MEDIUM, LONG. This attribute, within the field of Structural Engineering, is the distance between two intermediate supports for a structure, e.g. a beam or a bridge. A span can be closed by a solid beam or by a rope. The first kind is used for bridges, the second one for power lines, overhead telecommunication lines, some type of antennas or for aerial tramways.



- **REL-L**: which is a categorical or nominal variable and stands for relative length of the main span of the bridge to the total crossing length, it can assume three possible values that are S, S-F, F.
- Lastly, **TYPE** which indicates as a categorical or nominal attributes what type of bridge each record represents, among the possible 6 distinct classes or types of bridges that are: WOOD, SUSPEN, SIMPLE-T, ARCH, CANTILEV, CONT-T.

[3]:
```
# Show TYPE of Bridges
# ------------------------
show_bridges_types_images()
```



1.1.2 Data Investigation

The aim of this chapter is to get in the data, that are available within Pittsburgh Bridge Dataset, in order to investigate a bit more in to detail and generally speaking deeper the main or high level statistics quantities, such as mean, median, standard deviation of each attribute, as well as displaying somehow data distribution for each attribute by means of histogram plots. This phase allows or enables us to decide which should be the best feature to be selected as the target variable, in other word the attribute that will represent the dependent variable with respect to the remaining attributes that instead will play the role of predictors and independent variables, as well.

In order to investigate and explore our data we make usage of *Pandas library*. We recall mainly that, in computer programming, Pandas is a software library written for the Python programming language* for *data manipulation and analysis*. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software and a interesting and funny things about such tool is that the name is derived from the term “panel data”, an econometrics term for data sets that include observations over multiple time periods for the same individuals. We also note that as the analysis proceeds we will introduce other computer programming as well as programming libraries that allow or enable us to fulfill our goals.

Initially, once I have downloaded from the provided web page the dataset with the data samples about Pittsburgh Bridge we load the data by means of functions available using python library’s pandas. We notice that the overall set of data points is large up to 108 records or rows, which are sorted by Erected attributes, so this means that are sorted in decreasing order from the oldest bridge which has been built in 1818 up to the most modern bridge that has been erected in 1986. Then we display the first 5 rows to get an overview and have a first idea about what is inside the overall dataset, and the result we obtain by means of head() function applied onto the fetched dataset is equals to what follows:

1.1.3 Read Input Data

[4]:
```
# Some global script variables
#␣
 ↪--------------------------------------------------------------------------␣
 ↪#
dataset_path, dataset_name, column_names, TARGET_COL = \
    get_dataset_location() # Info Data to be fetched
estimators_list, estimators_names = get_estimators() # Estimator to be␣
 ↪trained

# variables used for pass through arrays used to store results
pos_gs = 0; pos_cv = 0

# 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))
```

[5]:
```
# Parameters to be tested for Cross-Validation Approach
# ----------------------------------------------------
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]
```

[6]:
```
# READ INPUT DATASET
#␣
  ↪-------------------------------------------------------------------------␣
  ↪#
dataset = pd.read_csv(os.path.join(dataset_path, dataset_name),␣
  ↪names=column_names, index_col=0)
```

[7]:
```
# SHOW SOME STANDARD DATASET INFOS
#␣
  ↪-------------------------------------------------------------------------␣
  ↪#
print('Dataset shape: {}'.format(dataset.shape)); print(dataset.info())
```

```
Dataset shape: (108, 12)
<class 'pandas.core.frame.DataFrame'>
Index: 108 entries, E1 to E109
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   RIVER     108 non-null    object
 1   LOCATION  108 non-null    object
 2   ERECTED   108 non-null    int64
 3   PURPOSE   108 non-null    object
 4   LENGTH    108 non-null    object
 5   LANES     108 non-null    object
 6   CLEAR-G   108 non-null    object
 7   T-OR-D    108 non-null    object
 8   MATERIAL  108 non-null    object
 9   SPAN      108 non-null    object
 10  REL-L     108 non-null    object
 11  TYPE      108 non-null    object
dtypes: int64(1), object(11)
memory usage: 11.0+ KB
None
```

[8]:
```
# SHOWING FIRSTS N-ROWS AS THEY ARE STORED WITHIN DATASET
#␣
  ↪-------------------------------------------------------------------------␣
  ↪#
dataset.head(5)
```

[8]:
	RIVER	LOCATION	ERECTED	PURPOSE	LENGTH	LANES	CLEAR-G	T-OR-D	MATERIAL	\
E1	M	3	1818	HIGHWAY	?	2	N	THROUGH	WOOD	
E2	A	25	1819	HIGHWAY	1037	2	N	THROUGH	WOOD	
E3	A	39	1829	AQUEDUCT	?	1	N	THROUGH	WOOD	
E5	A	29	1837	HIGHWAY	1000	2	N	THROUGH	WOOD	
E6	M	23	1838	HIGHWAY	?	2	N	THROUGH	WOOD	

	SPAN	REL-L	TYPE
E1	SHORT	S	WOOD
E2	SHORT	S	WOOD
E3	?	S	WOOD
E5	SHORT	S	WOOD
E6	?	S	WOOD

What we can notice from just the table above is that there are some attributes that are characterized by a special character that is '?' which stands for a missing value, so by chance there was not possibility to get the value for this attribute, such as for LENGTH and SPAN attributes. Analyzing in more details the dataset we discover that there are up to 6 different attributes, in the majority attributes with categorical or nominal nature such as CLEAR-G, T-OR-D, MATERIAL, SPAN, REL-L, and TYPE that contain at list one row characterized by the fact that one of its attributes is set to assuming '?' value that stands, as we already know for a missing value.

Here, we can follow different strategies that depends onto the level of complexity as well as accuracy we want to obtain or achieve for models we are going to fit to the data after having correctly pre-processed them, speaking about what we could do with missing values. In fact one can follow the simplest way and can decide to simply discard those rows that contain at least one attribute with a missing value represented by the '?' symbol. Otherwise one may also decide to follow a different strategy that aims at keeping also those rows that have some missing values by means of some kind of technique that allows to establish a potential substituting value for the missing one.

So, in this setting, that is our analyses, we start by just leaving out those rows that at least contain one attribute that has a missing value, this choice leads us to reduce the size of our dataset from 108 records to 70 remaining samples, with a drop of 38 data examples, which may affect the final results, since we left out more or less the 46% of the data because of missing values.

[9]:
```
# INVESTIGATING DATASET IN ORDER TO DETECT NULL VALUES
#␣
  ↪--------------------------------------------------------------------------␣
  ↪#
# print('Before preprocessing dataset and handling null values')
result = dataset.isnull().values.any(); # print('There are any null values ?␣
  ↪Response: {}'.format(result))
```

```
result = dataset.isnull().sum(); # print('Number of null values for each␣
 ↪predictor:\n{}'.format(result))
```

[10]:
```
# DISCOVERING VALUES WITHIN EACH PREDICTOR DOMAIN
#␣
 ↪-------------------------------------------------------------------------␣
 ↪#
columns_2_avoid = ['ERECTED', 'LENGTH', 'LOCATION', 'LANES']
list_columns_2_fix = show_categorical_predictor_values(dataset,␣
 ↪columns_2_avoid)
```

[11]:
```
# FIXING, UPDATING NULL VALUES CODED AS '?' SYMBOL
# WITHIN EACH CATEGORICAL VARIABLE, IF DETECTED ANY
#␣
 ↪-------------------------------------------------------------------------␣
 ↪#
print('"Before" removing \'?\' rows, Dataset dim:', dataset.shape)
for _, predictor in enumerate(list_columns_2_fix):
    dataset = dataset[dataset[predictor] != '?']
print('"After" removing \'?\' rows, Dataset dim: ', dataset.shape);␣
 ↪print('-' * 50)
_ = show_categorical_predictor_values(dataset, columns_2_avoid)
```

```
"Before" removing '?' rows, Dataset dim: (108, 12)
"After" removing '?' rows, Dataset dim:  (88, 12)
--------------------------------------------------
```

[12]:
```
# INTERMEDIATE RESULT FOUND
#␣
 ↪-------------------------------------------------------------------------␣
 ↪#
features_vs_values = preprocess_categorical_variables(dataset,␣
 ↪columns_2_avoid); print(dataset.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 88 entries, E1 to E90
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   RIVER     88 non-null     int64
 1   LOCATION  88 non-null     object
 2   ERECTED   88 non-null     int64
 3   PURPOSE   88 non-null     int64
 4   LENGTH    88 non-null     object
 5   LANES     88 non-null     object
```

```
 6   CLEAR-G   88 non-null     int64
 7   T-OR-D    88 non-null     int64
 8   MATERIAL  88 non-null     int64
 9   SPAN      88 non-null     int64
 10  REL-L     88 non-null     int64
 11  TYPE      88 non-null     int64
dtypes: int64(9), object(3)
memory usage: 8.9+ KB
None
```

[13]:
```
dataset.head(5)
```

[13]:
```
    RIVER LOCATION  ERECTED  PURPOSE LENGTH LANES  CLEAR-G  T-OR-D  MATERIAL␣
  ↪ \
E1      2        3     1818        2      ?     2        2       2         3
E2      1       25     1819        2   1037     2        2       2         3
E5      1       29     1837        2   1000     2        2       2         3
E7      1       27     1840        2    990     2        2       2         3
E8      1       28     1844        1   1000     1        2       2         1

    SPAN  REL-L  TYPE
E1     3      2     6
E2     3      2     6
E5     3      2     6
E7     2      2     6
E8     3      2     5
```

The next step is represented by the effort of mapping categorical variables into numerical variables, so that them are comparable with the already existing numerical or continuous variables, and also by mapping the categorical variables into numerical variables we allow or enable us to perform some kind of normalization or just transformation onto the entire dataset in order to let some machine learning algorithm to work better or to take advantage of normalized data within our pre-processed dataset. Furthermore, by transforming first the categorical attributes into a continuous version we are also able to calculate the *heatmap*, which is a very useful way of representing a correlation matrix calculated on the whole dataset. Moreover we have displayed data distribution for each attribute by means of histogram representation to take some useful information about the number of occurrences for each possible value, in particular for those attributes that have a categorical nature.

[14]:
```
# MAP NUMERICAL VALUES TO INTEGER VALUES
#␣
  ↪--------------------------------------------------------------------------␣
  ↪#
print('Before', dataset.shape)
columns_2_map = ['ERECTED', 'LANES']
for _, predictor in enumerate(columns_2_map):
```

```
    dataset = dataset[dataset[predictor] != '?']
    dataset[predictor] = np.array(list(map(lambda x: int(x),␣
  ↪dataset[predictor].values)))
print('After', dataset.shape); print(dataset.info())
```

```
Before (88, 12)
After (80, 12)
<class 'pandas.core.frame.DataFrame'>
Index: 80 entries, E1 to E90
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   RIVER     80 non-null     int64
 1   LOCATION  80 non-null     object
 2   ERECTED   80 non-null     int32
 3   PURPOSE   80 non-null     int64
 4   LENGTH    80 non-null     object
 5   LANES     80 non-null     int32
 6   CLEAR-G   80 non-null     int64
 7   T-OR-D    80 non-null     int64
 8   MATERIAL  80 non-null     int64
 9   SPAN      80 non-null     int64
 10  REL-L     80 non-null     int64
 11  TYPE      80 non-null     int64
dtypes: int32(2), int64(8), object(2)
memory usage: 7.5+ KB
None
```

[15]:
```
dataset.head(5)
```

[15]:

	RIVER	LOCATION	ERECTED	PURPOSE	LENGTH	LANES	CLEAR-G	T-OR-D	MATERIAL	SPAN	REL-L	TYPE
E1	2	3	1818	2	?	2	2	2	3	3	2	6
E2	1	25	1819	2	1037	2	2	2	3	3	2	6
E5	1	29	1837	2	1000	2	2	2	3	3	2	6
E7	1	27	1840	2	990	2	2	2	3	2	2	6
E8	1	28	1844	1	1000	1	2	2	1	3	2	5

[16]:
```
# MAP NUMERICAL VALUES TO FLOAT VALUES
#␣
 ↪--------------------------------------------------------------------------␣
 ↪#
# print('Before', dataset.shape)
columns_2_map = ['LOCATION', 'LANES', 'LENGTH']
for _, predictor in enumerate(columns_2_map):
    dataset = dataset[dataset[predictor] != '?']
    dataset[predictor] = np.array(list(map(lambda x: float(x),␣
 ↪dataset[predictor].values)))
```

[17]:
```
result = dataset.isnull().values.any() # print('After handling null␣
 ↪values\nThere are any null values ? Response: {}'.format(result))
result = dataset.isnull().sum() # print('Number of null values for each␣
 ↪predictor:\n{}'.format(result))
dataset.head(5)
```

[17]:

	RIVER	LOCATION	ERECTED	PURPOSE	LENGTH	LANES	CLEAR-G	T-OR-D	\
E2	1	25.0	1819	2	1037.0	2.0	2	2	
E5	1	29.0	1837	2	1000.0	2.0	2	2	
E7	1	27.0	1840	2	990.0	2.0	2	2	
E8	1	28.0	1844	1	1000.0	1.0	2	2	
E9	2	3.0	1846	2	1500.0	2.0	2	2	

	MATERIAL	SPAN	REL-L	TYPE
E2	3	3	2	6
E5	3	3	2	6
E7	3	2	2	6
E8	1	3	2	5
E9	1	3	2	5

[18]:
```
dataset.describe(include='all')
```

[18]:

	RIVER	LOCATION	ERECTED	PURPOSE	LENGTH	LANES	\
count	70.000000	70.000000	70.000000	70.000000	70.000000	70.000000	
mean	1.642857	25.438571	1911.542857	2.214286	1597.657143	2.857143	
std	0.723031	13.223347	36.010339	0.478308	780.237680	1.242785	
min	1.000000	1.000000	1819.000000	1.000000	840.000000	1.000000	
25%	1.000000	15.250000	1891.250000	2.000000	1000.000000	2.000000	
50%	1.500000	27.000000	1915.000000	2.000000	1325.000000	2.000000	
75%	2.000000	35.750000	1935.500000	2.000000	2000.000000	4.000000	
max	3.000000	49.000000	1978.000000	3.000000	4558.000000	6.000000	

	CLEAR-G	T-OR-D	MATERIAL	SPAN	REL-L	TYPE
count	70.000000	70.000000	70.000000	70.000000	70.000000	70.000000

```
mean    1.257143   1.814286   2.071429   1.742857   1.728571   3.457143
std     0.440215   0.391684   0.428054   0.629831   0.797122   1.575958
min     1.000000   1.000000   1.000000   1.000000   1.000000   1.000000
25%     1.000000   2.000000   2.000000   1.000000   1.000000   2.000000
50%     1.000000   2.000000   2.000000   2.000000   2.000000   4.000000
75%     1.750000   2.000000   2.000000   2.000000   2.000000   4.000000
max     2.000000   2.000000   3.000000   3.000000   3.000000   6.000000
```

1.1.4 Descriptive Statistics

After having performed the inital preprocessing phase about Pittsburgh Bridge Dataset, where we have clined the dataset from missing values as well as properly coded all the features, that are attributes and variables by which our dataset is made of, in order to reflect their own nature, wheter categorical or numerical, so continuous, we go ahead doing another step further, consisting in describing features properties by means of some usefull and well know tools coming from Descriptive Statistics area, that is a branch of the Statistics id considered as a whole.

In particular we are going to exploit some features that made up statistician's toolbox such as histograms, pie charts and the like, describing also their advantages as well as some of their backwards.

[19]:
```
# sns.pairplot(dataset, hue='T-OR-D', size=1.5)
```

Histograms: the main advantage of using such a chart is that i can be employed to describe the frequencies with which single values or a subset of distict values wihtin a range occurs for a given sample of observations, independently whether such a sample is representing a part of an entire population of examples and measurements, or the population itself, reminding that usually we deal with subsets or samples obtained or randomly sampled by an entire popualtion which might be real or just hypothetical and supposed. In particualr the advantages that Histogram graphs allow us to observe looking at a sampel of records and measurements are the following:

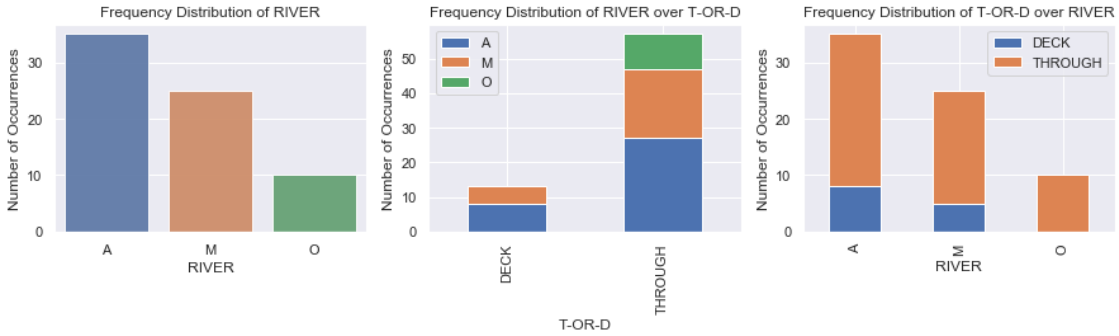
1. If the variable taken into account is acdto a continuous variable we may decide to discretize the range of possible values into a number of subintervals, that are also referred to as bins, and observe how the data is distributed into the different subintervals.
2. In particular, continuing from above, the histogram we might see can suggest us if the sample has one or more picks, describing which are the most occuring values or the most populated subintervalls, as well as the histogram follows a bell-like shape in order to spot also whether the graphs shows greater upper-tail or lower-tail, or a positive skew and altertnatively a negaitve skew. Where, in the former case usually we knwo that that the sample of data shows a greater probability of observing data measurements from the upper side of the bell-shaped graph, otherwise from the lower side of again a bell-shaped graph.
3. It is also important to say that generally speaking all these kind of observations and analyses are well suited for variables and featrues that assumes values that are continuous

in nature, such as height, weight, or as in our dataset LOCATION variable - Instead, if the variable under investigation is discrete or categorical in nature, the histogram graph is better called a bar graph and is a suitable choice for describing occourrencies or frequencies of different categories and classes, since someitmes there is not a natural order among the values such as Colors, even if we might find a natural order as dress's sizes.

Here, what follows is the sequence of several histograms created and illustrated to describe some other characteristics of the variables by which the dataset is made as well as to show the level or type of relationship of the frequency or the occurency of each value for a given attribute with the values assumed by the target variable we have selected amongst the overall variables.

[20]:
```
columns_2_avoid = ['ERECTED', 'LENGTH', 'LOCATION'];␣
  ↪show_frequency_distribution_predictor(dataset, predictor_name='RIVER',␣
  ↪columns_2_avoid=columns_2_avoid, features_vs_values=features_vs_values,␣
  ↪hue=TARGET_COL, verbose=1)
```

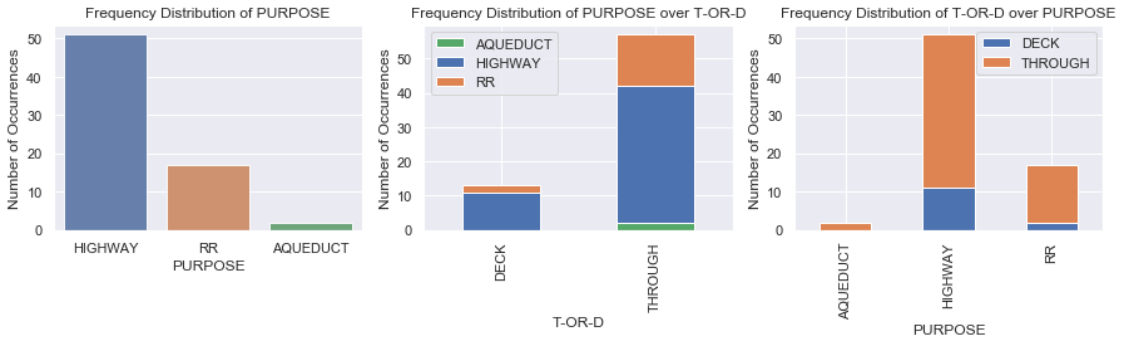
```
{'A': 1, 'M': 2, 'O': 3, 'Y': 4}
             A     M     O
DECK       8.0   5.0   NaN
THROUGH   27.0  20.0  10.0
```



The Histogram related to the frequency, in other sense the occurency, of RIVER datset's feature shows us that: - Among the three main rivers that cross the pittsburgh town, that are *Allegheny*, *Monongahela*, and *Ohio*, the one with the highest number of bridges the first Allegheny, followed by Monongahela, and finally the Ohio river whcih is also the converging river of the former two preciding rivers. - Instead, if we depict and illustrate the occurency, of RIVER datset's feature over our target variable T-OR-D dataset's feature we can understand that among the two binary values, that are DECK and THROUGH, the second seems the most exploited floor system for building bridges between the opposite edges of the rivers. Furthermore, speaking about bridges built around Ohio river just THROUGH structural element is the only technique adopted for those bridges. - What we can also sau about RIVER feature is that Allegheny and Monongahela show more or less the same number of bridges made from THROUGH surface, while for DECK surface Allegheny bits all the other rivers and Ohio does not figure among the rivers where there are bridges with DECK like structure at all.

[31]:
```
# show_frequency_distribution_predictor_v2(dataset,␣
  ↪predictor_name='ERECTED', columns_2_avoid=columns_2_avoid,␣
  ↪features_vs_values=features_vs_values, verbose=1)
show_frequency_distribution_predictor(dataset, predictor_name='PURPOSE',␣
  ↪columns_2_avoid=columns_2_avoid, features_vs_values=features_vs_values,␣
  ↪hue=TARGET_COL, verbose=1)
```

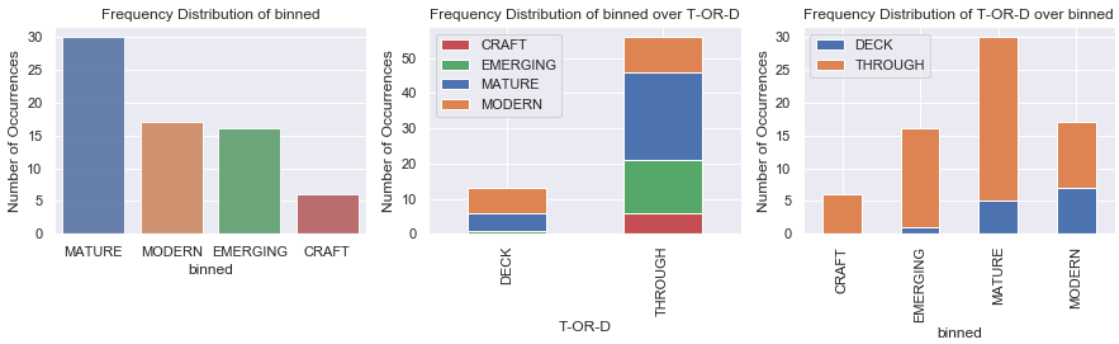
```
{'AQUEDUCT': 1, 'HIGHWAY': 2, 'RR': 3}
         AQUEDUCT  HIGHWAY    RR
DECK          NaN     11.0   2.0
THROUGH       2.0     40.0  15.0
```



The Histogram related to the frequency, in other sense the occurency, of PURPOSE datset's feature shows us that: - The Aqueduct bridges are only of THROUGH kind, and so, there are not occurences of type DECK which are exploited to carry on and support the Acqueduct system. - In both the two distinct classes, apart from Aqueduct kind of bridges that are just present for THROUGH class, the other two kind of brdiges which are for RR and Highway we can notice clearly that the latter are in proportion the most frequent kind of bridge, which means that the bridges we can found are most often exploited for supporting the traffic of labors that go to work by car.

[21]:
```
# show_frequency_distribution_predictor_v2(dataset,␣
  ↪predictor_name='ERECTED', columns_2_avoid=columns_2_avoid,␣
  ↪features_vs_values=features_vs_values, verbose=1)
show_frequency_distribution_predictor(dataset, predictor_name='ERECTED',␣
  ↪columns_2_avoid=columns_2_avoid, features_vs_values=features_vs_values,␣
  ↪hue=TARGET_COL, verbose=1)
```

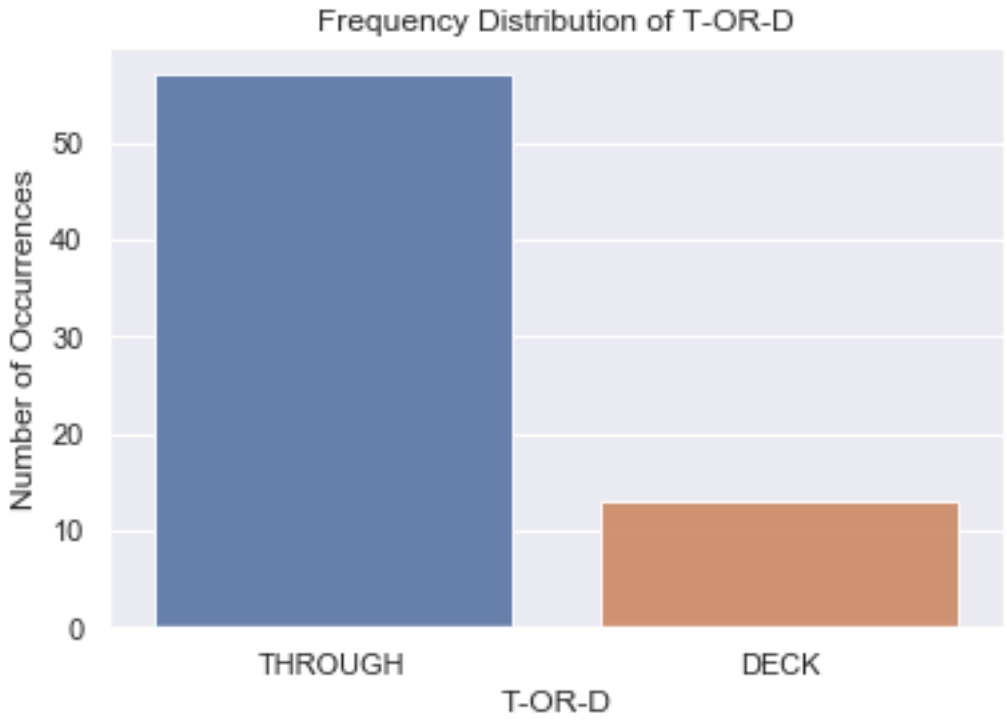
```
{'CRAFT': 1, 'EMERGING': 2, 'MATURE': 3, 'MODERN': 4}
         CRAFT  EMERGING  MATURE  MODERN
DECK       NaN       1.0     5.0     7.0
THROUGH    6.0      15.0    25.0    10.0
```



The Histogram related to the frequency, in other sense the occurency, of ERECTED datset's feature shows us that: - Mature bridges are the most frequent bridges, which means that the bridges have been built with more frequency in the recent period of time, also for substituting oldest bridges or for letting a greater number of labors to reach and enstablish into the city for finding job. - Then, follow Modern and Emerging bridges, which are more or less comparable in the sense of number of occurences of bridges for those two distinct period of time. - Lastly, Craft like bridges are the least frequent bridges, moreover, the bridges which have been built just of kind THROUGH.

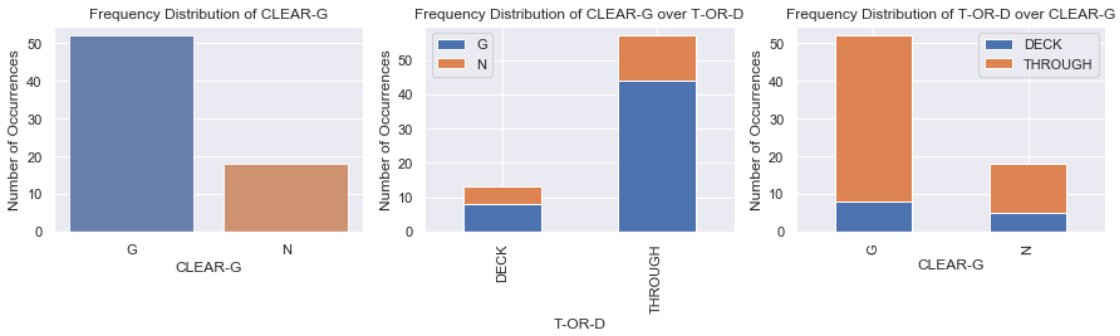
[21]:
```
# show_frequency_distribution_predictors(dataset, columns_2_avoid)
show_frequency_distribution_predictor(dataset, predictor_name='T-OR-D',␣
  ↪columns_2_avoid=columns_2_avoid, features_vs_values=features_vs_values,␣
  ↪verbose=1)
```

```
{'DECK': 1, 'THROUGH': 2}
```



[22]:
```
# show_frequency_distribution_predictors(dataset, columns_2_avoid)
show_frequency_distribution_predictor(dataset, predictor_name='CLEAR-G',␣
  ↪columns_2_avoid=columns_2_avoid, features_vs_values=features_vs_values,␣
  ↪hue=TARGET_COL, verbose=1)
```

```
{'G': 1, 'N': 2}
          G   N
DECK      8   5
THROUGH  44  13
```

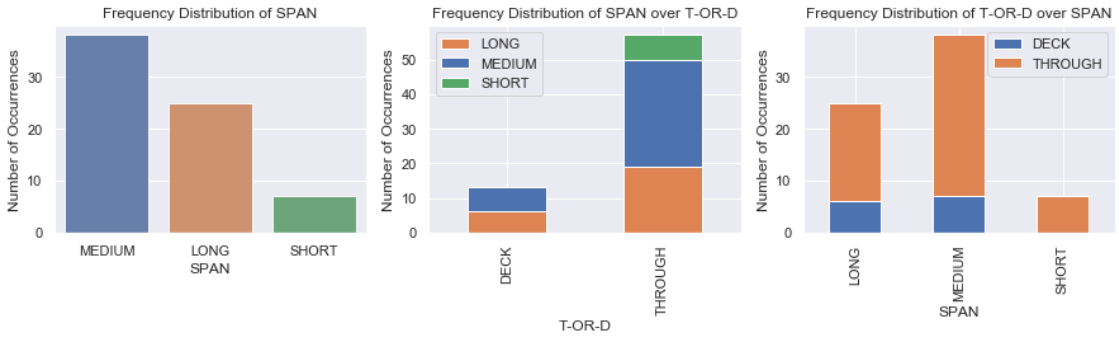


Instead looking at CLEAR-G feature we can notice that the *Vertical Clearance for navigation* is allowed for the majority of the bridges and when looing at the relationship of such a feature with the T-OR-D target variable we can see that THROUGH technology is the most adopted amongst the both the bridges that have or not gained the vertical Clearance for navigation, and in particular the THROUCH system is far more popular than DECK surface system in both G and N bridges, recalling us how the THROUGH technique become so important and widely

spread across time and space while speaking about bridge constructing.

[23]:
```
# show_frequency_distribution_predictors(dataset, columns_2_avoid)
show_frequency_distribution_predictor(dataset, predictor_name='SPAN',␣
  ↪columns_2_avoid=columns_2_avoid, features_vs_values=features_vs_values,␣
  ↪hue=TARGET_COL, verbose=1)
```

```
{'LONG': 1, 'MEDIUM': 2, 'SHORT': 3}
         LONG  MEDIUM  SHORT
DECK      6.0     7.0    NaN
THROUGH  19.0    31.0    7.0
```



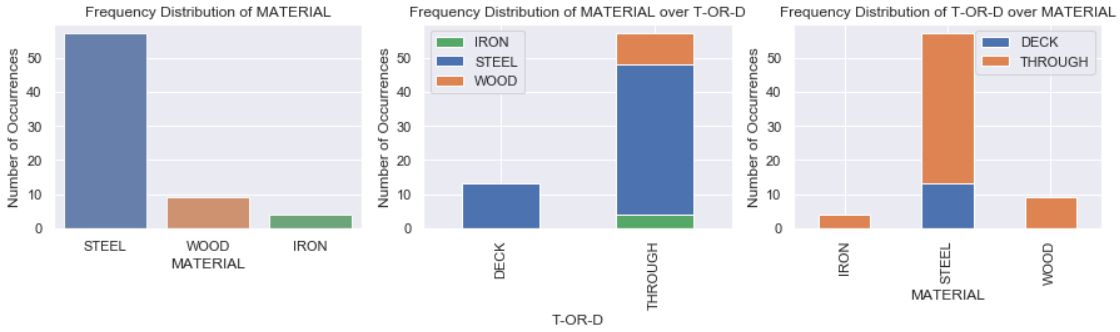
Span is the distance between two intermediate supports for a structure, e.g. a beam or a bridge. A span can be closed by a solid beam or by a rope. The first kind is used for bridges, the second one for power lines, overhead telecommunication lines, some type of antennas or for aerial tramways. With such a definition kept in mind what we can undestand is that:

- looking at the histogram graph about Occurency distribution of Bridge SPAN feature is that, since the three rivers are conisdered to be large riversa long most of their length considering the portion of them that cross the city of Pittsburgh, it becomes natural to observe that MEDIUM Span samples are the most occurring examples, while SHORT Span samples are the least frequent records and LONG span samples range in the between but seems to reach more closely the MEDIUM Span records.
- As usual, also here, we continue to observe that THROUGH brdiges are the ind of bridges analysing the T-OR-D feature that collect the majority of samples, while DECK bridges are just characterized by brdiges with LONG or MEDIUM Span feature and no SHORT span.
- Moreover, the ration between DECK and THROUGH reaches more or less 1 over 5, that is every 5 THROUGH bridges we find a DECK bridge with either LONG or MEDIUM Span elements.

[24]:
```
# show_frequency_distribution_predictors(dataset, columns_2_avoid)
show_frequency_distribution_predictor(dataset, predictor_name='MATERIAL',␣
  ↪columns_2_avoid=columns_2_avoid, features_vs_values=features_vs_values,␣
  ↪hue=TARGET_COL, verbose=1)
```

```
{'IRON': 1, 'STEEL': 2, 'WOOD': 3}
```

```
          IRON  STEEL  WOOD
DECK       NaN   13.0   NaN
THROUGH    4.0   44.0   9.0
```

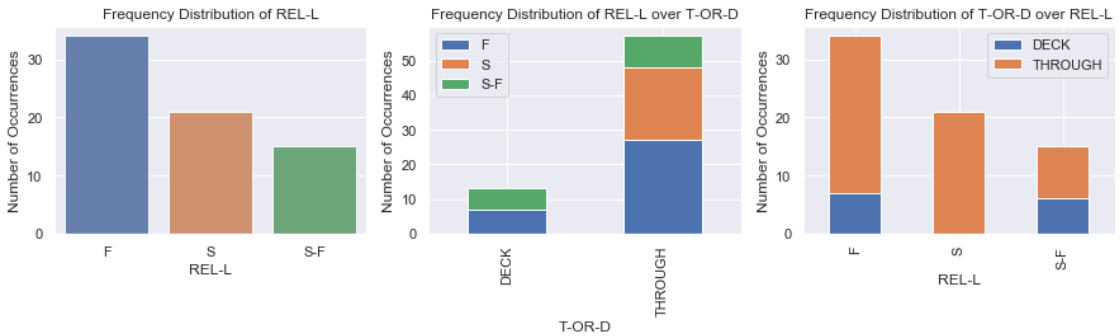


In the histohram graph illustrated above, we can clearly understand that:

- the STELL element is the most frequently exploited material for building bridges due to its strengthen and its restinance to the corrosion caused by the surrounding environment, while WOOD-like bridges are still present they are far less frequent than STEEL-like bridges but still have better properties than IRON bridges that are least frequent bridges since Iron leads to heavier bridges and Iron requires more extensive maintanance than Steel bridges and have less elastic properties than Wood brdiges.
- However, THROUGH-like bridges are the kind of bridges that present istances that have all examples of bridges with all the available materials that the dataset has shown, while DECK-like bridges exploit just Steel material for building bridges.

[25]:
```
# show_frequency_distribution_predictors(dataset, columns_2_avoid)
show_frequency_distribution_predictor(dataset, predictor_name='REL-L',␣
  ↪columns_2_avoid=columns_2_avoid, features_vs_values=features_vs_values,␣
  ↪hue=TARGET_COL, verbose=1)
```

```
{'F': 1, 'S': 2, 'S-F': 3}
             F     S  S-F
DECK       7.0   NaN  6.0
THROUGH   27.0  21.0  9.0
```



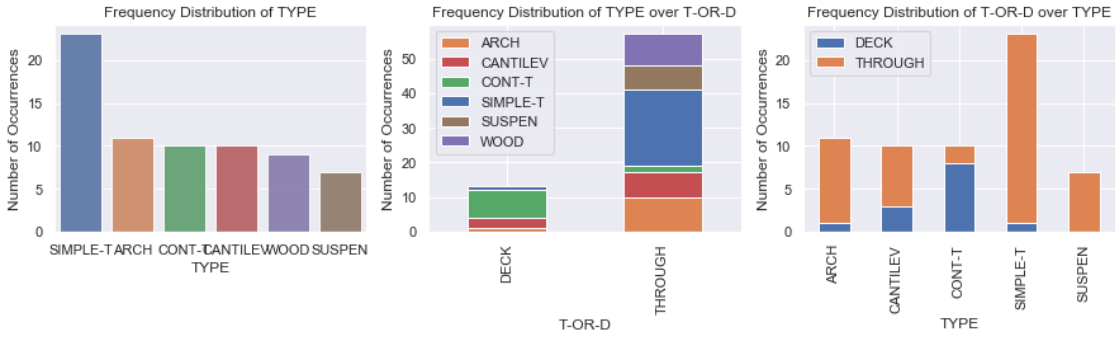
We knwo that The REL-L property is the relative length of the main span to the total crossing

length. With this short notion about such a feature in mind, what we can suggest observing ther first histogram depicted above is that:

- FULL kind of Bridge, shortly *F*, is the most frequent example of feature of the Pittsburgh bridges, also, the Through system is the bridge system that the most exploit or is characterized by FULL REL-L property.
- The, SMALL kind of Bridge, shortlu *S* is the second property in number of instances that show such a feature among the bridges and it is only a kind of feature shown only from THROUGH-like bridges, this means that we do not find bridges that show such a property amongs the DECK-like bridges.
- Lastly, an intermediate solution, represented by SMALL-FULL property, shortly *S-F*, is more or less present in both type of bridges that are classified with DECK or THROUGH system speaking about T-OR-D attribute.

[26]:
```
# show_frequency_distribution_predictors(dataset, columns_2_avoid)
show_frequency_distribution_predictor(dataset, predictor_name='TYPE',␣
  ↪columns_2_avoid=columns_2_avoid, features_vs_values=features_vs_values,␣
  ↪hue=TARGET_COL, verbose=1)
```

```
{'ARCH': 1, 'CANTILEV': 2, 'CONT-T': 3, 'SIMPLE-T': 4, 'SUSPEN': 5, 'WOOD': 6}
          ARCH  CANTILEV  CONT-T  SIMPLE-T  SUSPEN  WOOD
DECK       1.0       3.0     8.0       1.0     NaN   NaN
THROUGH   10.0       7.0     2.0      22.0     7.0   9.0
```



Lastly, we have to speak about the TYPE feature, which is an attribute that refers to the kind of architecture or strucutre used for building the final shape of the bridges. Looking at the first picture above, that is the first Histogram, what we can notice is that:

- SIMPLE-T architecture is the most frequent kind of shape or strucutre adopted to build brdiges amongs the pittsburgh bridges, than it is folowed by ARCH-like brideges.
- Howevere, starting from the ARCH-like brideges and going ahead considering the other remaining kind of technique for giving a strucuture to a bridge, what we can undestand is that these attribute are more or less distrubuted equally, instead SIMPLE-T shows the highest value to refer to the number of instances characterized by SIMPLE-T value for this attribute within the datast.

- Furthermore, DECK-like Birdges are just characterized by up to 4 out of 7 possible values for TYPE attribute, while THROUGH-like bridges show examples of istances from all of the possible kind of architectures for building a bridge.

1.1.5 Correlation Matrix Analysis

In fields of statistics as well as statistical learning, where the latter comes partly from the former, *correlation matrix* is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. A correlation matrix is used to summarize data, as an input into a more advanced analysis, and as a diagnostic for advanced analyses. Key decisions to be made when creating a correlation matrix include: choice of correlation statistic, coding of the variables, treatment of missing data, and presentation. Typically, a correlation matrix is square, with the same variables shown in the rows and columns.

Applications of a correlation matrix: there are three broad reasons for computing a correlation matrix:

- To summarize a large amount of data where the goal is to see patterns. In our example above, the observable pattern is that all the variables highly correlate with each other.
- To input into other analyses. For example, people commonly use correlation matrixes as inputs for exploratory factor analysis, confirmatory factor analysis, structural equation models, and linear regression when excluding missing values pairwise.
- As a diagnostic when checking other analyses. For example, with linear regression, a high amount of correlations suggests that the linear regression estimates will be unreliable.

Treatment of missing values: the data that we use to compute correlations often contain missing values. This can either be because we did not collect this data or don't know the responses. Various strategies exist for dealing with missing values when computing correlation matrixes. A best practice is usually to use *multiple imputation*. However, people more commonly use *pairwise missing values* (sometimes known as partial correlations). This involves computing correlation using all the non-missing data for the two variables. Alternatively, some use *listwise deletion*, also known as case-wise deletion, which only uses observations with no missing data. Both pairwise and case-wise deletion assume that data is missing completely at random.

[27]:
```
corr_matrix = dataset.corr()
```

Coding of the variables: if you also have data from a survey, you'll need to decide how to code the data before computing the correlations. Changes in codings tend to have little effect, except when extreme.

Presentation: when presenting a correlation matrix, you'll need to consider various options including:

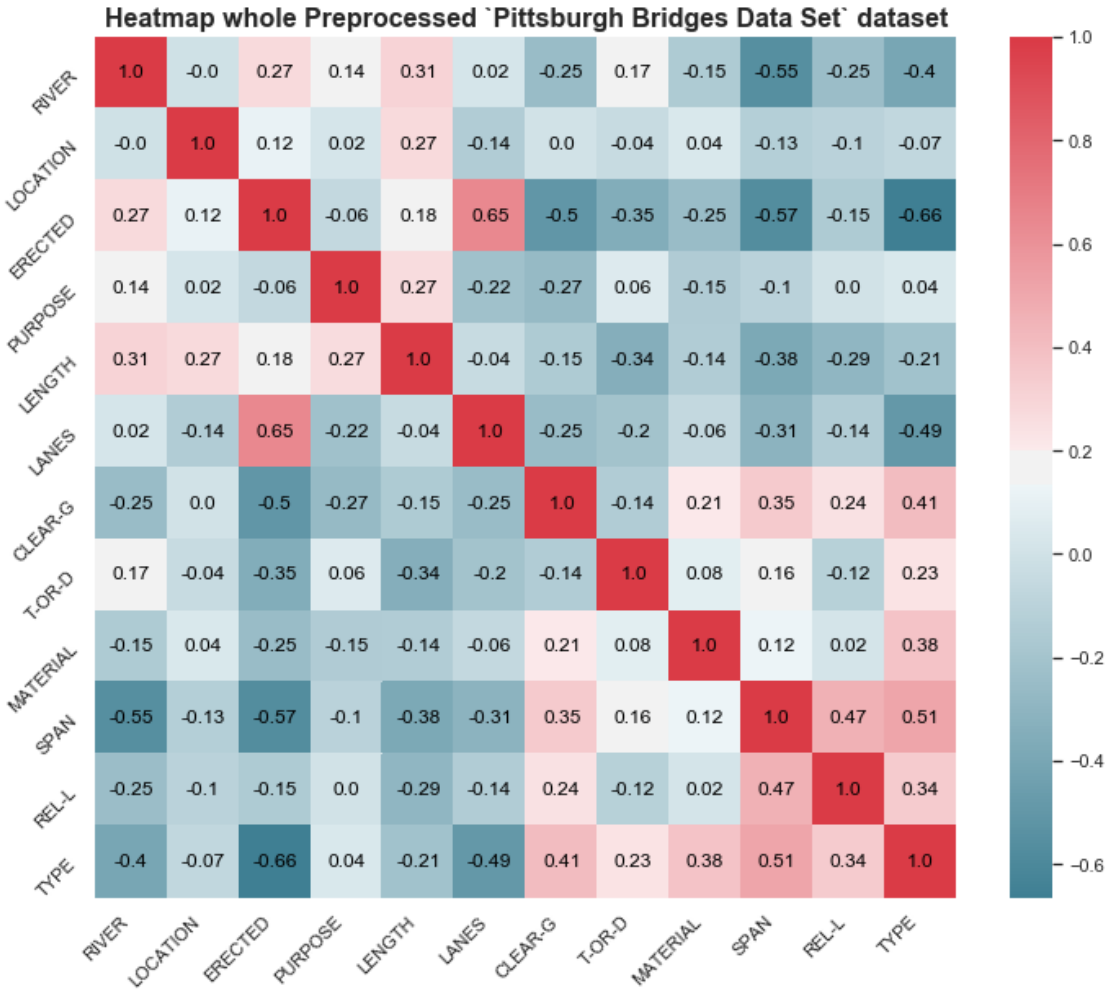
- Whether to show the whole matrix, as above or just the non-redundant bits, as below (arguably the 1.00 values in the main diagonal should also be removed).
- How to format the numbers (for example, best practice is to remove the 0s prior to the decimal places and decimal-align the numbers, as above, but this can be difficult to do in

most software).

- Whether to show statistical significance (e.g., by color-coding cells red).
- Whether to color-code the values according to the correlation statistics (as shown below).
- Rearranging the rows and columns to make patterns clearer.

This shows correlations between the stated importance of various things of attributes used to describe records, examples, and samples within bridge dataset. The line of 1.00s going from the top left to the bottom right is the main diagonal, which shows that each variable always perfectly correlates with itself. This matrix is symmetrical, with the same correlation is shown above the main diagonal being a mirror image of those below the main diagonal.

[28]:
```
display_heatmap(corr_matrix)
```



This kind of presentation allows us, once we choose a row number and column number let's say *ith-row* and *jth-column*, we can check locally the value assigned by the correlection factor to the pair made from to distinct features if *"i"* strctly different from *"j"*. As an instance, what we can suggest observing the correlation matrix depicted just above, as well as exploting some common properties of symmetic and square matrices, is that: - along the area that spread near the main diagonal the resulting features pairs seemes to either positively moderately correlate

or positively waekly correlate. - Convercely, along the area that spreads near the anti-diagonal matrix the resulting features pairs seems to instead either negatively moderately correlate or negatively waekly correlate.

Finally, as examples, in order to exploit the fact that we can access directly to the correlaction valuecomputed by means the math formula provided by the expression of correlation facotr, we can say that: - the pair represented by ERECTED and LANES (3d-row, 6th col) features seems to moderately positively correlate, with a value equals to 0.65. This is also reasoable since that by the time while the Pittsburgh city was growing in size also the need of more infrastructures and building for working and let the people in town to leave, lead to increase in winde the different bridges to manage the traffic from and to the city. - on the other hand, still speaking about ERECTED feature, when it is coupled with TYPE feature (3d-row, 12th col) we see in contrast that them are characterized by a negative value of correlation that lead to interprete the pair as negatively moderately correlated. The principal reason about such a behavior may be imputed by the observation that through the different years the building techniques and technologies have been employed to constrcut better, stronger bridges abandoning oldest techniques which instead imply the exploitation of less technological materials sucha s wood that requires instead more frequently maintenance tasks.

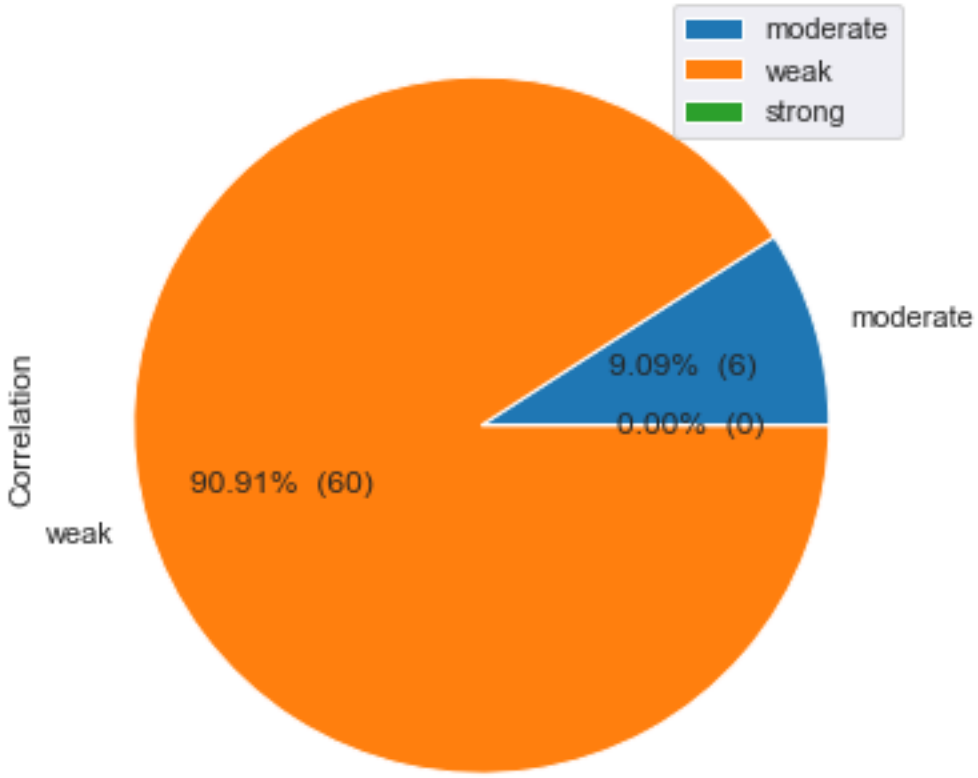
1.1.6 Pie chart as a continuation of Correlation Matrix Analysis

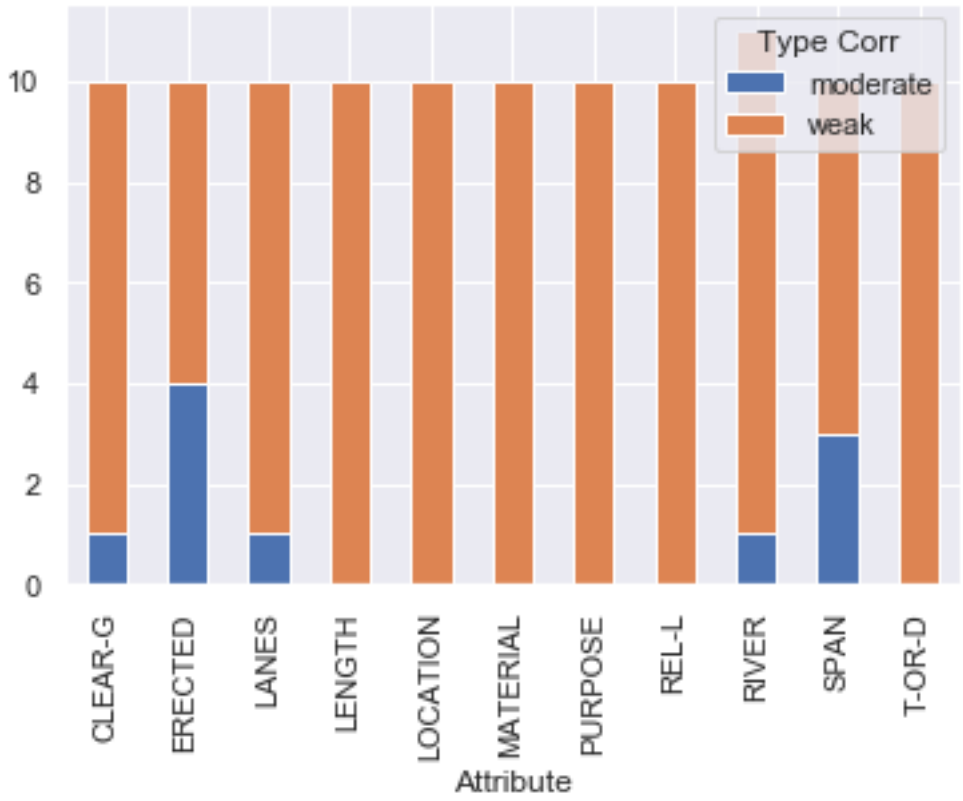
Here, within this subsequent section, I'm going to discuss and analyse the usefulness of exploiting piec-chart like graphs for describing some features or beahvior of correlation matrix values.

The first pie chart followed by also a realted histogram both aim at explaining and depicting how the features pairs are distributed among the three main subintervals that are named weak, moderate and strong as facotrs to label the kind of correlation the value of correlation referrend to a given pair represent, which interval are the following:

- weak, if $p_{ith} \leq .5$
- moderate, if $.5 < p_{ith} < .8$
- strong, if $p_{ith} \geq .8$

[29]:
```
show_pie_hist_charts_abs_corr(corr_matrix, figsize=(2, 2), gridshape=None)
```


moderate
weak
strong
Correlation
moderate
9.09% (6)
0.00% (0)
90.91% (60)
weak



The insight that we can understand is that up to nearly 90.90% of pairs of features is weakly correlated and just 9.09% is moderately correlated, without performing any finer distinction among positively or negatively correlated in each group. While we can notice that no pair is strongly correlated. Moreover, looking at the related histogram, just illustrated below the pie chart, we can clearly understand that just 5 over 12 features are showing also a moderate correlation patterns, which are: CLER-G, ERECTED, LANES; RIVER, and SPAN. Furthermore, we can end up saying that among those 5 features just ERECTED and SPAN shw the larger number of pairs in which them moderately correlate. instead in the majority of cases the possible pairs of features in large measure seem to weakly correlate.

The other two graphs, the first a pie chart and the second a histogram are a kind of zoom in of the other two graphs illustrated in the paragraphs above. In particular those two subsequents graphs aim at exploring in a deeper way the difference about correlation factor, taking into account also the positivness, or convercely the negativness of the kind of correlation, and so not just the information about the strongness in absolute value of the correlation factor.

2 Learning Process

2.1 Learning Process

Here in this section we are going to partly describe and in the remaining to test and evaluate performance of various machine learning models that we selected and adopted to built up learning models for accomplishing supervised machine learning tasks related to classification

problems. More precisely, we focused on binary classification problemsa, since the target variables, that is T-OR-D feature, amongst the 12 features from which the dataset is made of, and from which the more or less hundered of records are described, is a binary categorical feature which can assume the two values represented by labels DECK and THROUGH describing in two distinct manner a property about each bridge within the dataset, that property refers to the system used for constructing the bridge surface, that is commonly called deck, for let veicheles or trains or whatevert to cross the rivers that are three distinct rivers: A, M, O. Where A stands for Allegheny river, while M stands for Monongahela river and lastly O stands for Ohio river.
Before describing the fine tuning process applied to the different models accordingly to their own major properties, characteristics and features, we have decided and established to test each model performance by looking at how well all of them is going just exploiting the default setting and runnng cross-validation protocol, in other word also referred to as policy, to check the accuracy level, as ana instance and some other metrics.
To be more detailed we follow the common machine leaning working flow, that requires to split the data set, after having preprocessed it properly and in a suitable manner to meet machine learning models needs, into subsets commonly referred to as training set and test set. Where, the former is exploited to build up a inference model and the latter is used to check model's performance as well as behavior up on held out sample of istances never seen before and so those examples that the learning model wasn't provided with to learn weights and select right hyper-params to plug in back into model at the end of training procedure, in order to later test the model with the found weights as well as hyper-parameters and if it meet our requirements in terms of performance values reached at test time, then it will be ready for deployments.

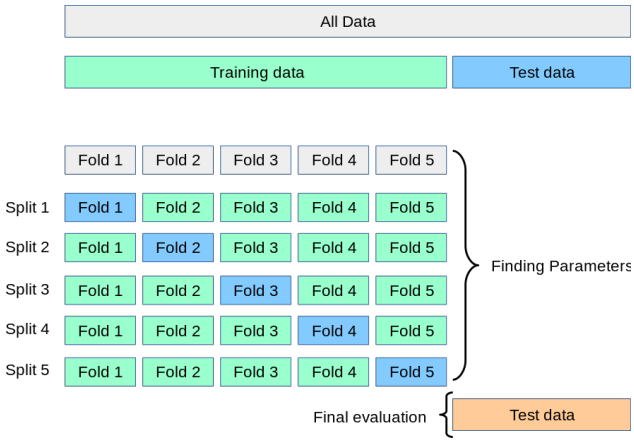
As we can see from the image just above what we have to do, following the purposed machine learning scheme or work flow are the subsequent steps:

1. **Split Dataset into Train and Test sets**: After having done with preprocessing phase we separate or divide the dataset into training set and test set, where usually the training set is bigger than the test set, and in some cases the test set, for recording some kind of performance measures, can be made of just a single instance against which the model will be tested and checked.

2. **Split Train set into smaller Train set and a Validation set**: Once have made the first split then we hold out a part for later checking the test set and we focus on training set. In fact once we have selected a machine learning model amongst those we want to adopt to fit a model and compare the results obtained we try to indentify the best fit parameters to setting the model with them. To reach such a goal we can adopt different aproaches to split further the training set into a validation set and a smaller train set in order to emulate before doing test phase a possible beahvior of the trainng model once we think it is ready for the following phase that is test step. There are several procedure

for establishing how to divide training set into two new subsets, that are validation and a little bit smaller train set, where all of them are also connected or reffered to a learning algorithm called cross-validation which consists roughly speaking into testing the model against a smaller portion of training set in order to record and measure the performance of a model before saying we are ready to proceed with test phase. Among the existing Cross-Validation Procedure we adopt and so describe briefly the following:

- **K-fold Cross-Validation**: which is a cross validation protocol exploited so that we split the training set into K-folds, in other words K-subsets all of the same size, a part eventually the last one which will be the remainder set of samples. One at a time the K-fold are left out and the model is firstly trained against K-1 subsets(folds) and then test against the left out fold for recording the performanc. At the end of the k-times we have trained the model we have recorded the performance measures and we so can average the results and understand how model in average is well doing. In oder words we can either assume the mean value as the driving value to assume the model as satisfying our constraint on performance measures or adopt the best result amongst the k-trains as the settings for hyper-params to adopt. This procedure is feasible and suitable if we do not carea botu the fact that, in cases of classification tasks, the categories might be balanced or not in terms of number of instances, as well as, it can be adopted also if we want to show a learning curve and some other performance graphics or schemes as Confusion matrtices and ROC or recall-precision curves. Latly usual values for K are: 5, 10, 20.



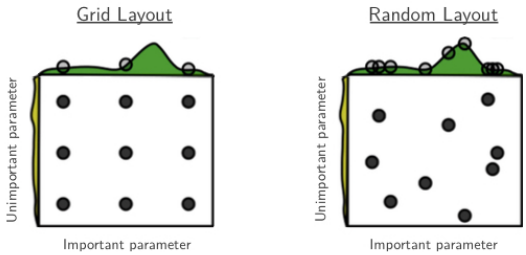
- **Leave One Out Cross-Validation**: It is a special case of the K-fold Cross-Validation. In fact, instead of adopting the usual values as 5, 10, 20, for K as the number of possible subsets, we establish to identify each single instance as a possible fold. It's clear that this algorithm requires more time to be completed and does not allow to show up the graphics cited just above, since we do not enough data for confusion matrix and ROC or recall-precision curves.

- **Stratified Cross-Validation**: It is a good compromise when the dataset is still large enough to be exploited for training purposes but we decide to split into K-folds the traning set such that each subset have the same proportion of samples coming from the differn classes. This operation reveals to be necessary or even mandatory when we detect that the dataset does not show the same number of samples, more or less,

for each class, in other word when the dataset is unbalanced for a given attribute that is the target attribute. This means that, while trying to mitigate the issue about the unbalanced dataset we thunk as well as hope that this management let the model to fit a model which will not be affected heavily just from the most numerous class, bus still learn how to classify the samples coming from the other less numerous classes, without too mcuh misclassifying errors. As usual also with Stratified Cross-Validation we are able to show same graphics as with plain K-fold Cross-Validation, the difference is that the folds are not randomly sampled from the original training set, but yet are sampled in the same proportion per each class in order to have the same number of samples for each class inside each fold.

We try out all of the three described cross-validation techniques to measure how weel default settings for different models are doing to gaina a base line against wich compare later results coming from fine tuning process carried out exploiting grid search technique for selecting the best combinatin of purposed values for each candidate machine learning technique.

3. **Grid Search approach**: is the technique adopted when taking into account one at a time all the machine learning algorithm for selecting the best set of hyper-parameters. It consists into defining for each model a grid of possible values for different hyper-parameters that in some sense represent our degree of freedom referred to some of the properties that characterize all the different models. The grid of values might be real number ranging within a more or less interval, or a string value used to trigger a certain feature of a model combined with other related aspects of the machine learning algorithm of the given model. We recall that the standard grid search will proceed until all possible combination of provided values have been test and training with such settings have been carried out. Opposite to classic grid search it is another technique called Random Grid Search, which implies instead to let a model to choose or sample randomly the hyper-parameters within the ranges or intervals related to each hyper-params. The latter technique can be potentially less expensive since we test a reduced number of combination but might be sub-optimal even if the results can be still acceptable and meaningful.

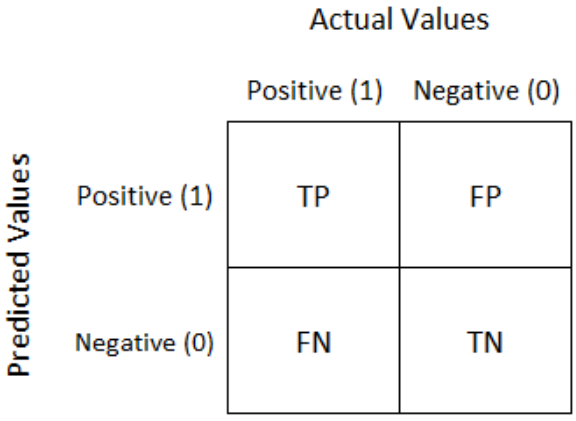


4. **Metrics and model performance's evaluation tools**: before deploying we have to test our model against a test set, that is a subset created or obtained from overall dataset which was, after preprocessing phase that turns feature values somehow into numbers, devided into two distinct sets generally of different size. The test set evaluation implies

exploiting some metrics such as Accuracy, but yet there exist several others that partly are derived from confusion matrix evaluation tool, such as Precsio, Recal F1-score, and the like.

So what we understand is that we can make use of a bouch of metrics but rather than using directly those metrics we can explore model's performance by means of more useful tools such as Confusion Matrix, Roc curve in order to better understanding model's behavior when fed with unobserved new samples, as well as, how to set a threshold for determing when a target variable will suggest us that the classified sample belong to one class or the other. So, here we briefly describe which instruments we exploit to measure model's peformance starting from confusion matrix and moving ahead toward ROC curve.

Confusion Matrix: in statistics a confusion matrix it's a grid or matrix of numbers and in the simplest scenario, correspoding to a binary classification task, it aims at showing how well a model was going when applied onto unknow or preiviously unseen data points and samples, in the following manner. Arbitrary we establish that along the rows we have the fraction of samples that the model has calssified or has assigned to agiven class, and so the rows account for *Predicted valuers*. Vice versa, along the columns we have the total number of samples for each class that all together resemble the so called *Actual Values* as we illustrate in the picture just below:



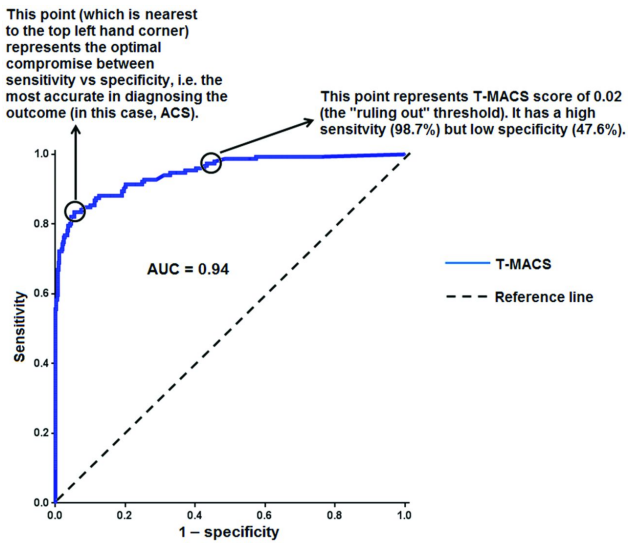
So, such a table of numbers allows us to measure the fraction of correctly classified examples belonging to the Positive class, also referred to as *True Positive(TP)* or to the Negative class, also named *True Negative(TN)* and at the same time we can derive also the fractions of wrongly classified Positive samples and Negative ones, respectively known as *False Positive(FP)* and *False Negative (FN)*. It is clear that looking at the diagonal matrix we can understand that the larger the value along the diagonal the better the model was able to correctly identify the samples accordingly to their own actual class. From the four statistics measure depiceted above, that are TP,TN, FP, and FN, by the time have been dirved other useful metrics that can be exploited when the most

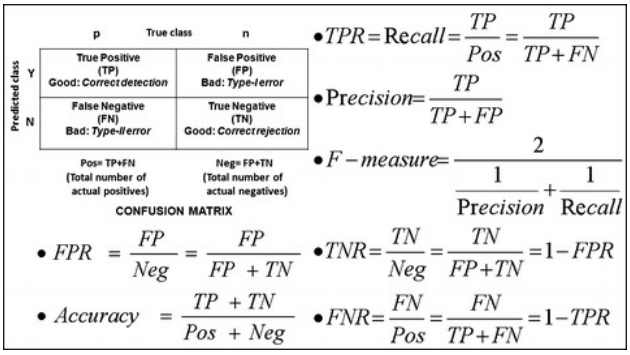
used and well known measurement performance of accuracy is not enough due to the fact as an instance we want to analyze deeper our model behavior toward optimization of some goal or because we have to deal with dataset which are not balanced through classes to be predicted and so we want some other metrics to ensure the goodness of our solutions.

Roc Curve: for the reason cited some lines earlier by exploiting the four basis metrics have been developed other useful tools for ensuring the goodness of a solution, among those tools we decide to adopt the following one, known as Roc Curve. It is a curve, whose acronym stands for Receiver Operating Curve, largely employed in the field of *Decision Theory* and aims at finding, suggesting or showing how model's performance vary when we are going to set different thresholds for a simple scenario in which we are going to solve a binary classification problem. Such a curve require to show on the x-axis the fraction of samples corresponding to *False Positive Rate (FPR)* at a different values of the model's hyper-params corresponding to threshold set for classifying items at inference time, as well as on the x-axis the fraction of samples corresponding to *True Positive Rate(TRP)* in order to plot a curve that originates at coordinates (0,0) and terminates at coordinates (1,1), varying in the middle accordingly to the pairs of values recorded at a given threshold for (FPR,TPR) pair. We are also reporting two driving curves that are respecitvely the curve related to the *Random Classifier* which corresponds to a classifier that for each instance is just randomly selecting the predicted class, and the *Perfetct Classifier* that always correctly classifies all the samples. Our goal by analysing such a graphics is to identify the threshold value such that we are near the points on the curve that are not so far from the upper-left corner so that to maximise the TPR as well as minimizing the FPR. Another useful quantity related to Roc Curve is represented by the so called amount Area Under the Curve (AUC) that suggests us how much area under the roc curve was accounted while varying the threshold for classifying the test samples, in particular the higher the value the better the classifier is doing varying the thresholds. Lastly we notice that the Random Sample accounts for an AUC equals to 0.5 %, while the Perfect Classifier for 1.0 % so we aim at obtaining a value for AUC that is in at least in the between but that approaches the unity. Here, below is presented an example of Roc Curve example:

Lastly, before moving ahead with describing Machine Learning Models we provide a brief list of other useful metrics that can be exploited if necessary during our analyses:

Learning Curve: learning curves constitute a great tool to diagnose bias and variance in any supervised learning algorithm. It comes in handt when we have to face against the so called **Bias-Variance Trade-Off**. In order to explain what that trade-off implies we have to say briefly what follows: - In supervised learning, we assume there's a real relationship between feature(s) and target and estimate this unknown relationship with a model. Provided the assumption is true, there really is a model, which we'll call f, which describes perfectly the relationship between features and target. - In practice, f is almost always completely unknown, and we try to estimate it with a model $\hat{f}$. We

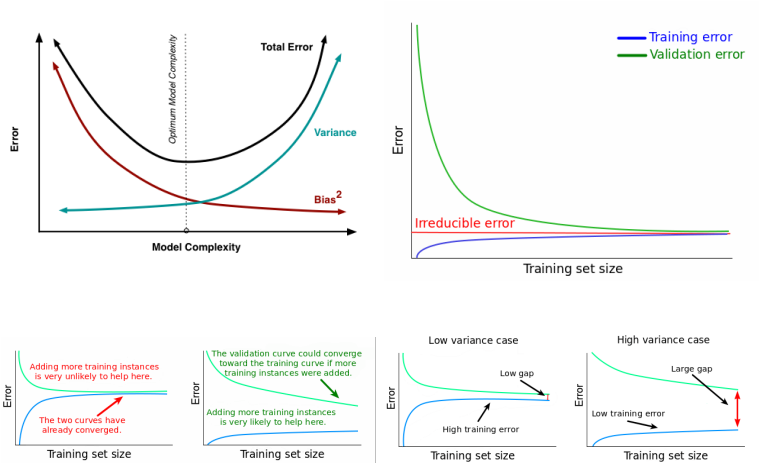




use a certain training set and get a certain $\hat{f}$. If we use a different training set, we are very likely to get a different $\hat{f}$. As we keep changing training sets, we get different sections/learning-process/outputs for $\hat{f}$. The amount by which $\hat{f}$ varies as we change training sets is called **variance**. - While, for most real-life scenarios, however, the true relationship between features and target is complicated and far from linear. Simplifying assumptions give **bias** to a model. The more erroneous the assumptions with respect to the true relationship, the higher the bias, and vice-versa. - In practice, however, we need to accept a trade-off. We can't have both low bias and low variance, so we want to aim for something in the middle, knowing that:

$$\begin{cases} Y = f(X) + \quad irreducible \quad error \\ f(X) = f(\hat{X}) + \quad reducible \quad error \\ Y = f(\hat{X}) + \quad reducible \quad error + \quad irreducible \quad error \end{cases} \tag{1}$$

P-Value Analysis: Here, in the following we are going to describe shortly what is and how we have exploited the statistical tool for assessing model performance from a particular point of view represented by the so called *P-value Analysis*. The p-value is widely used in *statistical hypothesis testing*, specifically in *null hypothesis significance testing*. In this method, as part of experimental design, before performing the experiment, one first chooses a model (the null hypothesis) and a threshold value for p, called



the *significance level* of the test, traditionally *5% or 1%* and denoted as α. If the p-value is less than the chosen significance level (α), that suggests that the observed data is sufficiently inconsistent with the null hypothesis and that the null hypothesis may be rejected. However, that does not prove that the tested hypothesis is true. When the p-value is calculated correctly, this test guarantees that the type I error rate is at most α. For typical analysis, using the standard $\alpha = 0.05$ cutoff, the null hypothesis is rejected when $\rho < .05$ and not rejected when $\rho > .05$. The p-value does not, in itself, support reasoning about the probabilities of hypotheses but is only a tool for deciding whether to reject the null hypothesis.

In our particular case:

- Null hypothesis (H_0): we ask ourselve whether a given particular model is able to predict class labels, with performance that matches just a uniform model's performance with $Prob(THROUGH) = 0.5$
- Test statistic: shuffleing target labels and measuring accuracy score to determine significance level
- Alpha level (α, designated threshold of significance): 0.05

5. **Deployment**: The last step for building a machine learning system comprise the deployment of one or more machine learning trained models that fit and satisfy constraints that was intially fixed before doing any kind of analysis. The main goal of deployemnt is to employ such statistics models for predicting, in other words making inference, against new data, unknown obeservatons for which we do not knwo their target class values.

2.1.1 Pricipal Component Analysis

After having investigate the data points inside the dataset, I move one to another section of my report where I decide to explore examples that made up the entire dataset using a particular technique in the field of statistical analysis that corresponds, precisely, to so called Principal Component Analysis. In fact, the major objective of this section is understand whether it is possible to transform, by means of some kind of linear transformation given by a mathematical calculation, the original data examples into reprojected representation that allows me to retrieve most useful information to be later exploited at training time. So, lets dive a bit whitin what is and which are main concepts, pros and cons about Principal Component Analysis.
Firstly, we know that **Principal Component Analysis**, more shortly PCA, is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called *principal components*. This transformation is defined in such a way that:

1. the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), - and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components.
   The resulting vectors, each being a linear combination of the variables and containing n observations, are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables.
   PCA is mostly used as a tool in *exploratory data analysis* and for making predictive models, for that reasons I used such a technique here, before going through the different learning technique for producing my models.

Several Different Implementation : From the theory and the filed of research in statistics, we know that out there, there are several different implementation and way of computing principal component analysis, and each adopted technique has different performance as well as numerical stability. The three major derivations are:

1. PCA by means of an iterative based procedure of extraing pricipal components one after the other selecting each time the one that account for the most of variance along its own axis, within the remainig subspace to be derived.
2. The second possible way of performing PCA is done via calculation of *Covariance Matrix* applied to attributes, that are our independent predictive variables, used to represent data points.
3. Lastly, it is used the technique known as *Singular Valued Decomposition* applied to the overall data points within our dataset.

Reading scikit-learn documentation, I discovered that PCA's derivation uses the *LAPACK*

implementation of the *full SVD* or a *randomized truncated SVD* by the method of *Halko et al. 2009*, depending on the shape of the input data and the number of components to extract. Therefore I will descrive mainly that way of deriving the method with respect to the others that, instead, will be described more briefly and roughly.

PCA's Iterative based Method : Going in order, as depicted briefly above, I start describing PCA obtained by means of iterative based procedure to extract one at a time a new principal componet explointing the data points at hand.
We begin, recalling that, PCA is defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some scalar projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.
We suppose to deal with a data matrix X, with column-wise zero empirical mean, where each of the n rows represents a different repetition of the experiment, and each of the p columns gives a particular kind of feature.

From a math poitn of view, the transformation is defined by a set of p-dimensional vectors of weights or coefficients $\mathbf{w}_{(k)} = (w_1, \dots, w_p)_{(k)}$ that map each row vector $\mathbf{x}_{(i)}$ of X to a new vector of principal component scores $\mathbf{t}_{(i)} = (t_1, \dots, t_l)_{(i)}$, given by: $t_{k(i)} = \mathbf{x}_{(i)} \cdot \mathbf{w}_{(k)} \qquad \text{for} \qquad i = 1, \dots, n \qquad k = 1, \dots, l.$

In this way all the individual variables $t_1, \dots, t_l$ of t considered over the data set successively inherit the maximum possible variance from X, with each coefficient vector w constrained to be a unit vector.

More precisely, the first component In order to maximize variance has to satisfy the following expression: $\mathbf{w}_{(1)} = \arg\max_{\|\mathbf{w}\|=1} \left\{ \sum_i (t_1)^2_{(i)} \right\} = \arg\max_{\|\mathbf{w}\|=1} \left\{ \sum_i \left(\mathbf{x}_{(i)} \cdot \mathbf{w} \right)^2 \right\}.$

So, with (w_1) found, the first principal component of a data vector (x_1) can then be given as a score $(t_{1(i)} = x_1 \cdot w_1)$ in the transformed co-ordinates, or as the corresponding vector in the original variables, $((x_1 \cdot w_1) w_1)$.

The others remaining components are computed as follows. The k_{th} component can be found by subtracting the first $k-1$ principal components from X, as in the following expression:

- $\hat{\mathbf{X}}_k = \mathbf{X} - \sum_{s=1}^{k-1} \mathbf{X} \mathbf{w}_{(s)} \mathbf{w}_{(s)}^{\mathrm{T}}$
- and then finding the weight vector which extracts the maximum variance from this new data matrix $\mathbf{w}_{(k)} = \arg\max_{\|\mathbf{w}\|=1} \left\{ \|\hat{\mathbf{X}}_k \mathbf{w}\|^2 \right\} = \arg\max \left\{ \frac{\mathbf{w}^T \hat{\mathbf{X}}_k^T \hat{\mathbf{X}}_k \mathbf{w}}{\mathbf{w}^T \mathbf{w}} \right\}$

It turns out that: - from the formulas depicted above me get the remaining eigenvectors of $X^T X$, with the maximum values for the quantity in brackets given by their corresponding eigenvalues. Thus the weight vectors are eigenvectors of $X^T X$. - The kth principal component of a data vector $x_(i)$ can therefore be given as a score $t_{k(i)} = x_{(i)} \cdot w_(k)$ in the transformed co-ordinates, or as the corresponding vector in the space of the original variables, $(x_{(i)} \cdot w_{(k)}) w_{(k)})$, where $w_{(k)}$ is the kth eigenvector of $X^T X$. - The full principal components decomposition of X can therefore be given as: $\mathbf{T} = \mathbf{XW}$, where W is a p-by-p matrix of weights whose columns are the eigenvectors of $X^T X$.

Covariance Matrix for PCA analysis : PCA made from covarian matrix computation requires the calculation of sample covariance matrix of the dataset as follows: $\mathbf{Q} \propto \mathbf{X}^T\mathbf{X} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^T$.

The empirical covariance matrix between the principal components becomes $\mathbf{W}^T\mathbf{Q}\mathbf{W} \propto \mathbf{W}^T\mathbf{W}\,\mathbf{\Lambda}\,\mathbf{W}^T\mathbf{W} = \mathbf{\Lambda}$.

Singular Value Decomposition for PCA analysis : Finally, the principal components transformation can also be associated with another matrix factorization, the singular value decomposition (SVD) of X, $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{W}^T$,where more precisely:

1. Σ is an n-by-p rectangular diagonal matrix of positive numbers $(\sigma_{(k)})$, called the singular values of X; - instead U is an n-by-n matrix, the columns of which are orthogonal unit vectors of length n called the left singular vectors of X;
2. Then, W is a p-by-p whose columns are orthogonal unit vectors of length p and called the right singular vectors of X.

Factorizingn the matrix $X^T X$, it can be written as:

$$\begin{aligned}\mathbf{X}^T\mathbf{X} &= \mathbf{W}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{W}^T \\ &= \mathbf{W}\mathbf{\Sigma}^T\mathbf{\Sigma}\mathbf{W}^T \quad . \\ &= \mathbf{W}\hat{\mathbf{\Sigma}}^2\mathbf{W}^T\end{aligned}$$

Where we recall that $\hat{\mathbf{\Sigma}}$ is the square diagonal matrix with the singular values of X and the excess zeros chopped off that satisfies $\hat{\mathbf{\Sigma}}^2 = \mathbf{\Sigma}^T\mathbf{\Sigma}\hat{\mathbf{\Sigma}}^2 = \mathbf{\Sigma}^T\mathbf{\Sigma}$.
Comparison with the eigenvector factorization of $X^T X$ establishes that the right singular vectors W of X are equivalent to the eigenvectors of $X^T X$, while the singular values $\sigma_{(k)}$ of X are equal to the square-root of the eigenvalues $\lambda_{(k)}$ of $X^T X$.
At this point we understand that using the singular value decomposition the score matrix T

$$\begin{aligned}\mathbf{T} &= \mathbf{XW} \\ &= \mathbf{U}\mathbf{\Sigma}\mathbf{W}^T\mathbf{W} \\ &= \mathbf{U}\mathbf{\Sigma}\end{aligned}$$

can be written as: $= \mathbf{U}\mathbf{\Sigma}\mathbf{W}^T\mathbf{W}$ so each column of T is given by one of the left singular vectors of X multiplied by the corresponding singular value. This form is also the polar decomposition of T.

Efficient algorithms exist to calculate the SVD, as in scikit-learn package, of X without having to form the matrix $X^T X$, so computing the SVD is now the standard way to calculate a principal components analysis from a data matrix

[1]:
```
from utils.all_imports import *;
%matplotlib inline
# Set seed for notebook repeatability
np.random.seed(0)
```

```
None
```

[2]:
```
#␣
  ↪============================================================================␣
  ↪#
# 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)
columns_2_avoid = ['ERECTED', 'LENGTH', 'LOCATION']
# 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))
```

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

[4]:
```
# sns.pairplot(dataset, hue=TARGET_COL, height=2.5);
```

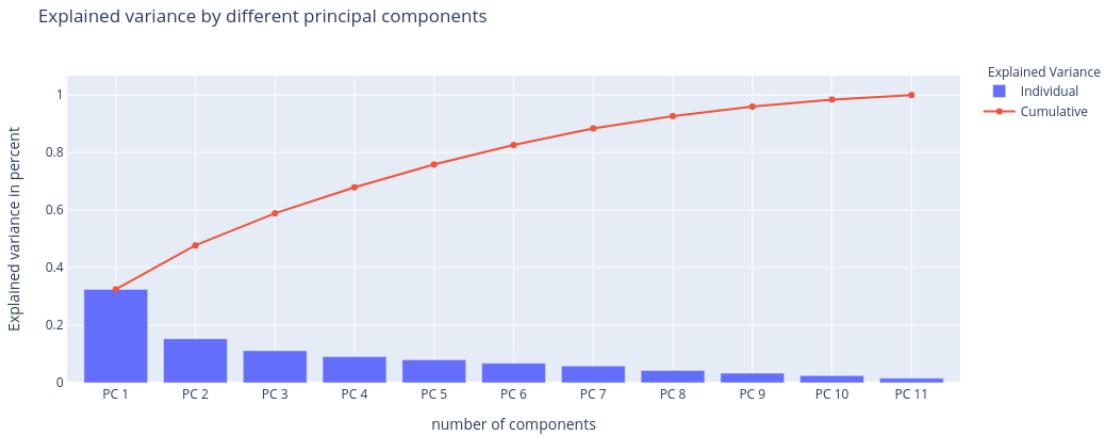
[5]:
```
df = show_table_pc_analysis(X=rescaledX); df.
  ↪to_csv('cumulative_varation_explained.csv') # df
```

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

[6]:

	# PCS	CumVarExp(%)	Linear CumVarExp(%)	Poly CumVarExp(%)	\
0	2.0	47.738342	47.738342	42.530458	
1	5.0	75.856460	75.856460	74.129318	
2	6.0	82.615768	82.615768	79.952749	
3	7.0	88.413903	88.413903	85.492450	
4	8.0	92.661938	92.661938	89.550432	
5	**9.0**	**95.976841**	**95.976841**	**93.443925**	
6	10.0	98.432807	98.432807	97.130597	

	Rbf CumVarExp(%)	Sigmoid CumVarExp(%)	Cosine CumVarExp(%)
0	34.158927	53.962444	41.290635
1	67.573023	81.158209	74.568239
2	74.971362	86.493958	80.721521
3	81.003755	91.341605	86.663670
4	86.846031	95.680993	91.703753
5	**92.367208**	**97.514014**	**95.705986**
6	96.519913	99.030672	98.176194



Major Pros & Cons of PCA The advantages of exploiting a Pca unsupervised technique for preprocessing input data to machine learning supervised technique for building models and predictors are the following:

- It allows for remapping input data expressed by means of a soruce feature space into another destination feature space where axes are arranged so that them are uniform in length and are orthogonal each others, so that them are indeed orthonormal.
- It allows for arranbging destination feature space axes so that them are ordered or sorted

in a manner for which the firts axis account for most of the variability or variance in the re-mapped data examples.

- The previous two observation allow as to understand that Pca Analysis enable to exploit reduced data examples in the number of features, just collecting those features that are the most informative, so we can reduce the effect of noise that wastes the training performance.

Some backwards, instead are the following:

- It necessitates to process the overall dataset, so if that dataset is very huge it can take a lot of time, becoming time-consuming and even unfeasible
- It may be not the best solution for pre-processing data examples if the employed *linear affine transformations* are not enough to capture all the variability and variance in the remapped data.

2.1.2 Kernel-PCa Based on different Kernel Tricks

Possible alternative when standard Pca is not enough might be employing a derived kind of pre-processing unsupervised technique called *kernel-Pca*, which is a statistical learning technique that aims at re-mapping input data examples by means of a provided kernel-trick amongst those available such as, *Polynomial, Rbf, Cosine, and Sigmoid*, with the following properties:

- ***Polynomial***: The polynomial kernel represents the similarity between two vectors. Conceptually, the polynomial kernels considers not only the similarity between vectors under the same dimension, but also across dimensions. When used in machine learning algorithms, this allows to account for feature interaction.
  Kernel-Polynomial expression is given by:
  $k(x,y) = (\gamma x^\top y + c_0)^d$

- ***Sigmoid***: The sigmoid kernel is also known as hyperbolic tangent, or Multilayer Perceptron (because, in the neural network field, it is often used as neuron activation function). It is defined as:
  $k(x,y) = \tanh(\gamma x^\top y + c_0)$

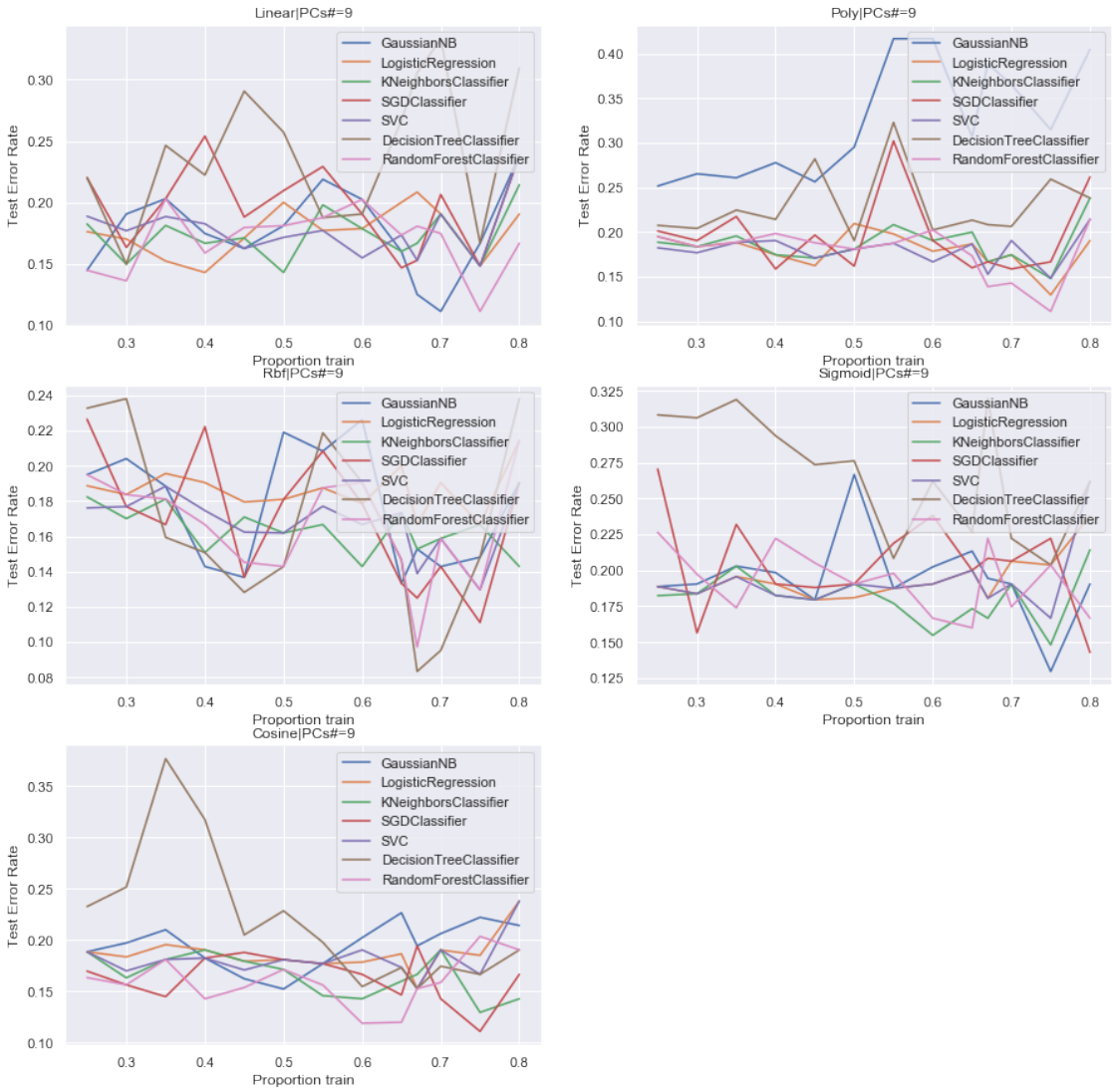
- ***RBF***: The radial basis function (RBF) kernel between two vectors is defined as:
  $k(x,y) = \exp(-\gamma \|x-y\|^2)$, where $\gamma = \sigma^{-2}$ is the kernel is known as the Gaussian kernel of variance σ^2

- ***Cosine***: The Cosine similarity computes the L2-normalized dot product of vectors. This is called cosine similarity, because Euclidean (L2) normalization projects the vectors onto the unit sphere, and their dot product is then the cosine of the angle between the points denoted by the vectors. And it is given by:
  $k(x,y) = \frac{xy^\top}{\|x\|\|y\|}$

2.1.3 Test Error Rate versus Training Sample Size

Here, with the following set of graphics we aim at describing how the *Test Error Rate* changes across different *Training Set Size*, in particular running the tests simply emploing the different estimators or classification algorithm with their default setting and running the training phase via cross-validation technique and adopint increasing training set size. Main goal with the illustration of such a bounch of graphics is to grasph a basic idea, that might be used to lead the training phase about how much the training set size affects the performance of a classifier at least with default settings of the hyper-parameter of the latter.

[6]:
```
try_comparing_various_online_solvers_by_kernel(X=rescaledX, y=y,␣
↪kernels=None, heldout=[.75, .7, .65, .6, .55, .5, .45, .4, .35, .33, .3, .
↪25, .2], rounds=20, n_classes=-1, n_components=9,␣
↪estimators=estimators_list, cv=StratifiedKFold(2), verbose=0,␣
↪show_fig=True, save_fig=False, stratified_flag=True, n_splits=3,␣
↪random_state=42, gridshape=(3, 2), figsize=(15, 15), title="try comparing␣
↪various online solvers by kernel",␣
↪fig_name="try_comparing_various_online_solvers_by_kernel.png")
```



The main results we can get from the analysis we can carry out looking at the graphics just reported above is what follows:

- the *GuassianNB Classifier* in four out of five trials, that are more precisely trial for which we have employed kernel-tricks such as Linear, Polynomial, Rbf, Sigmoid the resulting estimators follow a trend is characterized by increasing and decrising in test error rate with some large picks after we have used at least half of the samples from the dataset. Furthermore, the Polynomial based Estimator results to be the worst among those cosidered earlier, in fact near the half of the training set size the trend even becomes increasing going apart from the decreasing trend characterized from alternating local maxima and minima. Finally, when observing how such a classifier behaves when trained adopting Cosine trick we can refer that it follows a trend somewhat similar to the one followed also from other classifier expect for decision trees classifier.
- the estimators based of *Decision Trees Classifier*, together with also *GuassianNB Classifiers*, are the two techniques amongst the others that were importantly affected by the training set size chosen for performing such analysis. In fact, even if in general is

characterized from a decreasing trend, such a trend seems to be less stable than the others observed for the remaning classification techniques, in fact the trend is highly fluctuating with large picks when crossing and going towards larger training set sizes. The worst cases were when the classification technique was performed after having pre-preprocessed data examples by means of kernel-Pca based on Linear and Rbf tricks, because *Test Error Rate* turned to increasing after a given training set size which is larger than half of samples. While if we compare the generalization represented by Random Forest classifiers with respect to Decision Trees Classifier, it seems that the former across the different trials adopint also different kernel-trick for pre-processing the training examples follows a more stable decreasing trend for Test Error Rate versus training set size graphics, where just in two cases out of five which are trials where we have exploited Rbf and Sigmoid kernel-Pca tricks the random forest models seems to follwo in the second half of the Test Erro Rate curve a more fluctuating trend.

- while, speaking about all the other learning algorithms, what we have noticed is that in general the several training phase executed for the models varying the training set size follows a decreasing trend, which means either that allowing the estimatores to be trained by means of larger training set they gain more insights about how can be characterized training examples, in order to better discriminate between the two classes, or that models overfit to much against the training set. However, all the models seem to start at the very beginning with a Test Error Rate somewhat yet not too large and keep reducing it across the different training set size. More precisely, we have reported that Knn Classifier performs better across the different attempts when the training set size was in the range between half and *70%* of the training set size. While SVMs, Random-Forests, and Sgd classifiers seem to reach better performance when training examples in the training set reaches let to use a traning set size in the range between *70% and 80%* of the training set size. Lastly the logisti regression classifier seems to follow a decreasing trend for Test Error Rate which results to be the most conservative, with a reduced number of fluctuation which also have been a contained changes or fluctuations.

2.1.4 References

- Metrics and Diagnositc Tools:
  - (**Confusion Matrix**)
    https://en.wikipedia.org/wiki/Confusion_matrix
  - (**F-1-Sccore, Accuracy, and Precision-Recall**)
    https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c
  - (**Roc Curve**)
    https://en.wikipedia.org/wiki/Receiver_operating_characteristic
  - (**P-value**) https://en.wikipedia.org/wiki/P-value
- Statistics:
  - (**Correlation and dependence**)
    https://en.wikipedia.org/wiki/Correlation_and_dependence

3 Naive Bayes Classification

3.1 Naive Bayes Properties

Learning Technique	Type of Learner	Type of Learning	Classification	Regression	Clustering	Outlier Detection
Naive Bayes	*Generative Model*	*Supervised Learning*	*Supported*	*Supported*	*Not-Supported*	*Not-Supported*

Naive Bayes models are a group of extremely fast and simple classification algorithms that are often suitable for very high-dimensional datasets. Because they are so fast and have so few tunable parameters, they end up being very useful as a quick-and-dirty baseline for a classification problem. Here I will provide an intuitive and brief explanation of how naive Bayes classifiers work, followed by its exploitation onto my datasets.
I start saying that Naive Bayes classifiers are built on Bayesian classification methods. These rely on Bayes's theorem, which is an equation describing the relationship of conditional probabilities of statistical quantities.
n Bayesian classification, we're interested in finding the probability of a label given some observed features, which we can write as P(L | features). Bayes's theorem tells us how to express this in terms of quantities we can compute more directly:
$P(L|features) = \frac{P(L)\cdot P(features|L)}{P(features)} = \frac{Prior\cdot Likelihood}{Evidence}$
If we are trying to decide between two labels, and we call them L1 and L2, then one way to make this decision is to compute the ratio of the posterior probabilities for each label:
$\frac{P(L1|features)}{P(L2|features)} = \frac{P(features|L1)P(L1)}{P(features|L2)P(L2)}$
All we need now is some model by which we can compute P(features | L_i) for each label. Such a model is called a generative model because it specifies the hypothetical random process that generates the data. Specifying this generative model for each label is the main piece of the training of such a Bayesian classifier. The general version of such a training step is a very difficult task, but we can make it simpler through the use of some simplifying assumptions about the form of this model.
This is where the "naive" in "naive Bayes" comes in: if we make very naive assumptions about the generative model for each label, we can find a rough approximation of the generative model for each class, and then proceed with the Bayesian classification. Different types of naive Bayes classifiers rest on different naive assumptions about the data, and we will examine a few of these in the following sections.
A possible derivation it is given as follows:
$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1,\ldots,x_n|y)}{P(x_1,\ldots,x_n)}$,
using the naive conditional independence assumption that
$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y)$,
for all i, this relationship is simplified to
$P(y \mid x_1, \ldots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1,\ldots,x_n)}$
Since $P(x_1, \ldots, x_n)$ is constant given the input, we can use the following classification rule:

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

,
and we can use Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i \mid y)$; the former is then the relative frequency of class y in the training set.

Gaussian Naive Bayes Perhaps the easiest naive Bayes classifier to understand is Gaussian naive Bayes. In this classifier, the assumption is that data from each label is drawn from a simple Gaussian distribution. In fact, one extremely fast way to create a simple model is to assume that the data is described by a Gaussian distribution with no covariance between dimensions. This model can be fit by simply finding the mean and standard deviation of the points within each label, which is all you need to define such a distribution.
$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$
The parameters σ_y and μ_y are estimated usually using maximum likelihood.

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

# Set seed for notebook repeatability
np.random.seed(0)
```

```
None
```

[2]:
```
#
  ↪============================================================================
  ↪#
# READ INPUT DATASET
#
  ↪============================================================================
  ↪#
dataset_path, dataset_name, column_names, TARGET_COL = get_dataset_location()
estimators_list, estimators_names = get_estimators()
```

[3]:
```
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)
```

3.2 Training Results

[6]:
```
# 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_gauss_naive_bias = {
    'var_smoothing': (1e-9, 1e-6, 1e-3),
    # 'random_state': (0,),
}; param_grids.append(parmas_gauss_naive_bias)

# 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]
```

[7]:
```
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)
```

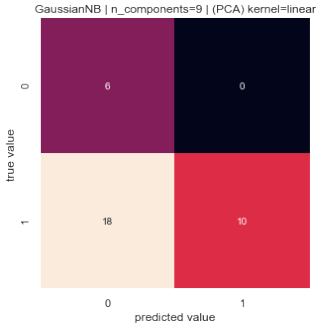
```
)
```

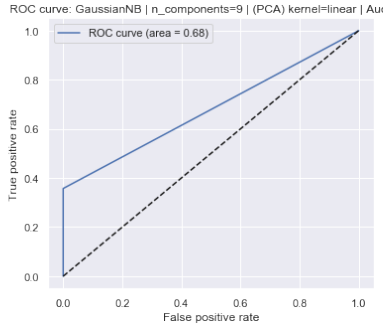
[8]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

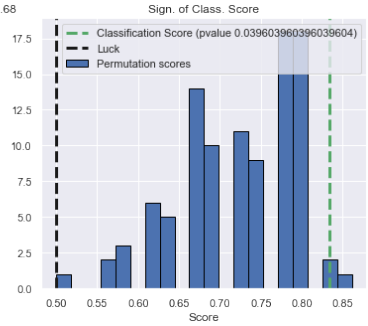
```
<IPython.core.display.Javascript object>
```

[9]:
```
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[0], \
    param_grids=param_grids[0],
    estimators_names=estimators_names[0], \
    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 | GaussianNB
=======================================================
              precision    recall  f1-score   support

     class 0       0.25      1.00      0.40         6
     class 1       1.00      0.36      0.53        28

    accuracy                           0.47        34
   macro avg       0.62      0.68      0.46        34
weighted avg       0.87      0.47      0.50        34

Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.86              0.47  0.68  0.03960
```

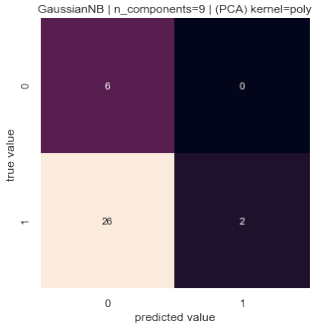


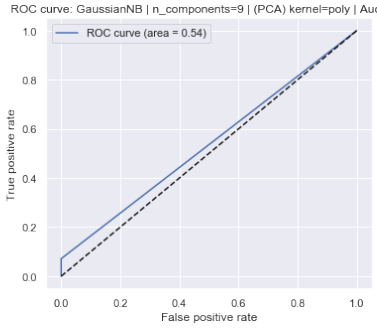


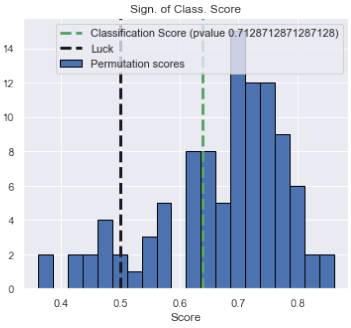



```
Kernel PCA: Poly | GaussianNB
```

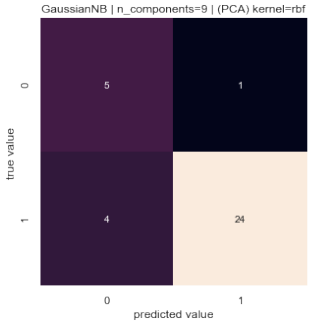
```
=====================================================
              precision    recall  f1-score   support

     class 0       0.19      1.00      0.32         6
     class 1       1.00      0.07      0.13        28

    accuracy                           0.24        34
   macro avg       0.59      0.54      0.22        34
weighted avg       0.86      0.24      0.17        34

Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.72                0.24  0.54  0.71287
```

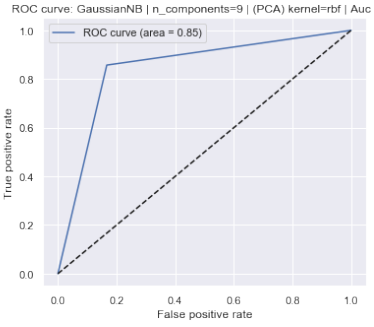







```
Kernel PCA: Rbf | GaussianNB
=====================================================
              precision    recall  f1-score   support

     class 0       0.56      0.83      0.67         6
     class 1       0.96      0.86      0.91        28

    accuracy                           0.85        34
   macro avg       0.76      0.85      0.79        34
weighted avg       0.89      0.85      0.86        34

Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.84                0.85  0.85  0.34653
```

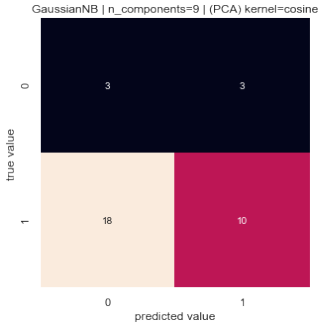


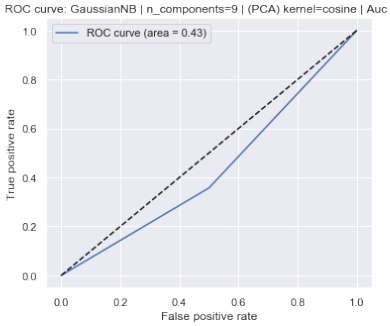


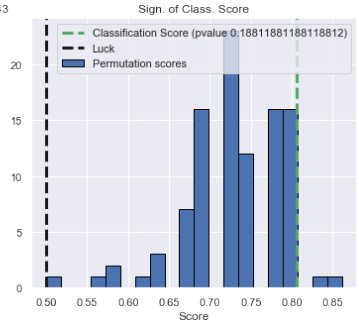
Kernel PCA: Cosine | GaussianNB

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

     class 0       0.14      0.50      0.22         6
     class 1       0.77      0.36      0.49        28

    accuracy                           0.38        34
   macro avg       0.46      0.43      0.36        34
weighted avg       0.66      0.38      0.44        34
```

Best Score (CV-Train)	Best Score (Test)	AUC	P-value
0.92	0.38	0.43	0.18812



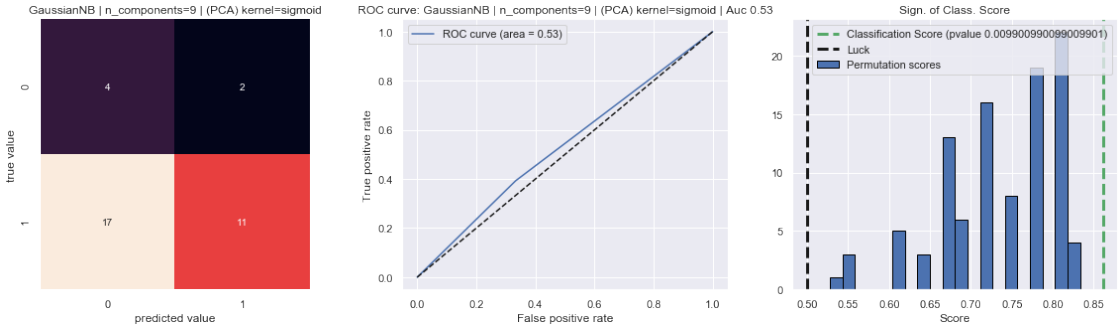




Kernel PCA: Sigmoid | GaussianNB

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

     class 0       0.19      0.67      0.30         6
     class 1       0.85      0.39      0.54        28

    accuracy                           0.44        34
   macro avg       0.52      0.53      0.42        34
weighted avg       0.73      0.44      0.49        34
```

Best Score (CV-Train)	Best Score (Test)	AUC	P-value
0.92	0.44	0.53	0.00990



1. Speaking about **Linear kernel Pca based Naive Bayes Classifier**, we can notice that such a model exploiting a default threshold of *.5* reaches a poor test accuracy score of just *47%* with respect to a train accuracy score of *86%*, while the Auc score reaches a lower value of just *68%*. Since the model has obtained a even lower than Random Classifier referencing model it might be due to an overfit situation detected at train time, also maybe because the dataset is small and small dataset tend to overfit easier. The model trained by means of pre-preprocessed data with linear-trick for kernel-Pca method is highly precise and has a high recall for class 1 and class 0 respectively. However, it results at the same time very low precise when predicting a new instance as belonging to class 0, furthermore it wasn't able to correctly detect the majority of samples from class 1, in fact most of them were mispredicted. Also the Roc Curve shows a trend for which just whit a small set of threshold the model increases the TPR while the FPR remains still zero, but then at a given threshold the trend becomes linear with a slope lower than that of Random Classifier Roc Curve so that the model turns to quicklier grows up the FPR than the TPR.
2. Observing **Polynomial kernel Pca based Naive Bayes Classifier**, we can notice that such a model exploiting a default threshold of *.5* reaches even a lower test accuracy of *24%* than a still high train accuracy of *24%*, instead Auc Score has reached a value of *54%*. The model that raises up when fine-tuning against a dataset prepared by means of polynomial trick for kernel-Pca method has a performance tha is slightly better than the Random Classifier as we can notice also by means of Roc Curve trend, however the model since has obtained very low test accuracy score we can imagine that overfit at train time against the training set and in particular it gotten a configuration for which tends to misclassify the majoirity of example that indeed belong to class 1, so the model has very low recall and low precision for class 1 and class 0, respectively, in contrast, has high recall referred to class 0, which means that such a classifier is able to recognize most of class 0 even if when predicting class 0 label it is generally estimated with high uncertainty. Both this model and the previus one should be discarded, in other words, avoided for later using at inference time, for prediciting class label for new unseen data examples.
3. Review **Rbf kernel Pca based Logisti Classifier**, obtained higher accuracy on test set of *.85* and high train accuracy of *.84*, that allow for a Auc score that reaches a value of *.85*. The model even goes better at test time than at train time, however, we can notice that it reaches high precision and recall for class 1, so such a configuration where data

examples have been pre-processed using Rbf-trick for kernel-Pca leads to an estimator that is able to highly recognize examples from class 1. On the other hand, due to the fact that the model is trained against a unbalanced dataset, even if we have been to correctly predict the class labels for most of instances belonging to class 0, we kept to mispredict a number of examples from the opposite class that lead to a low precision for class 0, so when a label 0 is predicted it is done with high uncertainty. While looking at Roc Curve, we can clearly understand that a default threshold of 0.5 has a discriminative cutt-off is sufficient, even more, the model is characterized from a roc curve where in the first part the relationship between the TPR and FPR grow ogheter linearly but with a high slope and only when reached a TPR of more or less *80%*, the trend keeps to be linear but whit a slope lower than that of random classifier, which means the model let FPR now to grows quicklier.

4. **Cosine kernel Pca based Logisti Classifier** results to be the worst solution found while performing grid search algorithm for Logisti Regression method, when it is fixed a defualt threshold of *.5* for classification. The rationale is that this trial retrieves a model that does overfit to the train set, since test set accuracy is *38%*, instead train accuracy score reaches a value of *92%*, while Auc score is even lower than that of random classifier, more precisle of just *43%*. In particular the model misclassifyes the majority of class 1 instances at test time, and also mispredicts class labels for half of samples from class 0. So it is characterized from low recall for both classes, and on one hand low precision for class 0 but on the other hand a high precision for class 1. Summarizing, such a trial, when pre-processing the data examples by means of cosine-trick for kernel-Pca learing leads to an estimator that tends to predict the majority of samples as belonging to class 0, while still precise when predicting as class 1 the different samples from class 1. Whe should avoid and discard such a model since do not reaches important performance.
5. Lastly, also **Sigmoid kernel Pca based Logisti Classifier** does not perfome very well, in fact, obtained accuracy on test set of *.44* and train accuracy of *.92*, that allow for a Auc score that reaches a value of *.53*. The predictor learned by means of training examples preprocessed exploiting Sigmoid-trick performs slightly bettern than random classifier when speaking about Roc Curve trend together with Auc score, while looking at other performance measurements we can clearly notice that the estimator tends to misclassify the most examples from class 1, obtaining so low precision when predicting a class label equal to class 0, while seems to be very preices and so predict with low uncertainty when the resulting class label is class 1. Also such a model is not really promising since does not obtain important performance scores to let it be exploited later for inference tasks.

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. Looking strictly to p-values scores, we can notice that three out of five models got a p-value scores that are widely larger than the significance level, which are *Poly, Rbf, and Cosine kernel Pca based Naive Bayes Classifiers*. However, even if the remaining classifiers represented by *Linear, and Sigmoid kernel Pca based Bayes Classifiers* obtained p-value scores that are acceptable since lower than fixed a priori significance level, only linear-trick-based model can be considere meaningful since the other leads to too much

poor perfomance. We conclude saying that if we do not reject Null-Hypothesis for *Poly, Rbf, and Cosine kernel Pca based Naive Bayes Classifiers* we should fall into *Type II error*, on the other hand, if we reject Null-Hypothesis for *Linear, and Sigmoid kernel Pca based Bayes Classifiers* we fall into *Type I error*, lastly amongst the two that we should keep the former can be used alone or even with a ensemble technique, while the latter just for the ensemble technique.

Table Fine Tuned Hyper-Params (Logisti Regression)

[10]:
```
# 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)
```

[10]:

	AUC(%)	P-Value(%)	Acc Train(%)	Acc Test(%)	var_smoothing
GaussianNB linear	0.68	3.96	0.86	0.47	1e-09
GaussianNB poly	0.54	71.29	0.72	0.24	1e-09
GaussianNB rbf	**0.85**	**34.65**	**0.84**	**0.85***	**1e-09**
GaussianNB cosine	0.43	18.81	0.92	0.38	1e-09
GaussianNB sigmoid	0.53	0.99	0.92	0.44	1e-09

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 *Linear, Polynomial, Cosine, and Sigmoid Tricks* or less overfit solution such as in the case of *Rbf Trick*. Speaking about the hyper-parameters, we can say what follows:

- the unique tunable hyper-param which is **var_smoothig parameter** assumes same value across the different trials, which means that amongst the purposed values for performing grid-search technique at train time all the models prefer to select the same value, which means that the choice of a particular kernel for performing kernel-Pca unsupervised learing method in order to pre-process data points do not affect importantly such a parameter.

In the end, 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 *Naive Bayes classifier*, for sure, we could employ the classifier found from *Rbf kernel Pca based Sgd Classifiers*.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. In fact we can adopt instead *linear, Polynomial and Sigmoid kernel Pca based Sgd Classifiers* for creating a kind of Boosting classifier.

When to Use Naive Bayes Because naive Bayesian classifiers make such stringent assumptions about data, they will generally not perform as well as a more complicated model. That said, they have several advantages:

- They are extremely fast for both training and prediction
- They provide straightforward probabilistic prediction
- They are often very easily interpretable
- They have very few (if any) tunable parameters

These advantages mean a naive Bayesian classifier is often a good choice as an initial baseline classification. If it performs suitably, then congratulations: you have a very fast, very interpretable classifier for your problem. If it does not perform well, then you can begin exploring more sophisticated models, with some baseline knowledge of how well they should perform.
Naive Bayes classifiers tend to perform especially well in one of the following situations:

- When the naive assumptions actually match the data (very rare in practice)
- For very well-separated categories, when model complexity is less important
- For very high-dimensional data, when model complexity is less important

The last two points seem distinct, but they actually are related: as the dimension of a dataset grows, it is much less likely for any two points to be found close together (after all, they must be close in every single dimension to be close overall). This means that clusters in high dimensions tend to be more separated, on average, than clusters in low dimensions, assuming the new dimensions actually add information. For this reason, simplistic classifiers like naive Bayes tend to work as well or better than more complicated classifiers as the dimensionality grows: once you have enough data, even a simple model can be very powerful.

3.2.1 Naive Bayes References

- (**Gnerative Model: Naive Bayes**) https://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes
- (**Useful Example: Out-of-core classification of text documents**) https://scikit-learn.org/stable/auto_examples/applications/plot_out_of_core_classification.html#sphx-glr-auto-examples-applications-plot-out-of-core-classification-py

4 Logistic Regression

4.1 Logistic Regression Properties

Learning Technique	Type of Learner	Type of Learning	Classification	Regression
Logistic Regression	*Linear Model*	*Supervised Learning*	*Supported*	*Not-Supported*

Logistic regression, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function.
In the logistic model, the log-odds (the logarithm of the odds) for the value labeled "1" is a linear combination of one or more independent variables ("predictors"); the independent variables can each be a binary variable (two classes, coded by an indicator variable) or a

continuous variable (any real value):
$\ell = \log_b \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$.

We can recover the odds by exponentiating the log-odds: $\frac{p}{1-p} = b^{\beta_0+\beta_1 x_1+\beta_2 x_2}$.

The above formula shows that once β_i are fixed, we can easily compute either the log-odds that $Y = 1$ for a given observation, or the probability that $Y = 0$ for a given observation. For Multiple explanatory variables it turns to be: $p = \frac{1}{1+b^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+\cdots+\beta_m x_m)}}$

By simple algebraic manipulation, the probability that $Y = 1$ is Logistic regression is implemented in LogisticRegression. This implementation can fit binary, One-vs-Rest, or multinomial logistic regression with optional l_1, l_2 or Elastic-Net regularization.

As an optimization problem, binary class l_2 penalized logistic regression minimizes the following cost function:

- $\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1)$

Similarly, l_1 regularized logistic regression solves the following optimization problem:

- $\min_{w,c} \|w\|_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1)$

Elastic-Net regularization is a combination of l_1 and l_2, so minimizes the following cost function:

- $\min_{w,c} \frac{1-\rho}{2} w^T w + \rho \|w\|_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1)$
- where ρ controls the strength of l_1 regularization vs. l_2 regularization

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

# Set seed for notebook repeatability
np.random.seed(0)
```

```
None
```

[2]:
```
#
  ↪============================================================================
  ↪#
# READ INPUT DATASET
#
  ↪============================================================================
  ↪#
dataset_path, dataset_name, column_names, TARGET_COL = get_dataset_location()
estimators_list, estimators_names = get_estimators()
```

[3]:
```
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)
```

4.2 Learning Models

[6]:
```
# 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)

# 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]
```

[7]:
```
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],
```

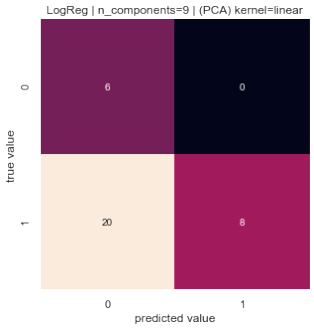
```
    verbose=0,
    by_pairs=True,
    savefigs=True,
    scoring='accuracy',
    figs_dest=os.path.join('figures', 'learning_curve',␣
  ↪f"Pcs_{n_components}"), ignore_func=True,
    # figsize=(20,5)
)
```

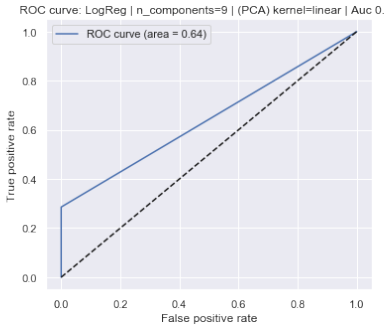
[8]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

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

[9]:
```
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[1], \
    param_grids=param_grids[0],
    estimators_names=estimators_names[1], \
    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
```

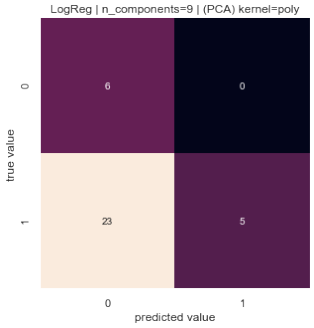
```
Kernel PCA: Linear | LogReg
=========================================================
              precision    recall  f1-score   support

     class 0       0.23      1.00      0.38         6
     class 1       1.00      0.29      0.44        28

    accuracy                           0.41        34
   macro avg       0.62      0.64      0.41        34
weighted avg       0.86      0.41      0.43        34

Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.97              0.41  0.64  0.13861
```

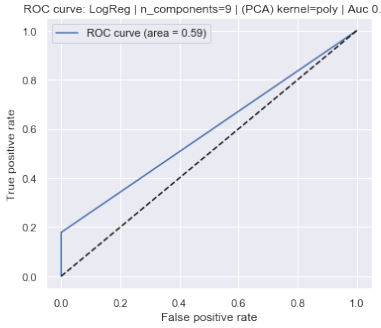


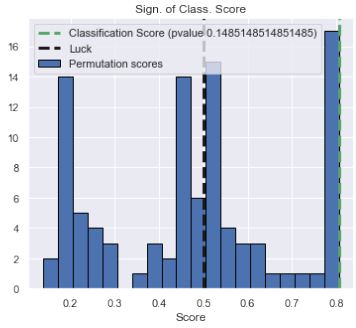


Kernel PCA: Poly | LogReg

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

     class 0       0.21      1.00      0.34         6
     class 1       1.00      0.18      0.30        28

    accuracy                           0.32        34
   macro avg       0.60      0.59      0.32        34
weighted avg       0.86      0.32      0.31        34

Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.92              0.32  0.59  0.14851
```

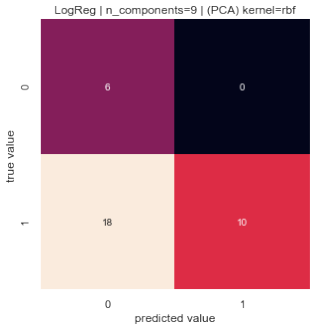


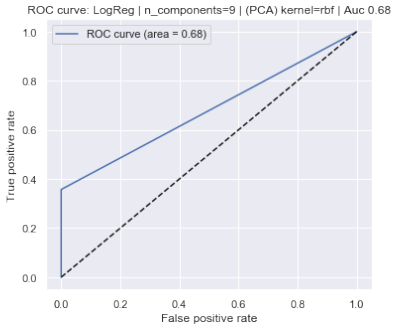


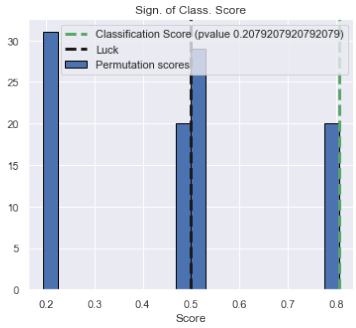


Kernel PCA: Rbf | LogReg

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

     class 0       0.25      1.00      0.40         6
     class 1       1.00      0.36      0.53        28

    accuracy                           0.47        34
   macro avg       0.62      0.68      0.46        34
weighted avg       0.87      0.47      0.50        34

Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.92              0.47  0.68  0.20792
```

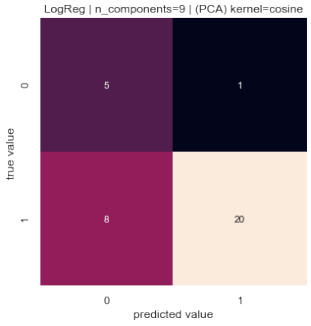


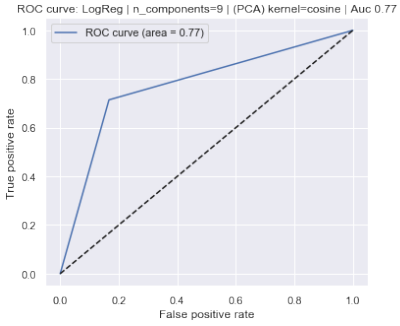




Kernel PCA: Cosine | LogReg

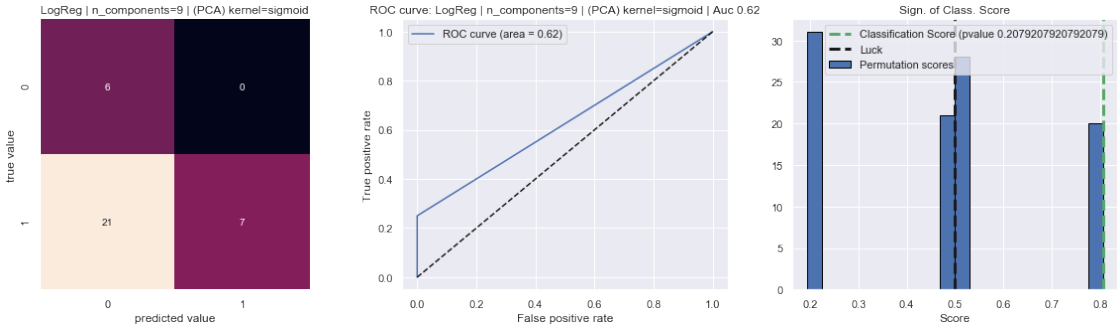
```
=======================================================
              precision    recall  f1-score   support

     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
                 0.92              0.74  0.77  0.81188
```





Kernel PCA: Sigmoid | LogReg

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

     class 0       0.22      1.00      0.36         6
     class 1       1.00      0.25      0.40        28

    accuracy                           0.38        34
   macro avg       0.61      0.62      0.38        34
weighted avg       0.86      0.38      0.39        34

Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.91              0.38  0.62  0.20792
```



Looking at the results obtained running *Logistic Regression 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 that generally speaking all the methods shonw a very high *Train Accuracy Score* which reaches in the most of the case a value greater than *90%*. However only one trial out of five, that is trial in which we adopted *Cosine Trick* we was able to account for *74%* of accuracy than the other cases where instead we do not reach a *train accuracy score* greater than *50%*. So, we can end up saying that the other models either overfit to the *train set* and wasn't able to generalize well on *test set*, or the fact that our dataset is not a balanced one leads to models and estimators that were able to correctly predict one among the two classes and more specifically, the models seems to recognize better the *class 0*, that is *Deck Bridges* than *class 1*, that is *Through Bridges*. In other words, usually working with unbalanced dataset we expect that the most frequent classes or most numerous classes were advantaged against the less numerous, but here emploping logistic Regression Classifier we obtained models that were more able to correctly classify the less numerous class and to worngly predict the more numerous class. More precisely:

1. Speaking about **Linear kernel Pca based logistic Classifier**, we can notice that such a model exploiting a default threshold of *.5* reaches a poor test accuracy score of just *41%* with respect to a train accuracy score of *97%*. The model indeed overfits to the overall train set and tends to better predict the less numerous class, so the model gains weight parameters suitable to identify class 0 samples. Looking at *recall and precision scores*, the model was really precise when predicting class 1 examples and was able to correcly predict labels for class 0, so maximzes recall of negative class. But we cannot say it is also precise when predicting class 0 this means that it wrongly inferes the true label for positive class. Lastly the model obtained high and low weighted average precision and recall, such that weigthed *F1-score* was low as well. Speaking about *Roc curve and Auc Score*, we can unbderstand that the model obtains a intermediate Auc score, of *.64* than the Random Classifier, and the relationship between *FPR and TPR* is linear most of the time changing the threshold value for classification.
2. Observing **Polynomial kernel Pca based logistic Classifier**, we can notice that such a model exploiting a default threshold of *.5* reaches even a lower test accuracy of *32%* than a still high train accuracy of *91%*. So also here for this trial the resulting model overfit to the train set and becaus of both lower accuracy scores we can can state that the model wrongly predicts a larger number of samples from class 1. In fact the model's precision and recall of class 0 and class 1 lowered than the values seens for the previous trial, while the precision and recall of class 1 and class 0 still remain the same, so this model preidcts with high precision samples from class 1 but with great uncertainty about class

0, even if most of the sample from such a classes were correctly labeled. Looking at *Roc Curve and Auc Score* we can observe that the best found model with this configuration indeed is going slightly bettere than random classifier, in fact has a auc score equals just to *.59* and the *TPR and FPR* behaviors is that them grows linearly while modifying the default threshold value most of the time.

3. Review **Rbf kernel Pca based logistic Classifier**, we can stricly and briefly saying that as the two preivous models also here discussing this estimator performance we do not obtain satisfying results in fact the model behaves more or less as the first reviewed, and more precisely the model obtained a slighlty better accuracy on test set of *.47* and a weighted F1-Score of *.5*, that allow for a Auc score that reaches a value of *.68* However also this model overfit to the train set with an accuracy score of *92%*, and is more able to correcly predict class 1 instances with high precision and class 0 instances with more uncertainty, even if has a high recall related to class 0.
4. **Cosine kernel Pca based logistic Classifier** results to be the best solution found while performing grid search algorithm for logistic Regression method, when it is fixed a defualt threshold of *.5* for classification. The rationale is that this trial retrieves a model that does not overfit to the train set, since test set accuracy is *74%*, just nearly 20 percent points than train accuracy score of *92%*. Moreover, we obtain high values for both *averaged precions, recalla and F1-Score metrics*, where the latter more precisely was even gretaer than test accuracy score, reachiung a value of *77%*. However, this model as others is less precise when predicting labels for class 0, than when inferring class 1 lables, this is mostly due to the fact that the dataset is not balanced. So we still remain more confident and precise when predicting class labels for class 1 examples. Looking at Roc Curve and Auc Score, we can say that for this model we have a curve which is able to account for up to *77%* of auc, and that this model works fine for many thresholds, in particular we can imagine to lower down a little bit the default threshold so that we can improve *TPR* reducing slightly *FPR* scores.
5. Lastly, also **Sigmoid kernel Pca based logistic Classifier** as other previous trials except the one represented by the moedl trained fixing Cosine Trick as kernel Pca method, gains poor and lower performance, due to overfit issue and as other more or less same low performance models generally speaking obtains high and low weighted average precision and recall scores, meaning that while the few instances predicted as belonging to class 1 was done with high precision instead of samples from class 0 which was prediceted with high uncertainty, even if most of the time the model correctly predicts instances that indeed belongs to class 0. The Roc Curve and Auc Score of *62%* show that also this run leads to a model which TPR and FPR are most of the time growing linearly across the thresholds.

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, 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 not grdi search result from training set that was able to overcome such cutt-off value of *%%* and therefore the different models are not uncertain enough to be adopted and configured with those hyperparameters and model's weights for describing the underling model related to the data.

Table Fine Tuned Hyper-Params (logistic Regression)

[10]:
```
# 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)
```

[10]:

	AUC(%)	P-Value(%)	Acc Train(%)	Acc Test(%)	C	class_weight
LogReg linear	0.64	13.86	0.97	0.41	0.001	balanced
LogReg poly	0.59	14.85	0.92	0.32	0.001	balanced
LogReg rbf	0.68	20.79	0.92	0.47	0.001	balanced
LogReg cosine	**0.77**	**81.19**	**0.92**	**0.74***	**1.0**	**None**
LogReg sigmoid	0.62	20.79	0.91	0.38	0.001	balanced

	fit_intercept	penalty	solver	tol
LogReg linear	True	l2	sag	0.001
LogReg poly	True	l2	sag	0.001
LogReg rbf	True	l2	sag	0.0001
LogReg cosine	**True**	**l2**	**liblinear**	**0.0001**
LogReg sigmoid	True	l2	sag	0.0001

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 *Linear, Polynomial, Rbf, and Sigmoid Tricks* or less overfit solution such as in the case of *Cosine Trick*. Speaking about the hyper-parameters, we can say what follows:

- Speaking about the **hyper-param C**, that is inverse of regularization strength where smaller values specify stronger regularization, we observe that except the Cosine kernel trick case all other kernel-Pca tricks adopted have preferred to exploit a very low value for *C parameter* equals to *0.001* and accounts for a very strtong regularization, but such a choice does not lead to models that obtained a high generalization capability, instead the *Cosine based kernel-Pca* model opted for a default value for such a parameter.

- Instead referring to **class_weight parameter**, we knwo that it can be set with balanced strategy which stends for a strategy where 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 have been surprised that all the method that obtained worst performance choose a balanced strategy than the best model which was fine even with a default strategy that does not require to use a balanced mode.

- Whereas **fit_intercept parameter** refers to the fact that we specify if a constant (a.k.a. bias or intercept) should be added to the decision function, and allows for modeling a certain behavior and a certain response different from zero even when the input sample is mostly made of zero components, we can understand that in all the cases the models

obtained best results enabling such strategy and so the models are fitted taking into account also a intercept weight or parameter, increasing model complexity.

- Model's **penalty parameter** allows to specify the norm used in the penalization, among the folowing list of possible choices *l1, l2, elasticnet*. In all the models the best choice was for *l2 regularization*, this means that all the models opted for a kind of regularization that do not consider at all the *l1 normalization* as a regularization technique, so we avoid to obtain models that instead may lead weights to zero values, in other words sparse models.
- Model's **solver parameter** which is the algorithm to use in the optimization problem. It is curios to notice that almost all the models except cosine based kernel-Pca which adopted *liblinear* solver. What we can understand is that for all the overfitted models the choice of *sag* solver does not lead to significant results in term of performance, and we can say instead that we correctly except that for such a small dataset a *liblinear* choice is the most suitable and the best model found here is coherent with such a suggestion from theory field.
- lastly, looking at **tol parameter**, which stends for tolerance for stopping criteria, we can clearly see that the first two models adopted a lowe tolerance value instead the last three preferred a lower value of tolerance, so the first two methods accordingly with the kind of kernel trick technique adopted for kernel-Pca seem to go well when a tolerance value is not so small as the last three methods, furthermore the first two methods request less time than the last three because of the lareger tolerance set for training convergence.

4.2.1 Logistic Regression References

- (Linear Model: Logistic Regression)
  https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

5 Knn

5.1 Knn Classifier Properties

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*

In *Pattern Recognition*, the *K-Nearest Neighbors Algorithm (k-NN)* is a **non-parametric method** used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The sections/knn-classifier/output depends on whether k-NN is used for classification or regression:

- In *k-NN classification*, the sections/knn-classifier/output is a class membership. An object

is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

- In *k-NN regression*, the sections/knn-classifier/output is the property value for the object. This value is the average of the values of k nearest neighbors.

What follows is a briefly explanation of Knn:

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point. A commonly used distance metric for continuous variables is *Euclidean distance*.

Example of k-NN classification. The test sample (green dot) should be classified either to blue squares or to red triangles. If k = 3 (solid line circle) it is assigned to the red triangles because there are 2 triangles and only 1 square inside the inner circle. If k = 5 (dashed line circle) it is assigned to the blue squares (3 squares vs. 2 triangles inside the outer circle).

Choice of Nearest Neighbors Algorithm: The optimal algorithm for a given dataset is a complicated choice, and depends on a number of factors, number of samples N (i.e. n_samples) and dimensionality D (i.e. n_features): - *Brute force* query time grows as $O[DN]$ - *Ball tree* query time grows as approximately $O[D\log(N)]$ - *KD tree* query time changes with D in a way that is difficult to precisely characterise. For small D (less than 20 or so) the cost is approximately $O[D\log(N)]$, and the KD tree query can be very efficient. For larger D, the cost increases to nearly O[DN], and the overhead due to the tree structure can lead to queries which are slower than brute force.

Therefore, we end up saying that, for small data sets (N less than 30 or so), $\log(N)$ is comparable to N, and brute force algorithms can be more efficient than a tree-based approach. Both KDTree and BallTree address this through providing a leaf size parameter: this controls the number of samples at which a query switches to brute-force. This allows both algorithms to approach the efficiency of a brute-force computation for small N.

[1]:
```
from utils.all_imports import *;
%matplotlib inline
# Set seed for notebook repeatability
np.random.seed(0)
```

```
None
```

[2]:
```
# 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)
```

[3]:
```
columns_2_avoid = ['ERECTED', 'LENGTH', 'LOCATION']
```

[4]:
```
# 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)
```

5.2 Learning Models

[5]:
```
# 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_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)

# 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]
```

[6]:
```
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],
```

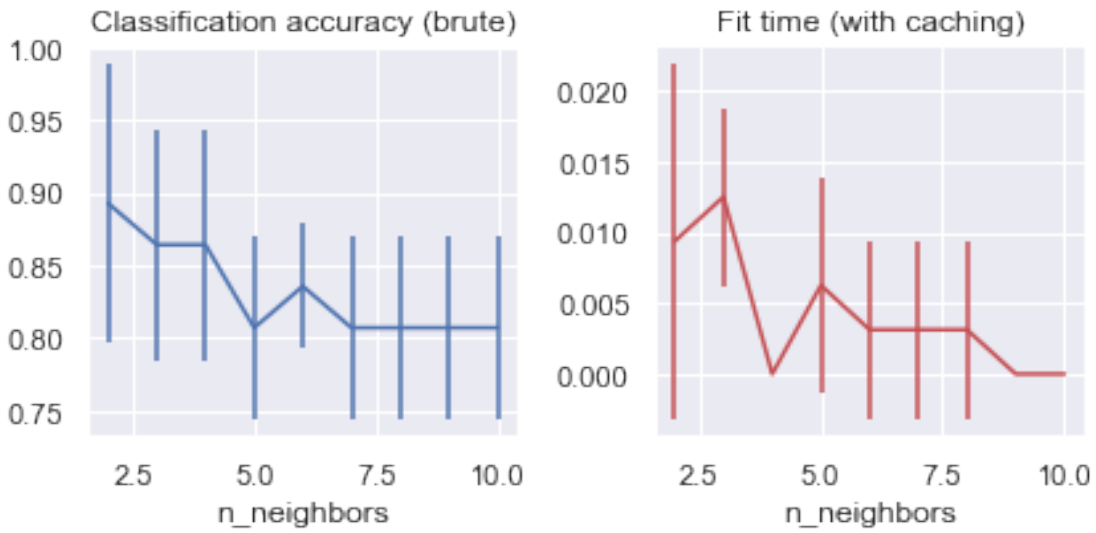
```
    verbose=0,
    by_pairs=True,
    savefigs=True,
    scoring='accuracy',
    figs_dest=os.path.join('figures', 'learning_curve',␣
  ↪f"Pcs_{n_components}"), ignore_func=True,
    # figsize=(20,5)
)
```

[7]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

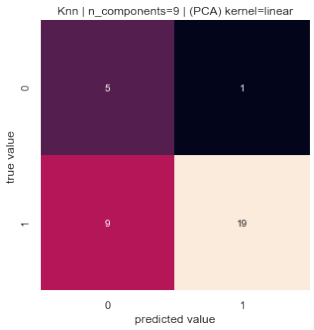
```
<IPython.core.display.Javascript object>
```

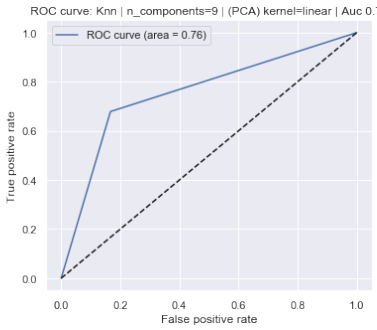
[8]:
```
plot_dest = os.path.join("figures", "n_comp_9_analysis", "grid_search")
X = rescaledX; pos = 2

df_gs, df_auc_gs, df_pvalue = grid_search_all_by_n_components(
    estimators_list=estimators_list[pos], \
    param_grids=param_grids[0],
    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
```



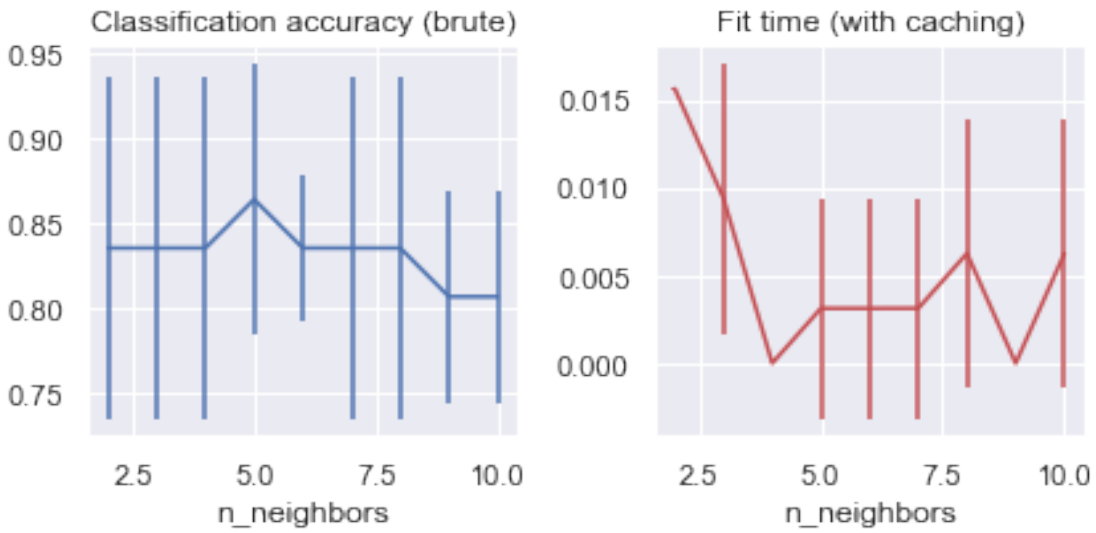
Best Score (CV-Train)	Best Score (Test)	AUC	P-value
0.92	0.71	0.76	0.32673



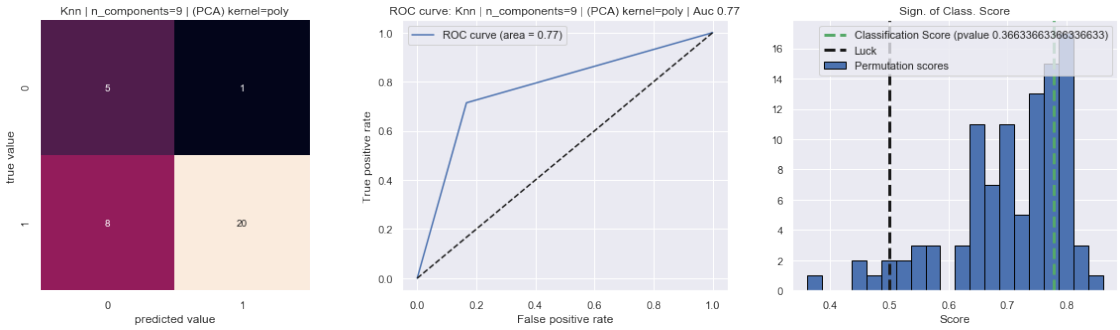


Kernel PCA: Poly | Knn

```
=======================================================
              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
```

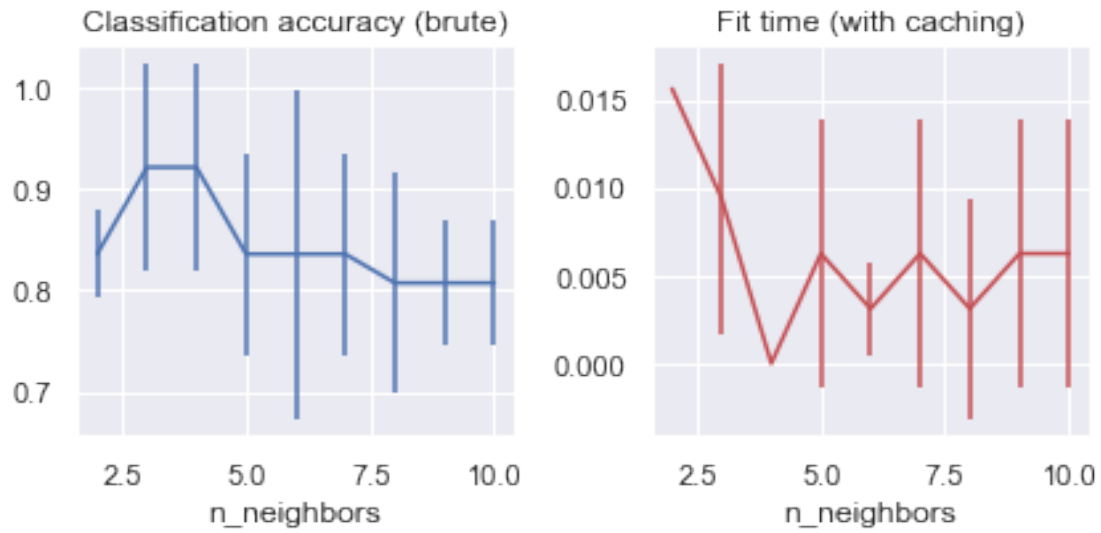



```
Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.89                0.74  0.77  0.36634
```

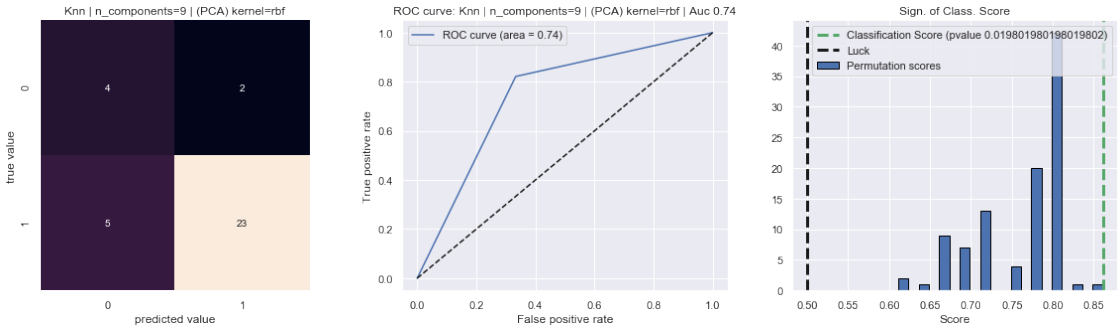


Kernel PCA: Rbf | Knn

```
=====================================================
              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
```

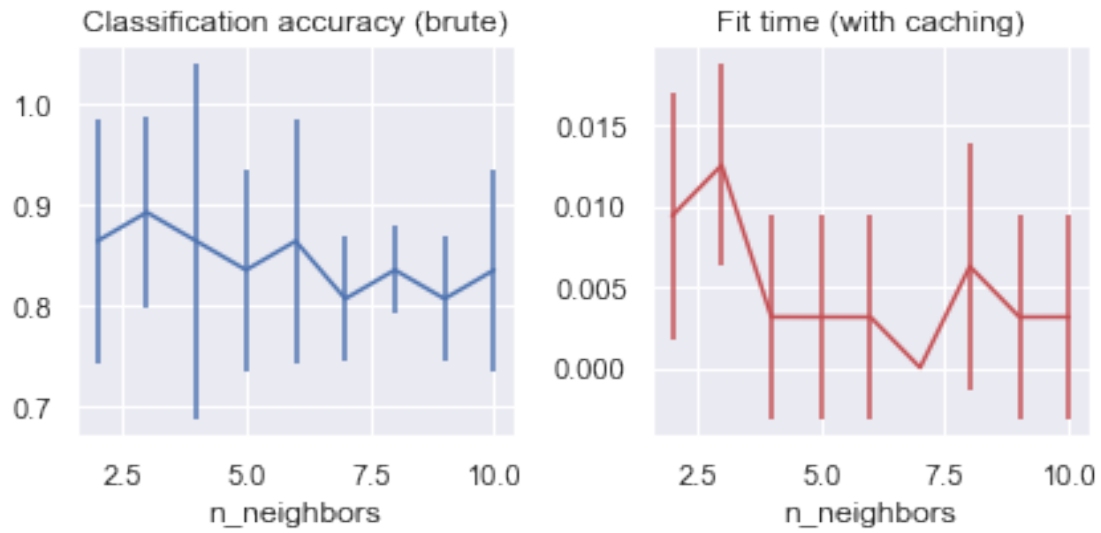


Best Score (CV-Train)	Best Score (Test)	AUC	P-value
0.95	0.79	0.74	0.01980

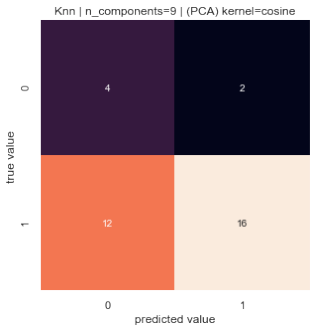


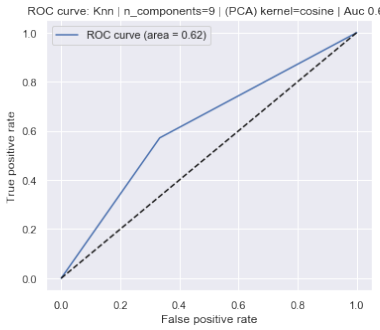
Kernel PCA: Cosine | Knn

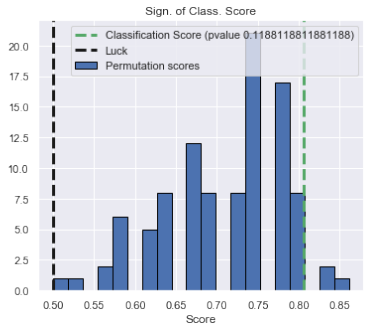
	precision	recall	f1-score	support
class 0	0.25	0.67	0.36	6
class 1	0.89	0.57	0.70	28
accuracy			0.59	34
macro avg	0.57	0.62	0.53	34
weighted avg	0.78	0.59	0.64	34



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

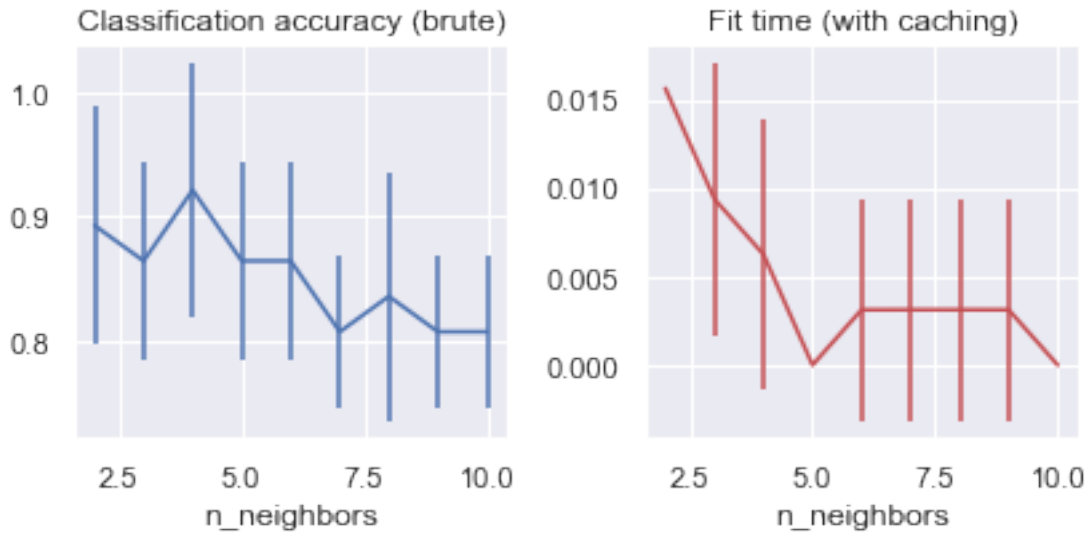




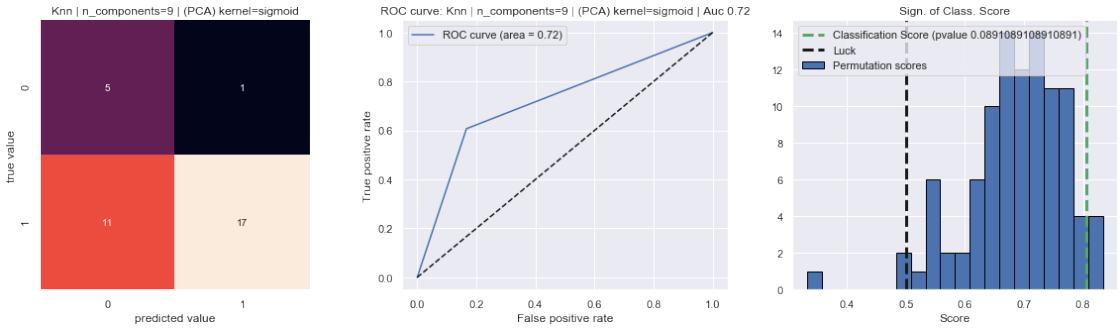


Kernel PCA: Sigmoid | Knn

	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:

1. 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.

2. 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.
3. 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.
4. 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.
5. 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.

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, 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. Two classifier out of five, which are *Linear- and Poly-kernel Pca based Knn Classifier* have a p-value that widely exceeds the value set for significance level, so for those two cases rejecting Null-Hypothesis will causes a *Type I error*. Also looking at *Cosine- and Sigmoid-kernel Pca based Knn Classifier* we can state same conclusions as just above for the two previous models. While just *Rbf-kernel Pca based Knn Classifier* seems the right classifier that obtained a p-value lower than the pre-defined p-value of *5%*, so we have to select this method as the one which allows us to reject the Null-Hypothesis and adopting such a classifier for describing the data behavior.

Table Fine Tuned Hyper-Params(Knn Classifier)

[9]:
```
# 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)
```

[9]:

	AUC(%)	P-Value(%)	Acc Train(%)	Acc Test(%)	algorithm	leaf_size	metric	n_neighbors	weights
Knn linear	0.76	32.67	0.92	0.71	ball_tree	5	euclidean	3	distance
Knn poly	0.77	36.63	0.89	0.74	ball_tree	5	euclidean	3	distance
Knn rbf	**0.74**	**1.98**	**0.95**	**0.79***	**ball_tree**	**5**	**euclidean**	**7**	**distance**
Knn cosine	0.62	11.88	0.92	0.59	ball_tree	5	euclidean	3	distance
Knn sigmoid	0.72	8.91	0.92	0.65	ball_tree	5	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.

5.2.1 K-Nearest Neighbor References

- (Neighbor-based Learning: Knn) https://scikit-learn.org/stable/modules/neighbors.html

6 Stocastic Gradient Descent

6.1 Stocastic Gradient Descent Properties

Learning Technique	Type of Learner	Type of Learning	Classification	Regression	Clustering
Stochastic Gradient Descent (SGD)	*Linear Model*	*Supervised Learning*	*Supported*	*Supported*	*Not-Supported*

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to fitting linear classifiers and regressors under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning.
SGD has been successfully applied to large-scale and sparse machine learning problems often encountered in *text classification* and *natural language processing*. Given that the data is sparse, the classifiers in this module easily scale to problems with more than 10ˆ5 training examples and more than 10ˆ5 features.
Mathematical formulation: we describe here the mathematical details of the SGD procedure.
Given a set of training examples $(x_1, y_1), \ldots, (x_n, y_n)$ where $x_i \in \mathbf{R}^m$ and $y_i \in \mathcal{R}$ ($y_i \in -1, 1$ for classification), our goal is to learn a linear scoring function $f(x) = w^T x + b$ with model parameters $w \in \mathbf{R}^m$ and intercept $b \in \mathbf{R}$. In order to make predictions for binary classification, we simply look at the sign of $f(x)$. To find the model parameters, we minimize the regularized training error given by:

- $E(w, b) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w)$
- where **L** is a loss function that measures model (mis)fit and **R** is a regularization term (aka penalty) that penalizes model complexity; $\alpha > 0$ is a non-negative hyperparameter that controls the regularization strength.

Different choices for **L** entail different classifiers or regressors:

- Hinge (soft-margin): equivalent to Support Vector Classification: $L(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i))$.
- Perceptron: $L(y_i, f(x_i)) = \max(0, -y_i f(x_i))$
- Modified Huber: $L(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i))^2$ if $y_i f(x_i) > 1$, otherwise in $L(y_i, f(x_i)) = -4 y_i f(x_i)$

- Log: equivalent to Logistic Regression: $L(y_i, f(x_i)) = \log(1 + \exp(-y_i f(x_i)))$
- Least-Squares: Linear regression (Ridge or Lasso depending on **R**): $L(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2$.
- Huber: less sensitive to outliers than least-squares. It is equivalent to least squares when $|y_i - f(x_i)| \leq \varepsilon$, and $L(y_i, f(x_i)) = \varepsilon|y_i - f(x_i)| - \frac{1}{2}\varepsilon^2$ otherwise.
- Epsilon-Insensitive: (soft-margin) equivalent to Support Vector Regression: $L(y_i, f(x_i)) = \max(0, |y_i - f(x_i)| - \varepsilon)$

Finally, popular choices for the regularization term (the penalty parameter) include:

1. L2 norm: $R(w) := \frac{1}{2}\sum_{j=1}^{m} w_j^2 = ||w||_2^2$
2. L1 norm: $R(w) := \sum_{j=1}^{m} |w_j|$,which leads to sparse solutions.
3. Elastic Net: $R(w) := \frac{\rho}{2}\sum_{j=1}^{n} w_j^2 + (1-\rho)\sum_{j=1}^{m} |w_j|$, a convex combination of L2 and L1, where ρ is given by $1 - l1_{ratio}$.

Adavantages and Backwards: the advantages of Stochastic Gradient Descent are:

- Efficiency.
- Ease of implementation (lots of opportunities for code tuning).
- Complexity: The major advantage of SGD is its efficiency, which is basically linear in the number of training examples. If X is a matrix of size (n, p) training has a cost of $O(kn\bar{p})$, where k is the number of iterations (epochs) and $\bar{p}$ is the average number of non-zero attributes per sample.

The disadvantages of Stochastic Gradient Descent include:

- SGD requires a number of hyperparameters such as the regularization parameter and the number of iterations.
- SGD is sensitive to feature scaling.

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

```
None
```

[2]:
```
# Some global script variables
#␣
  ↪--------------------------------------------------------------------------␣
  ↪#
dataset_path, dataset_name, column_names, TARGET_COL = \
    get_dataset_location() # Info Data to be fetched
estimators_list, estimators_names = get_estimators() # Estimator to be␣
  ↪trained

dataset, feature_vs_values = load_brdiges_dataset(dataset_path, dataset_name)

# variables used for pass through arrays used to store results
pos_gs = 0; pos_cv = 0

# 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))
```

[3]:
```
columns_2_avoid = ['ERECTED', 'LENGTH', 'LOCATION']
```

[4]:
```
# 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)
```

[5]:
```
# Parameters to be tested for Cross-Validation Approach
# ------------------------------------------------------
param_grids = []
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)
# 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]
```

[6]:
```
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,
```

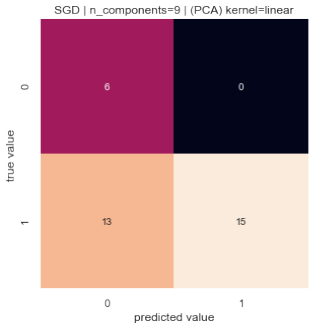
```
    by_pairs=True,
    savefigs=True,
    scoring='accuracy',
    figs_dest=os.path.join('figures', 'learning_curve',␣
  ↪f"Pcs_{n_components}"), ignore_func=True,
    # figsize=(20,5)
)
```

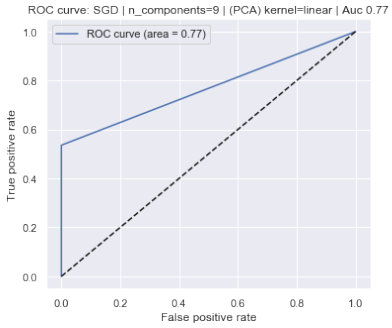
[7]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

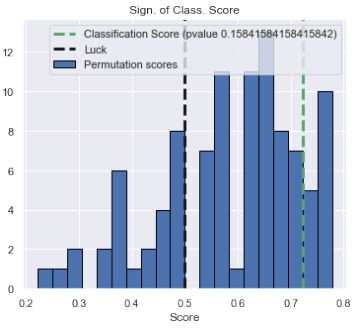
```
<IPython.core.display.Javascript object>
```

[8]:
```
plot_dest = os.path.join("figures", "n_comp_9_analysis", "grid_search")
X = rescaledX; pos = 3

df_gs, df_auc_gs, df_pvalue = grid_search_all_by_n_components(
    estimators_list=estimators_list[pos], \
    param_grids=param_grids[0],
    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 | SGD
=====================================================
              precision    recall  f1-score   support

     class 0       0.32      1.00      0.48         6
     class 1       1.00      0.54      0.70        28

    accuracy                           0.62        34
   macro avg       0.66      0.77      0.59        34
weighted avg       0.88      0.62      0.66        34

Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.95              0.62  0.77  0.15842
```



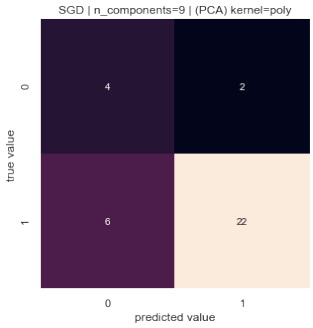


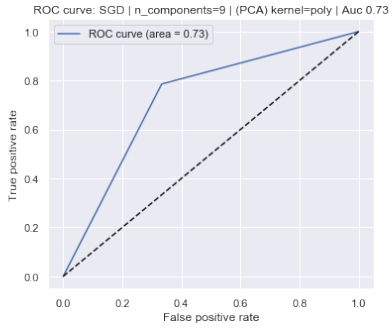


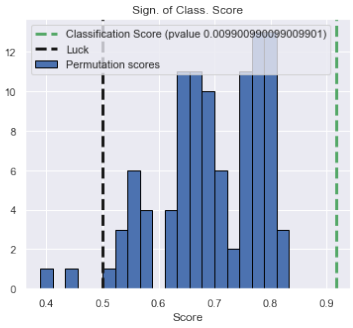
Kernel PCA: Poly | SGD

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

     class 0       0.40      0.67      0.50         6
     class 1       0.92      0.79      0.85        28

    accuracy                           0.76        34
   macro avg       0.66      0.73      0.67        34
weighted avg       0.83      0.76      0.79        34
```

Best Score (CV-Train)	Best Score (Test)	AUC	P-value
0.92	0.76	0.73	0.00990



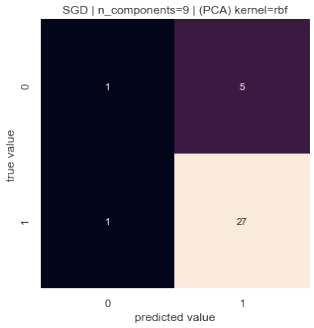


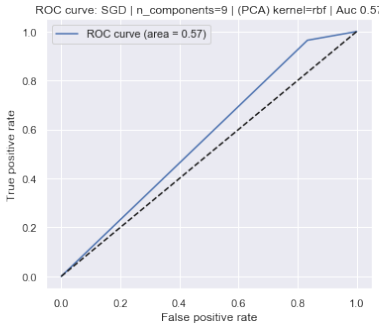


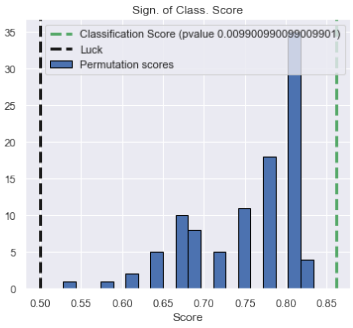
Kernel PCA: Rbf | SGD

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

     class 0       0.50      0.17      0.25         6
     class 1       0.84      0.96      0.90        28

    accuracy                           0.82        34
   macro avg       0.67      0.57      0.57        34
weighted avg       0.78      0.82      0.79        34
```

```
Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.92                0.82  0.57  0.00990
```

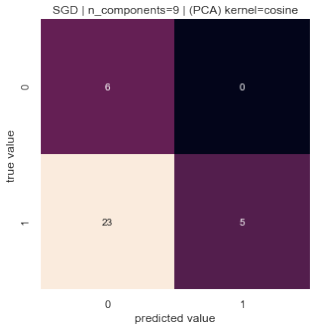


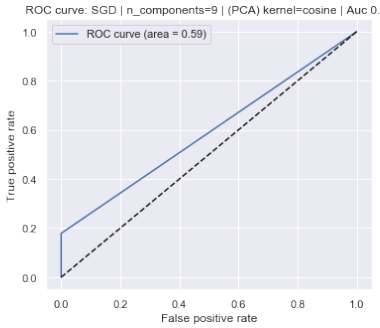


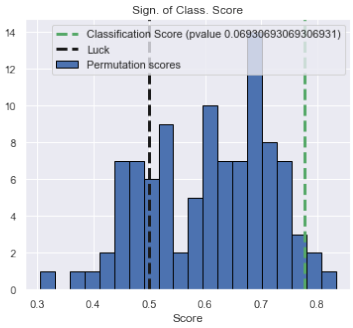



```
Kernel PCA: Cosine | SGD
=======================================================
              precision    recall  f1-score   support

     class 0       0.21      1.00      0.34         6
     class 1       1.00      0.18      0.30        28

    accuracy                           0.32        34
   macro avg       0.60      0.59      0.32        34
weighted avg       0.86      0.32      0.31        34
```

```
Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.95                0.32  0.59  0.06931
```

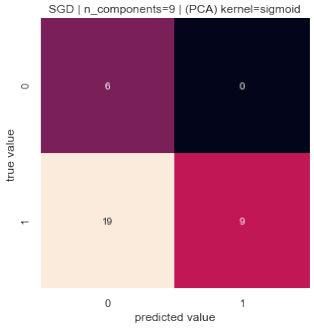


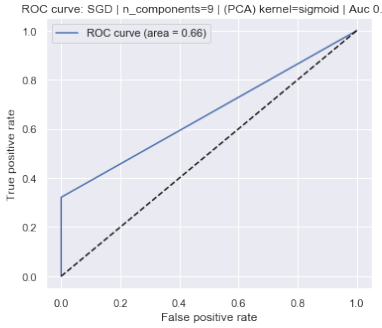


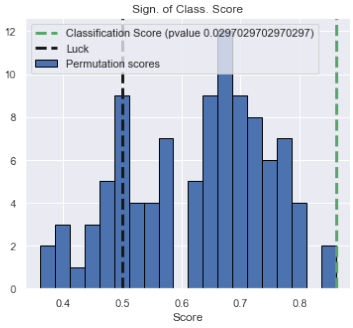



```
Kernel PCA: Sigmoid | SGD
=======================================================
              precision    recall  f1-score   support

     class 0       0.24      1.00      0.39         6
     class 1       1.00      0.32      0.49        28
```

```
    accuracy                           0.44        34
   macro avg       0.62      0.66      0.44        34
weighted avg       0.87      0.44      0.47        34

Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.92                0.44  0.66  0.02970
```







Looking at the results obtained running *Sgd 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 looking at the weighted values of *Recall, Precision, and F1-Scores* we obtained good performance and and except for one trial where we got lower and worst results, when *Polynomial and Rbf Tricks* is selected, in the remaning cases have gotten remarkable results. More precisely we can say what follows:

1. Speaking about **Linear kernel Pca based Sgd Classifier**, when adoping the default threshold of *.5* for classification purposes we have a model that reaches an accuracy of *65%* at test time against an accuracy of *92%* at train step, while the Auc score reaches a value of *79%* with a Roc Curve that shows a behavior for which the model increases its *TPR* without affecting the *FPR* score, however at a given point the Roc Curve trend turns so that the two cited scores begin to increase linearly and with a slope lower than that of Random Classifier so that FPR increases faster. The model is very precise when predicting class 1 instances but it has a recall of just *54%* so misclassified more or less half of samples from class 1 and this fact influenced instead the precision of class 0 that is a bit low, just *32%*, while class 0 recall is very high. Since the test accuracy score loses nearly 30 percent points we can assume that sucha model quite overfit to train data, we are not really encouraged to adopt it except we decied to exploit it for including it in an ensemble classifier, more boosting like than bagging one.
2. Observing **Polynomial kernel Pca based Sgd Estimator**, we can notice that such a model exploiting a default threshold of *.5* reaches an accuracy of *76%* at test time against an accuracy of *92%* at train step, while the Auc score reaches a value of *73%*. It represents the best result obtained running th SGD based Training Algorithm upon our input dataset, in particular it obtained high precision and high recall for class 1, in other words such a model is able to recognize and correctly classify most of the data examples whose true label is indeed class 1. However, even if the model has high recall related to class 0, since the dataset is unbalanced we cannot say the same things for precision score about the class 0. So the model is somewhat uncertain when predicting class 0 as

label value for new observations.

3. Review **Rbf kernel Pca based Sgd Classifier**, we can notice that such a model exploiting a default threshold of *.5* reaches an accuracy of *82%* at test time against an accuracy of *92%* at train step, while the Auc score reaches a value of *57%*. In particular such a trial along with the *PCosine kernel Pca based Sgd Classifier* are the two attempts that lead to worts results, since the model overfit against the data employed at training time, but also the model gained weights that tend to predict every thing as class 1 instance. So, the resulting scores tell us that the model is highly precise and obtained high recall related to class 1, convercely has very low performance for precision and recall referred to class 0. Since such a model is performing just a little bit better than random classifier, can be largely adopted along other similar models for building voting classifier, following boosting like classifier policy and behavior.
4. Looking at **Cosine kernel Pca based Sgd Classifier**, we can notice that such a model exploiting a default threshold of *.5* reaches an accuracy of *32%* at test time against an accuracy of *95%* at train step, while the Auc score reaches a value of just *59%*. Here the fine tuned model obtained from grid-search approach tells us that we are able to classify with high precision a few data examples from class 1, and even if we correctly classify all instances from class 0, we also wrongly predict class labels for most of instances,. whose true label is class 1. This means that the model is highly uncertain when predicting class 0 as the sections/sgd-classifier/output target label. Moreover, the model's ROC Curve performs slighltly better than the random classifier, and we end up saying that such a model has gained weights and hyper-params that tend to predict the unknown instances as belonging to class 0 most of the time. We cannot neither say that switching the class labels between the two classes will allow us to obtain a better result since the roc curve trend is just a little bit better than the random classifier.
5. Finally, referring to **Sigmoid kernel Pca based Sgd Model**, we can notice that such a model exploiting a default threshold of *.5* reaches an accuracy of *44%* at test time against an accuracy of *92%* at train step, while the Auc score reaches a value of *66%*. This model behaves more or less as the model obtained from the first trial performed for Sgd-based classifier, so as the first model is slightly worst than the best model found here when adopting as classifier Sgd technique, that is the *Cosine kernel Pca based Sgd Classifier*.

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 SGD 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 calim that only two out of five trials lead to a *p-vlaue* worst than *selected significance level equal to 5%*, which are *Linear- 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 *Poly-, Rbf- and Sigmoid-kernel Pca based Sgd Classifier* have obtained a p-value over the range $[.9, 3]$ *in percet points*, so we are satisfyed for the results obtained in terms of significance scores, however, only *Poly-, and Rbf-kernel Pca based Sgd Classifier* really matter or are worth models since they do not overfit too

much and do not go worstly as *Sigmoid-kernel Pca based Sgd Classifier* at test time.

Table Fine Tuned Hyper-Params(SGD Classifier)

[9]:
```
# 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)
```

[9]:

	AUC(%)	P-Value(%)	Acc Train(%)	Acc Test(%)	alpha	class_weight
SGD linear	0.77	15.84	0.95	0.62	0.1	balanced
SGD poly	**0.73**	**0.99**	**0.92**	**0.76***	**0.1**	**None**
SGD rbf	0.57	0.99	0.92	0.82	0.1	None
SGD cosine	0.59	6.93	0.95	0.32	0.0001	balanced
SGD sigmoid	0.66	2.97	0.92	0.44	0.001	balanced

	learning_rate	loss	max_iter	penalty	tol
SGD linear	optimal	modified_huber	50	l1	None
SGD poly	**optimal**	**modified_huber**	**50**	**l2**	**None**
SGD rbf	optimal	modified_huber	50	l2	None
SGD cosine	optimal	modified_huber	50	l1	None
SGD sigmoid	optimal	modified_huber	50	l1	None

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 *Rbfd Trick* or less overfit solution such as in the case of *Linear, Polynomial, Cosine, and Sigmoid Tricks*. Speaking about the hyper-parameters, we can say what follows:

- Looking at **alpha hyper-parameter**, that is constant that multiplies the regularization term. The higher the value, the stronger the regularization. Also used to compute the learning rate when set to *learning_rate* is set to *'optimal'*, as was here, we can notice that the final choice through the different trials was more or less tha same, meanning that the adopted kernel trick for performing kernel-Pca does not affected appreciably such a hyper-param, which three cases out of five was set to *0.1*, and the remaining case adopted *0.0001*, *0.001* for respectively Cosine and Sigmoid based *kernel-Pca*. This also remind us that while training the classifiers was not necessary to force a high regularization contribute for reducing the overfit as well as the learning process, even if we know that *Rbf kernel Pca based Sgd Classifier* overfits mostly against train data, and gained weights that encourages predicting all samples as belonging to class 1.
- Reviewing **class_weight hyper-param**, what we can state about such a parameter is that it represents weights associated with classes. If not given, all classes are supposed to have weight one. 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))**. In particular we can notice that three out five models

that were fine tuned accepted or selected *balanced weights*, which are *Linear-, Sigomoid-, Cosine-kernel Pca based Sgd Classifier*, while the remaining obtain better, when setting uniform weights which are models *Polynomial-, Rbf-kernel Pca based Sgd Classifier*. So the choiche of the right *kernel-trick* affected the subsequent selection at fine tuning time of the *class_weight hyper-param*. What we can further notice is that *Polynomial-, Rbf-kernel Pca based Sgd Classifier* more or less adopted same kind of values for hyper-params, as an instance for penalty hyper-param, however Polynomial model got worst performance in terms of accuracy but considering the other metrics simultaneously we can understand that the Poly model overfits less than Rbf one and so get better performance in general.

- Speaking of **learning_rate hyper-param**, since we force this as the unique available choice it was just report for completeness.
- Interesting it is the discussion about **loss parameter**, if fact we know that the possible options are *'hinge'*, *'log'*, *'modified_huber'*, *'squared_hinge'*, *'perceptron'*, where the *'log' loss* gives logistic regression, a probabilistic classifier. *'modified_huber'* is another smooth loss that brings tolerance to outliers as well as probability estimates. *'squared_hinge'* is like hinge but is quadratically penalized. 'perceptron' is the linear loss used by the perceptron algorithm. Here, speaking about loss parameter we can clearly understand that the choice of a particular kernel trick does not affect the following choice of the loss function to be optimized, in fact uniformly all the models adopted or tend to prefer *modified_huber* loss function, allowing the models to fit to the data taking into account the fact that such a loss function is less sensitive to outliers, recalling inn fact that the Huber loss function is used in robust statistics, M-estimation and additive modelling. This loss is so cllaed beacues it derives from the plain version normally exploited for regression problems.
- Also when referring to **max iteration parameter**, we can easily say that thte models evenly adopted somewhat small number of iteration before stopping the learning procedure, this might be also becaues we work with a small dataset and so a set of data points that is small tend to overfit quickly and this migth be the reason for which in order to avoid too much overfit the training procedure performed employing grid-search technique for fine-tuning tend to prefer tiny number of iterations set for training the model.
- **penalty parameter**, we recall here that it represents regularization term to be used. More precisely, defaults to *'l2'* which is the standard regularizer for linear SVM models. *'l1'* and *'elasticnet'* might bring *sparsity* to the model (feature selection) not achievable with *'l2'*. Also for such a hyper-param the choice of a particular *kernel-trick* to be used for *kernel-Pca* was affecting the subsequent selection of penalty contribute to regularize learning task, as was for *class weight hyper-param*. Here three over five models that are *Linear-, Sigomoid-, Cosine-kernel Pca based Sgd Classifier* adopted *l1-norm* as regularization term so the models's weights tend to be more sparse, while the remaining *Polynomial-, Rbf-kernel Pca based Sgd Classifier* models adopted *l2-nrom*. For the trials we have done, the models with *l1-regularization term* seem to get worst performance, more precisely *Sigomoid-, Cosine-kernel Pca based Sgd Classifier* even were worser than random classifier, while the *Linear-kernel Pca based Sgd Classifier* was slightly worst than Polynomial one, so does not overfit too much however we can say it can be exploited

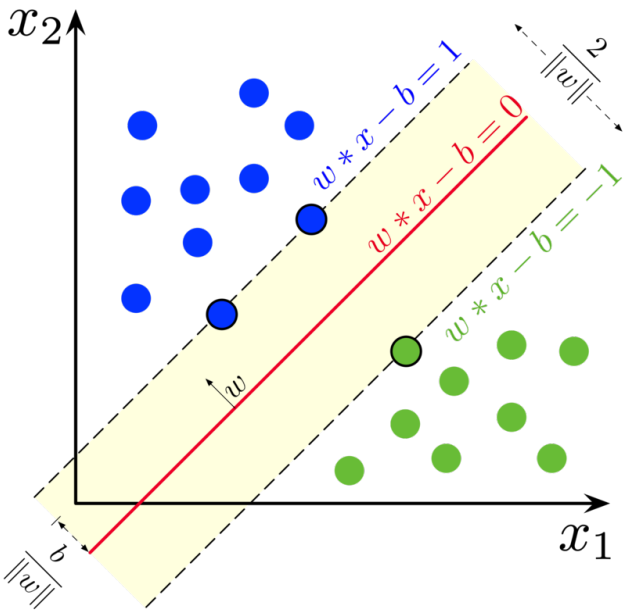
for ensemble method that follows a Boosting Policy.
If we imagine to build up an *Ensemble Classifier* from the family of *Average Methods*, which state that the underlying principle leading their creation requires to build separate and single classifiers than averaging their prediction in regression context or adopting a majority vote strategy for the classification context, we can claim that amongst the purposed *Sgd classifier*, for sure, we could employ the classifier found from all the trials, except for *Rbf, Cosine and Sigmoid kernel Pca based Sgd Classifiers*, since the first model is overly overfitting to the data used at train time and more precisely most of the time predicted correctly just samples from class 1 and misclassifyes instances from class 0, the others instead assumed the opposite behavior. Also, 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.

6.1.1 Stochastic Gradient Descent Classifier References

- (Stochastc Learning: SGD Classifier) https://scikit-learn.org/stable/modules/sgd.html#sgd

7 Support Vector Machines Classifier

7.1 Support Vector Machines Classifier Properties



Learning Technique	Type of Learner	Type of Learning	Classification	Regression	Clustering	Outlier Detection
Support Vector Machines(SVMs)	*Discriminative Model*	*Supervised Learning*	*Supported*	*Supported*	*Not-Supported*	*Supported*

Here, in this section I'm going to exploit a machine learning techinique known as Support Vectors Machines in order to detect and so select the best model I can produce throughout the usage of data points contained within the dataset at hand. So let discuss a bit those kind of classifiers.
In machine learning, **support-vector machines**, shortly SVMs, are *supervised learning models* with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a *non-probabilistic binary linear classifier*. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.
More formally, a support-vector machine constructs a hyperplane or set of hyperplanes in a high-dimensional space, which can be used for classification, regression. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class, so-called *functional margin*, since in general the larger the margin, the lower the *generalization error* of the classifier.

Mathematical formulation of SVMs Here, I'm going to describe the main mathematical properties and characteristics used to derive from a math point of view the algorithm derived and proven by researches when they have studied SVMs classifiers.
I start saying and recalling that A support vector machine constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space, which can be used for classification, regression or other tasks. Intuitively, a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class so-called functional margin, since in general the larger the margin the lower the generalization error of the classifier.
When demonstrating SVMs classifier algorithm I suppose that We are given a training dataset of nn points of the form:

$$(\vec{x_1}y_1), .., (\vec{x_n}, y_n) \tag{2}$$

where the y_1 are either 1 or -1, each indicating the class to which the point $\vec{x}_i$ belongs. Each $\vec{x}_i$is a *p-dimensional real vector*. We want to find the *maximum-margin hyperplane* that divides the group of points $\vec{x}_i$ for which $y_1 = 1$ from the group of points for which $y_1 = 1$, which is defined so that the distance between the hyperplane and the nearest point $\vec{x}_i$ from either group is maximized.
Any hyperplane can be written as the set of points $\vec{x}_i$ satisfying : $\vec{w}_i \cdot \vec{x}_i - b = 0$; where $\vec{w}_i$ is the, even if not necessarily, normal vector to the hyperplane. The parameter $\frac{b}{\|\vec{w}\|}$ determines the offset of the hyperplane from the origin along the normal vector $\vec{x}_i$.
Arrived so far, I have to distinguish between two distinct cases which both depend on the nature of data points that generally made up a given dataset. Those two different cases are called *Hard-Margin* and *Soft Margin* and, respectively.
The first case, so the ***Hard-Margin*** case, happens just for really optimistics datasets. In fact it is the case when the training data is linearly separable, hence, we can select two parallel

hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the *margin*, and the maximum-margin hyperplane is the hyperplane that lies halfway between them. With a normalized or standardized dataset, these hyperplanes can be described by the equations:

- $\vec{w}_i \cdot \vec{x}_i - b = 1$, that is anything on or above this boundary is of one class, with label 1;
- and, $\vec{w}_i \cdot \vec{x}_i - b = -1$, that is anything on or above this boundary is of one class, with label -1.

We can notice also that the distance between these two hyperplanes is $\frac{2}{\|\vec{w}\|}$ so to maximize the distance between the planes we want to minimize $\|\vec{w}\|$. The distance is computed using the distance from a point to a plane equation. We also have to prevent data points from falling into the margin, we add the following constraint: for each i:

- either, $\vec{w} \cdot \vec{x}_i - b \geq 1$, $y_i = 1$;
- or, $\vec{w} \cdot \vec{x}_i - b \leq -1$, if $y_i = -1$.

These constraints state that each data point must lie on the correct side of the margin. Finally, we collect all the previous observations and define the following optimization problem:

- from $y_i(\vec{w}_i \cdot \vec{x}_i - b) \geq 1$, for all $1 \leq i \leq n$;
- to minimize $y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1$ $i = 1, \ldots, n$.

The classifier we obtain is made from $\vec{w}$ and b that solve this problem, and he max-margin hyperplane is completely determined by those $\vec{x}_i$ that lie nearest to it. These $\vec{x}_i$ are called *support vectors*.

The other case, so the ***Soft-Margin*** case, convercely happens when the training data is not linearly separable. To deal with such situation, as well as, to extend SVM to cases in which the data are not linearly separable, we introduce the hinge loss function, that is: $max(y_i(\vec{w}_i \cdot \vec{x}_i - b))$. Once we have provided the new loss function we go ahead with the new optimization problem that we aim at minimizing that is:

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(\vec{w} \cdot \vec{x}_i - b)\right)\right] + \lambda\|\vec{w}\|^2, \tag{3}$$

where the parameter λ determines the trade-off between increasing the margin size and ensuring that the $\vec{x}_i$ lie on the correct side of the margin. Thus, for sufficiently small values of λ , the second term in the loss function will become negligible, hence, it will behave similar to the hard-margin SVM, if the input data are linearly classifiable, but will still learn if a classification rule is viable or not.

What we notice from last equations written just above is that we deal with a quadratic programming problem, and its solution is provided, detailed below.

We start defining a *Primal Problem* as follow:

- For each $\{1, \ldots, n\}$ we introduce a variable $\zeta_i = \max\left(0, 1 - y_i(w \cdot x_i - b)\right)$. Note that ζ_i is the smallest non-negative number satisfying $y_i(w \cdot x_i - b) \geq 1 - \zeta_i$;
- we can rewrite the optimization problem as follows: minimize $\frac{1}{n}\sum_{i=1}^{n}\zeta_i + \lambda\|w\|^2$, subject to $y_i(w \cdot x_i - b) \geq 1 - \zeta_i$ and $\zeta_i \geq 0$, for all i.

However, by solving for the *Lagrangian dual* of the above problem, one obtains the simplified

problem:

$$\text{maximize } f(c_1 \dots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i (x_i \cdot x_j) y_j c_j, \tag{4}$$

$$\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i. \tag{5}$$

Moreover, the variables c_i are defined as $\vec{w} = \sum_{i=1}^{n} c_i y_i \vec{x}_i$. Where, $c_i = 0$ exactly when $\vec{x}_i$ lies on the correct side of the margin, and $0 < c_i < (2n\lambda)^{-1}$ when $\vec{x}_i$ lies on the margin's boundary. It follows that $\vec{w}$ can be written as a linear combination of the support vectors.
The offset, b, can be recovered by finding an $\vec{x}_i$ on the margin's boundary and solving: $y_i(\vec{w} \cdot \vec{x}_i - b) = 1 \iff b = \vec{w} \cdot \vec{x}_i - y_i$.
This is called the *dual problem*. Since the dual maximization problem is a quadratic function of the c_i subject to linear constraints, it is efficiently solvable by quadratic programming algorithms.
Lastly, I will discuss what in the context of SVMs classifier is called as ***Kernel Trick***. Roughly speaking, we know that a possible way of dealing with datasets that are not linearly separable but that can become linearnly separable within an higher dimensional space, or feature space, we can try to remap the original data points into a higher order feature space by means of some kind of remapping function, hence, solve the SVMs classifier optimization problem to find out a linear classifier in that new larger feature space. Then, we project back to the original feature space the solution we have found, reminding that in the hold feature space the decision boundaries founded will be non-linear, but still allow to classify new examples.
Usually, especially, dealing with large datasets or with datasets with large set of features this approach becomes computationally intensive and and unfeasible if we run out of memory. So, in other words, the procedure is constrained in time and space, and might become time consuming or even unfeasible because of the large amount of memory we have to exploit.
An reasonable alternative is represented by the usage of kernel functions that are function which satisfy $k(\vec{x}_i, \vec{x}_j) = \varphi(\vec{x}_i) \cdot \varphi(\vec{x}_j)$, where we recall that classification vector $\vec{w}$ in the transformed space satisfies $\vec{w} = \sum_{i=1}^{n} c_i y_i \varphi(\vec{x}_i)$, where, the c_i are obtained by solving the optimization problem:

$$\begin{aligned}\text{maximize } f(c_1 \dots c_n) &= \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i (\varphi(\vec{x}_i) \cdot \varphi(\vec{x}_j)) y_j c_j \\ &= \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i k(\vec{x}_i, \vec{x}_j) y_j c_j\end{aligned}$$

$$\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i.$$

The coefficients c_i can be solved for using quadratic programming, and we can find some index i such that $0 < c_i < (2n\lambda)^{-1}$, so that $\varphi(\vec{x}_i)$ lies on the boundary of the margin in the transformed space, and then solve, by substituting doto product between remapped data points with kernel function applied upon the same arguments:

$$b = \vec{w} \cdot \varphi(\vec{x}_i) - y_i = \left[\sum_{j=1}^{n} c_j y_j \varphi(\vec{x}_j) \cdot \varphi(\vec{x}_i) \right] - y_i$$
$$= \left[\sum_{j=1}^{n} c_j y_j k(\vec{x}_j, \vec{x}_i) \right] - y_i.$$

Finally, $\vec{z} \mapsto \operatorname{sgn}(\vec{w} \cdot \varphi(\vec{z}) - b) = \operatorname{sgn}\left(\left[\sum_{i=1}^{n} c_i y_i k(\vec{x}_i, \vec{z}) \right] - b \right)$.

What follows is a briefly list of the most commonly used kernel functions. They should be fine tuned, by means of a either grid search or random search approaches, identifying the best set of values to replace whitin the picked kernel function, where the choice depend on the dataset at hand:

- Polynomial (homogeneous): $k(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j)^d$.
- Polynomial (inhomogeneous): $k(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j + 1)^d$.
- Gaussian radial basis function: $\gamma = 1/(2\sigma^2)\gamma = 1/(2\sigma^2)$.
- Hyperbolic tangent: $k(\vec{x}_i, \vec{x}_j) = \tanh(\kappa \vec{x}_i \cdot \vec{x}_j + c)$ for some (not every) $\kappa > 0$ and $c < 0$.

What follows is the application or use of SVMs classifier for learning a model that best fit the training data in order to be able to classify new instance in a reliable way, selecting the most promising model trained.

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

```
None
```

[2]:
```
# 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)
```

[3]:
```
columns_2_avoid = ['ERECTED', 'LENGTH', 'LOCATION']
```

[4]:
```
# 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)
```

[5]:
```
# 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 = []
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':(50, 100, 200, 500, 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)
# 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]
```

[6]:
```
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',
```

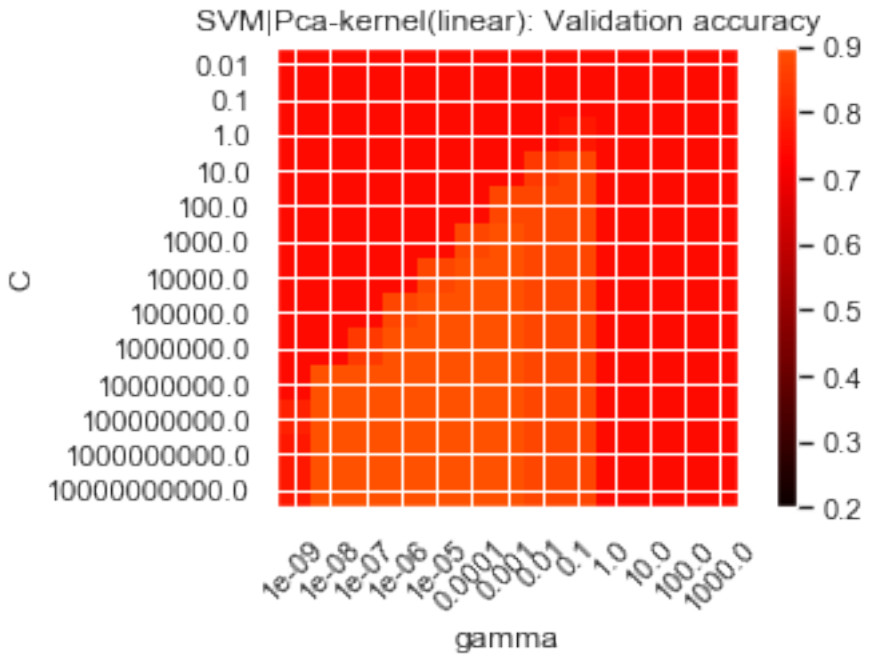
```
    figs_dest=os.path.join('figures', 'learning_curve',␣
  ↪f"Pcs_{n_components}"), ignore_func=True,
    # figsize=(20,5)
)
```

[7]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

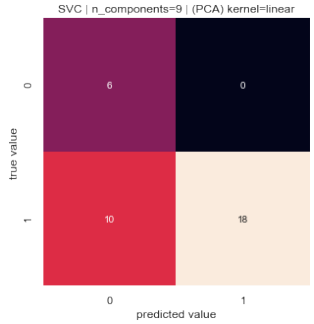
```
<IPython.core.display.Javascript object>
```

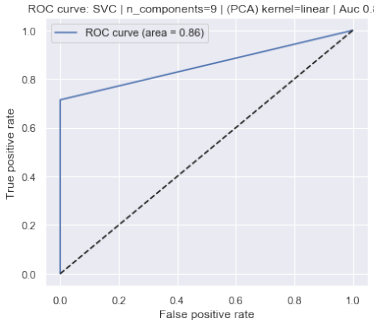
[8]:
```
plot_dest = os.path.join("figures", "n_comp_9_analysis", "grid_search")
X = rescaledX

df_gs, df_auc_gs, df_pvalue = grid_search_all_by_n_components(
    estimators_list=estimators_list[4], \
    param_grids=param_grids[0],
    estimators_names=estimators_names[4], \
    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
```

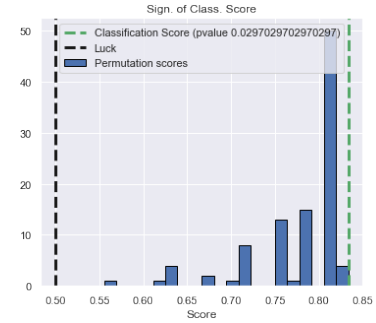
```
Kernel PCA: Linear | SVC
=========================================================
              precision    recall  f1-score   support

     class 0       0.38      1.00      0.55         6
     class 1       1.00      0.64      0.78        28

    accuracy                           0.71        34
   macro avg       0.69      0.82      0.66        34
weighted avg       0.89      0.71      0.74        34

The best parameters are {'C': 1000.0, 'gamma': 0.001} with a score of 0.90
```



Best Score (CV-Train)	Best Score (Test)	AUC	P-value
0.89	0.71	0.86	0.02970



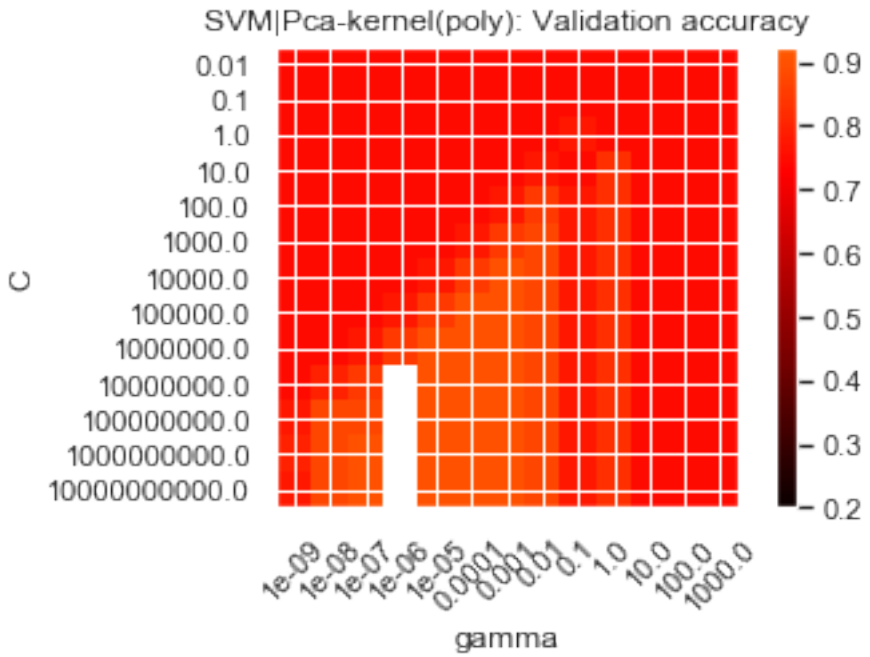




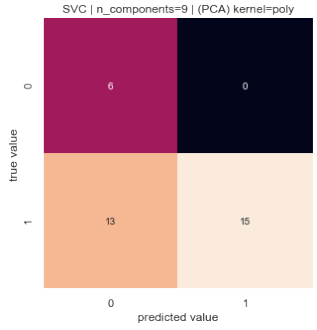
Kernel PCA: Poly | SVC

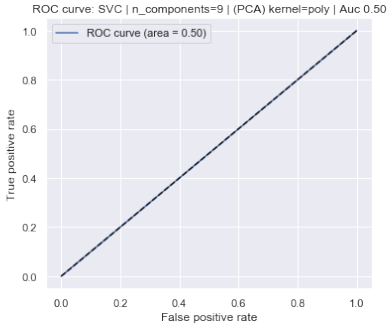
```
=======================================================
              precision    recall  f1-score   support

     class 0       0.32      1.00      0.48         6
     class 1       1.00      0.54      0.70        28

    accuracy                           0.62        34
   macro avg       0.66      0.77      0.59        34
weighted avg       0.88      0.62      0.66        34
```

The best parameters are {'C': 10000000.0, 'gamma': 1e-06} with a score of 0.93



Best Score (CV-Train)	Best Score (Test)	AUC	P-value
0.92	0.62	0.50	0.06931

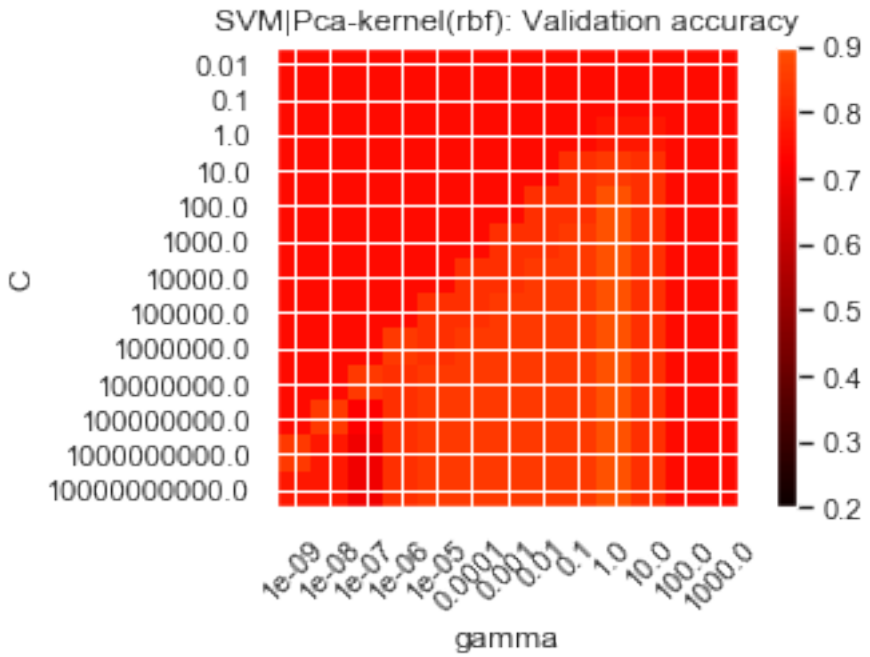




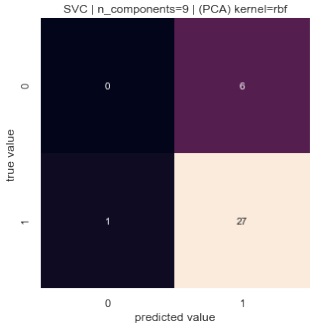
Kernel PCA: Rbf | SVC

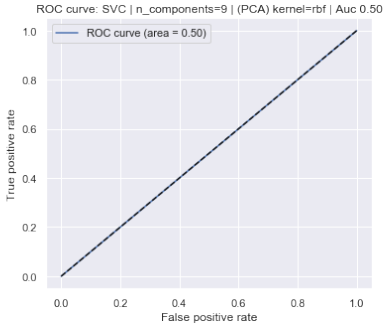
```
=======================================================
              precision    recall  f1-score   support

     class 0       0.00      0.00      0.00         6
     class 1       0.82      0.96      0.89        28

    accuracy                           0.79        34
   macro avg       0.41      0.48      0.44        34
weighted avg       0.67      0.79      0.73        34
```

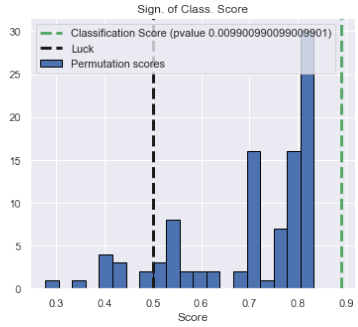
The best parameters are {'C': 100.0, 'gamma': 1.0} with a score of 0.90



Best Score (CV-Train)	Best Score (Test)	AUC	P-value
0.92	0.79	0.50	0.00990



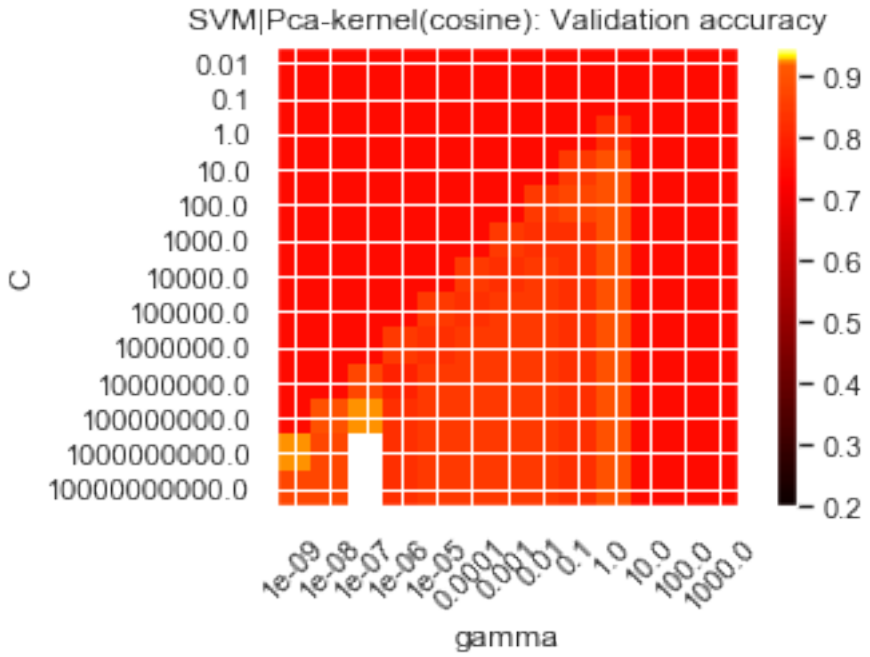




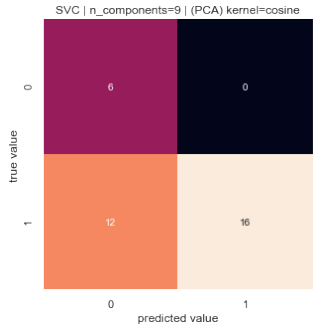
Kernel PCA: Cosine | SVC

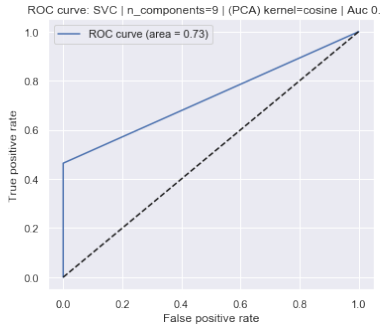
```
=======================================================
              precision    recall  f1-score   support

     class 0       0.33      1.00      0.50         6
     class 1       1.00      0.57      0.73        28

    accuracy                           0.65        34
   macro avg       0.67      0.79      0.61        34
weighted avg       0.88      0.65      0.69        34
```

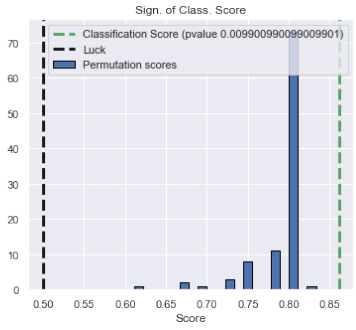
```
The best parameters are {'C': 1000000000.0, 'gamma': 1e-07} with a score of 0.
↪95
```



Best Score (CV-Train)	Best Score (Test)	AUC	P-value
0.92	0.65	0.73	0.00990

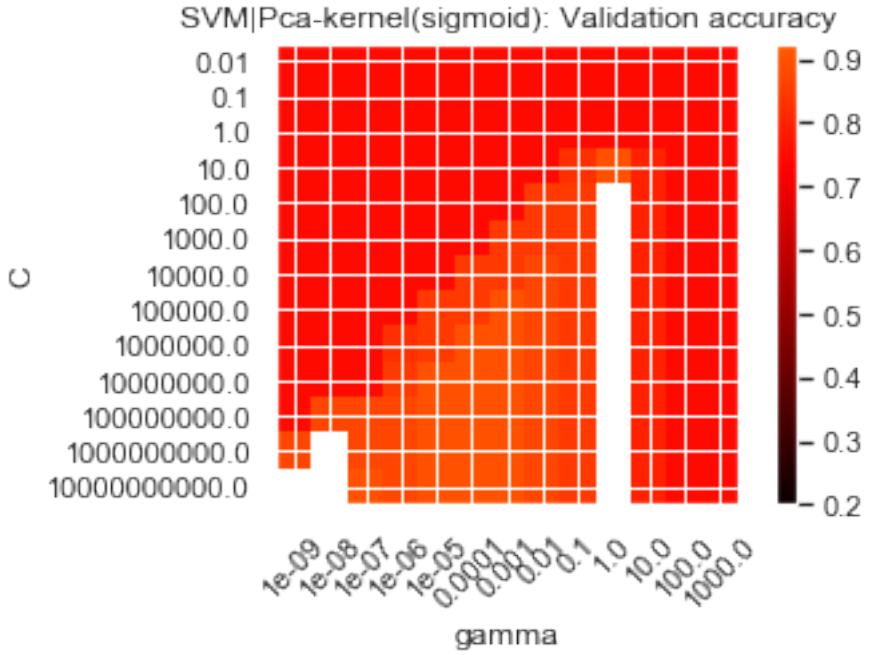




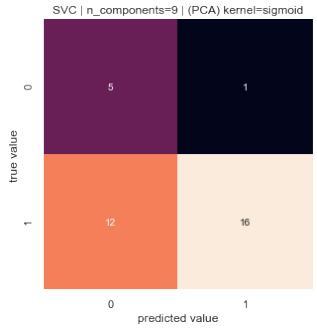


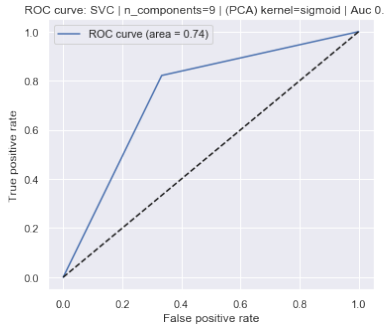

```
Kernel PCA: Sigmoid | SVC
=========================================================
              precision    recall  f1-score   support

     class 0       0.29      0.83      0.43         6
     class 1       0.94      0.57      0.71        28

    accuracy                           0.62        34
   macro avg       0.62      0.70      0.57        34
weighted avg       0.83      0.62      0.66        34
```

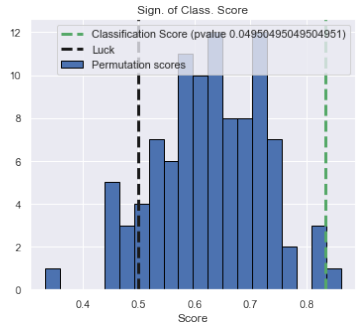
```
The best parameters are {'C': 100.0, 'gamma': 1.0} with a score of 0.93
```



Best Score (CV-Train)	Best Score (Test)	AUC	P-value
0.95	0.62	0.74	0.04950







Looking at the results obtained running *Svm 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:

1. Speaking about **Linear kernel Pca based Svm 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 *89%* at train step, while the Auc score reaches a value of *86%* with a Roc Curve that shows a behavior for which the model for a wide range of possible thresholds lower than .5 seems to improve over *TPR* quicklier than *FPR*, unitl the Roc Curve turns so that also *FPR* increases linearly with *TPR* metric. The resulting model shows both high precision and high recall when considering the performance of such metrics related to class 1, while reflecting upon class 0 performance metrics we notice as for other previous models that generally the several classifiers and estimators tend to show high recall, so correctly predict labels for true class 0 examples, but relatively low precision due to the fact that since the dataset is unbalanced on average one third of the true class 1 samples are misclassified and such a number is comparable with the amount of correctly classified true class 0 examples. However, such a model do not overfit too much against train set, since we can see that on test set has obtained an accuracy of just 10 percent point less than the train accuracy score.
2. Observing **Polynomial kernel Pca based Svm Estimator**, we can notice that such a model exploiting a default threshold of *.5* reaches an accuracy of *62%* at test time against an accuracy of *92%* at train step, while the Auc score reaches a value of *50%*. This model as we can simply understand is a trial in which the final hyper-params configuration leads to a model that in practice when adopted for inference tasks tends to predict correctly all samples from class 0, however mispredict most of samples from class 1, so this witness leads us saying that the model is characterized from high recall but low precision for class 0, and high precision but a middle recall for class 1. In other words the model has high uncertainty when attemptin as class 0 a given unknown data point. For sure we should discard such a classifier and to get rid of the combining gien from polynomial kernel-trick for performing kernel-Pca when prep-processing data examples before carrying out training step.
3. Looking at **Rbf kernel Pca based Svm 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 *92%* at train step, while the Auc score reaches a value of *50%*. This trial together with *Polynomial kernel Pca based Svm Estimator*, leads to the worst models we have retained from grid-search based training step performed for SVMs based learning. In particular, this model as we can clearly understand is a trial in which the final hyper-params configuration leads to a model that in practice when adopted for inference tasks tends to predict as sections/svm-classifier/output class label the class 1 regardless whether a sample belongs indeed to such a category. So, in other words, the model overfit to the training set and all the examples from class 0 seem to be extremely different from those within the training set and thus were all mispredicted. Iven if the model has high recall related to class 0 and high preicision related to class 1, in thruth the model does not go better than the random classifier, in fact as we can grasp from the Roc Curve behaviour neither modifining the default threshold values we are able to improve the performance metrics scores. For sure we should discard such a classifier and to get rid of the combining gien from rbf kernel-trick for performing kernel-Pca when

prep-processing data examples before carrying out training step.

4. looking at **Cosine kernel Pca based Svm Classifier**, 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 *73%*. The fine-tuned model which has been trained with data examples pre-processed by means of kernel-Pca exploiting cosine kernel, has obtained a much lower accuracy at test time than the one measured at training step, this means that the model overfits at least a little bit, and the model is characterized by means of somewhat high AUC score, which means that such a estimator might be set using a different discriminative threshold in place of the default one. However the Roc Curve trend firstly shows a TPR that grows without increasing FPR, then at a given point also the FPR begins to increase and the two scores increase simultaneosly with a linear relationship. In particular, at default threshold the model is higly precise and has a middle recall value for class 1, instead has low precision and high recall for class 0, so also that model tends to predict class 0 label with higher uncertainty.
5. finally, referring to **Sigmoid kernel Pca based Svm Model**, we can notice that such a model exploiting a default threshold of *.5* reaches an accuracy of *62%* at test time against an accuracy of *95%* at train step, while the Auc score reaches a value of *74%*. Such a model has a behavior that is somewhat quit similar to that of the best classifier found by means of grid-search approach related to *Cosine kernel Pca based Svm Classifier example*. However has a slighlty lower recall for class 0 than the previous model cited just above, but has a higher AUC Score, which looking at the Roc Curve trend we know that such a model allows for a larger set of thresholds which correspond to TPR and FPR values where the former is higher than the latter before the second turns to grow quicklier.

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, 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. While exploiting SVMs classifier as learning technique, we have obtained up to four models out of five which allow us to reject the *Null-Hypothesis*, so refusing to reject such a hypothesis leads us to a *Type I Error*, in particular these models are *Linear, Rbf, Cosine and Sigmoid kernel Pca based Svm Model*. This means that just a model, which is *Polynomial kernel Pca based Svm Model* leads to a model for which we are suggested not to reject the Null-Hypothesis, so that if we reject such a hypothesis we are leaded to a *Type II Error*. In other words we should exploit all the previous classifiers for inference tasks except for *Polynomial kernel Pca based Svm Model*, but since exploiting p-value analysis is just a step of investingating classifier performance, indeed just *Linear and Cosine kernel Pca based Svm Models* seem to be the models to be deployed for real applications.

Table Fine Tuned Hyper-Params(Knn Classifier)

[9]:
```
# 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)
```

[9]:
```
            AUC(%) P-Value(%) Acc Train(%) Acc Test(%)   C class_weight \
```

SVC linear	**0.86**	**2.97**	**0.89**	**0.71***	**0.0001**	**None**
SVC poly	0.50	6.93	0.92	0.62	0.01	balanced
SVC rbf	0.50	0.99	0.92	0.79	0.0001	None
SVC cosine	0.73	0.99	0.92	0.65	0.0001	None
SVC sigmoid	0.74	4.95	0.95	0.62	1000.0	None

	coef0	***degree***	***gamma***	***kernel***	***max_iter***	***probability***	***shrinking***
SVC linear	**0.001**	**1**	**1000.0**	**poly**	**50**	**True**	**True**
SVC poly	1.0	4	1.0	poly	50	True	True
SVC rbf	0.001	8	10	poly	50	True	True
SVC cosine	10.0	2	100.0	poly	50	True	True
SVC sigmoid	0.1	1	10	sigmoid	50	True	True

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 *Linear, Polynomial, and Sigmoid Tricks* or less overfit solution such as in the case of *Cosine, Trick*. Speaking about the hyper-parameters, we can say what follows:

- Looking at svm's **C Parameter**, and suggesting that such a parameter briefly trades off correct classification of training examples against maximization of the decision function's margin. For larger values of C, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. So that, a lower C will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy. In other words *C* behaves as a regularization parameter in the SVM. What we can say about such a parameter is that three out of five classifiers set a very low value for *C*, that is *1e-4*, so they allow for margins that let samples to cross them often, however those classifier perform well at test time. Just Sigmoid-trick based model set a very large *C* parameter so that it built a decision rule that was really strict when classifying samples, but such a model perfomrs somewhat bad at test time. Lastly looking at Poly-trick based model was the model whcih adopted a C value with a somewhat middle value, of *0.01*, that however does not let such a classifier to obtain important results at test time. Latly the choice of a kernel-trick affected importantly the resultin value of *C parameter*.
- 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)). Most of the models chosen a strategy for which the differnt data examples belonging to some class are uniformely weighted , while just Poly-trick based model adopted a strategy or policy for which the diverse examples

are weighted accordingly to the class membership, however, this was also one of the classifier which does not performs somewhat well, so if we exclude such a classifier, then we can state that the selection of a particular kernel-trick for kernel-Pca method does not affect such widely a hyper-parameter.

- Analysing **coef0 hyper-param**, which represents independent term in kernel function. It is mostly significant in *'poly' and 'sigmoid' based classifiers, we can observe that the choice of the right kernel-trick for pre-processing the data point affect widely the subsequent choice for such a parameter, in fact we can observe that the* coef0 param's values* range within a virtual interval that goes from 1e-3, up to 10. However Rbf and Linear-trick based models adopted a somewhat low coefficient value, while Poly, Cosine and Sigmoid-tricks base models adopted a larger coefficient, so that its contribute is more significant and important, so acquires weight at inference time.
- Instead, when speaking about svm's **Degree Parameter**, which is degree of the *polynomial kernel function (poly)*, therefore ignored by all other kernels, we are talking about a niche hyper-params that usually taking larger values implyes attempting to fit models that tends to lower bias values and higher variance, instead taking into account lower polynomial order my bit on higher bias and lower variance models. In other words, in one case we attempt to explain with a more fancy model a problem that might not follow a linear and smooth trend, vice versa, in the second case we prefer to represent our model by means of a lower capaciy classifier, easier to explain and understand. Poly and Rbf-trikcs based models was the classifier which most have been characterized by more complex learners since the degree of the polynomial expression of the classifier has higher degree than the others so that higher variance and lower bias affect such models. Viceversa, the remaining models preferred to adopt a lower polynomial expression so that the resulting models are characterized from lower variance but higher bias.
- Referring to the svm's **Gamma Parameter**, which we recall that corresponds normally to *Kernel Coefficient* for *rbf, poly and sigmoid reprojection strategies*, we also remind that intuitively, the gamma parameter defines how far the influence of a single training example reaches, with *low values* meaning *far* and *high values* meaning *close*. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. Shortly we observe that Linear and Cosine-tricks based models required a larger value of gamma so that for building the classifier and in particular the margins was taken into account a reduced number of samples the closest data points for deriving the margins, instead the poly-trick based model was the one which amongst the others allows for taking into account also much farer data points to be employed in oreder to identify the support verctors for the margins. Lastly the two remaining classifiers, Rbf and Sigmoid-tricks based models adopted a middle value for gamma parameter, which means that the final learners are in the between in terms of memory size requested to store and use for inference tasks.
- The svm's **Shrinking Parameter**, from research field we found that if the number of iterations is large, then shrinking can shorten the training time. This hyper-param was adopted from all the classifier we have found after having carryed out grid-search based training step. So such a parameter indirectly affected also the other one represented by *max_iter*, in fact we have noticed that in all cases the training phase does not last too much also because of the reduced or small size of our dataset as well as the fact

that small datasets tends to overfit quicklier so that halting the training procedure near the beginning of the process might lead to classifiers which less overfit and are able to generalize better. In fact, for each classifier was sufficient to set a max iteration numeber equals to 50 rounds.

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 *SVMs classifier*, for sure, we could employ the classifier found from all the trials, except for *Poly and Rbf kernel Pca based Sgd Classifiers*, since the first model is overly overfitting to the data used at train time and more precisely most of the time predicted correctly just samples from class 1 and misclassifyes instances from class 0, the others instead assumed the opposite behavior. Also, 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.

7.1.1 Support Vector Machines References

- (Discriminative Model: SVM) https://scikit-learn.org/stable/modules/svm.html

8 Decision Trees

8.1 Decision Trees Properties

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*

Decision Trees, for short DTs, are a *non-parametric supervised learning method* used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.
Their mathematical formulation is generally provided as follows: Given training vectors $x_i \in R^n$, $i = 1, \cdots, l$ and a label vector $y \in R^l$, a decision tree recursively partitions the space such that the samples with the same labels are grouped together.
Let the data at node m be represented by Q. For each candidate split $\theta = (j, t_m)$ consisting of a feature j and threshold t_m, partition the data into $Q_{left}(\theta)$ and $Q_{right}(\theta)$ subsets as:

$$\begin{aligned} Q_{left}(\theta) &= (x, y)|x_j <= t_m \\ Q_{right}(\theta) &= Q \setminus Q_{left}(\theta) \end{aligned} \tag{6}$$

The impurity at m is computed using an impurity function $H()$, the choice of which depends on the task being solved (classification or regression) like:

$$G(Q,\theta) = \frac{n_{left}}{N_m} H(Q_{left}(\theta)) + \frac{n_{right}}{N_m} H(Q_{right}(\theta)) \tag{7}$$

Select the parameters that minimises the impurity: $\theta^* = \text{argmin}_\theta \, G(Q,\theta)$.
Recurse for subsets $Q_{left}(\theta^*)$ and $Q_{right}(\theta^*)$ until the maximum allowable depth is reached, $N_m < \min_{samples}$ or N_m = 1.
Speaking about *Classification Criteria* referred to the procedure used for learining or fit to the data a decision tree we can state what follows: If a target is a classification outcome taking on values $0, 1, \cdots, K-1$, for node m, representing a region R_m with N_m observations, let $p_{mk} = 1/N_m \sum_{x_i \in R_m} I(y_i = k)$ be the proportion of class k observations in node m.
So, Common measures of impurity are:

- Gini, specified as $H(X_m) = \sum_k p_{mk}(1 - p_{mk})$, Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. Useful features of the Gini coefficient include:
  - it favours large partitions, and it is simple to understand
  - it is associated a maximum value of impurity when it assume the value of *.5* for binary classification problems, and $1 - \frac{1}{no.of classes}$ when dealing with multicategory problems. While it is associated with a minimum value of impurity when it is equal to 0.
  - Some **bacwards** instead are, since it tends to produce few large partitions it leads to higher bias and lower variance, especially when dealing with small datasets.
- Entropy, definead as $(X_m) = -\sum_k p_{mk} \log(p_{mk})$, which is a basic quantity in information theory associated to any random variable, which can be interpreted as the *average level of "information", "surprise", or, "uncertainty"* inherent in the variable's possible outcomes. To understand the meaning of $(X_m) = -\sum_k p_{mk} \log(p_{mk})$, first define an information function I in terms of an event i with probability p_i. The amount of information acquired due to the observation of event i follows from Shannon's solution of the fundamental properties of information:
  - $I(p)$ is monotonically decreasing in p : an increase in the probability of an event decreases the information from an observed event, and vice versa.
  - $I(p) \geq 0$: information is a non-negative quantity.
  - $I(1) = 0$: events that always occur do not communicate information, where, it corresponds to the minimum value of impurity, so we seek for those features that allows for small value of *Information Gain*. On the other hand, a maximum value of impurity corresponds to a Information Gain that is equal to 1.

We recall that X_m is the training data in node m

The main behaviors of both *Gini Index, and Information Gain implemented by means of Entropy* are summarized in the following table:

Algo / Split Criterion	Description	Tree Type
Gini Split / Gini Index	*Favors large partitions. Very simple to implement*	*CART*
Information Gain / Entropy	*Favors paritionts that have small counts but many distinct values.*	*ID3 / C4.5*

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

```
None
```

[2]:
```
# 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)
```

[3]:
```
columns_2_avoid = ['ERECTED', 'LENGTH', 'LOCATION']
```

[4]:
```
# 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)
```

[5]:
```
# 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_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)
# 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]
```

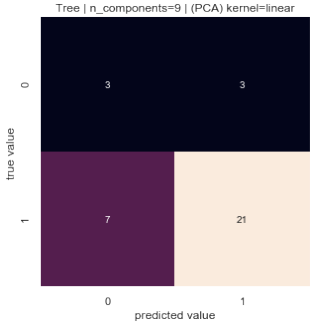
[6]:
```
n_components=9
learning_curves_by_kernels(
# learning_curves_by_components(
    estimators_list[:], estimators_names[:],
    rescaledX, y,
    train_sizes=np.linspace(.1, 1.0, 10),
    n_components=9,
    pca_kernels_list=pca_kernels_list[0],
    verbose=0,
    by_pairs=True,
    savefigs=True,
    scoring='accuracy',
    figs_dest=os.path.join('figures', 'learning_curve',␣
 ↪f"Pcs_{n_components}"), ignore_func=True,
    # figsize=(20,5)
)
```

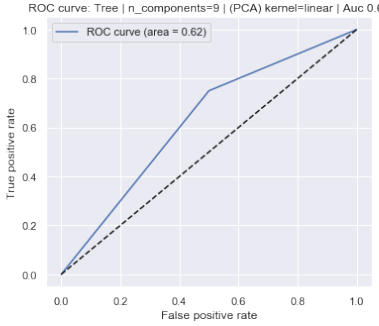
[7]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

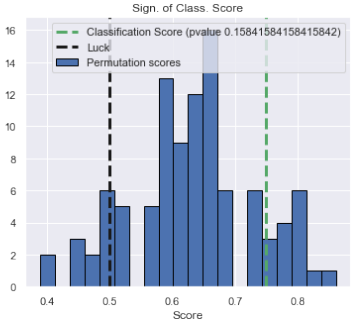
```
<IPython.core.display.Javascript object>
```

[8]:
```
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[0],
    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
```



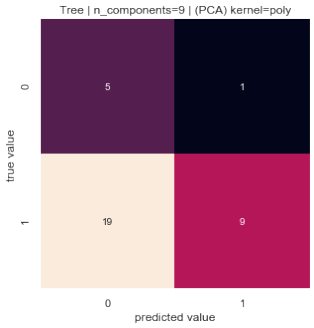


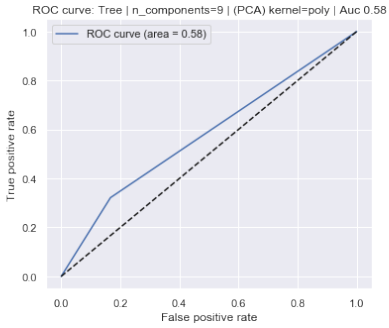


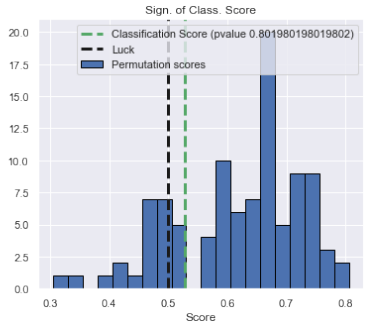

```
Kernel PCA: Poly | Tree
=========================================================
              precision    recall  f1-score   support

     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
```





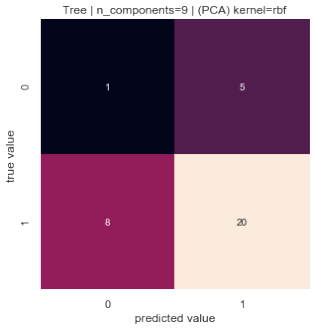


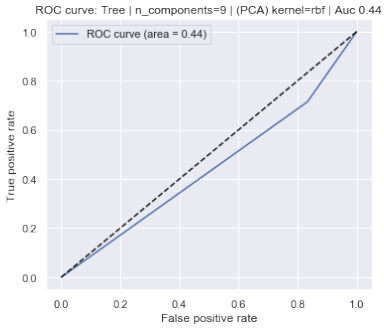

```
Kernel PCA: Rbf | Tree
```

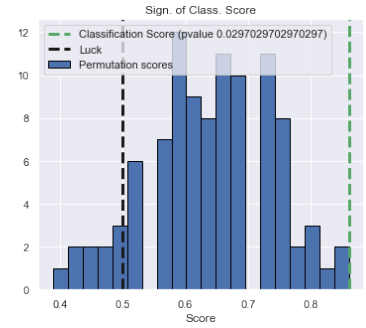
```
=========================================================
```

```
              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
```







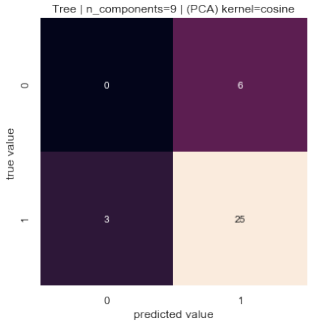

```
Kernel PCA: Cosine | Tree
```

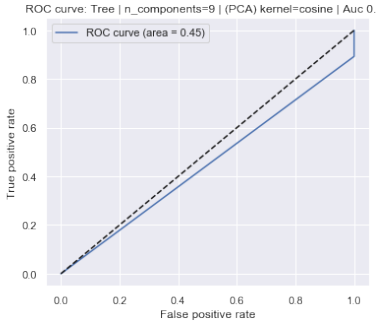
```
=========================================================
```

```
              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
```

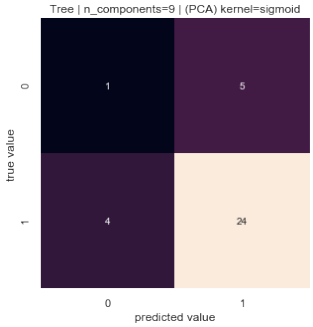
```
Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.86              0.74  0.45  0.04950
```

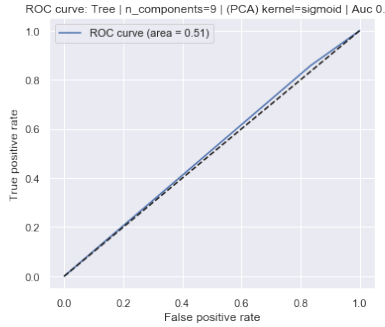


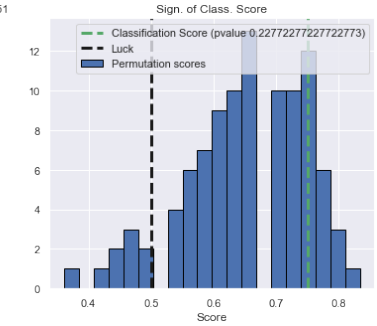



```
Kernel PCA: Sigmoid | Tree
=========================================================
              precision    recall  f1-score   support

     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:

1. 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.
2. 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.
3. 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.
4. 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.
5. Lastly the **Sigmoid kernel-Pca based Decision Tree Classifier**, with a default thresh-

old 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)

[9]:
```
# 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)
```

[9]:

	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.

8.1.1 Decision Trees Classifiers References

- (**Ensemble, Non-Parametric Learning: Decision Trees**) https://scikit-learn.org/stable/modules/tree.html#tree
- (**Gini Index**) https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifi
- (**Entropy**) https://en.wikipedia.org/wiki/Entropy_(information_theory)

9 Ensemble methods

9.1 Ensemble methods

The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single

estimator.
Two families of ensemble methods are usually distinguished:

- In *Averaging Methods*, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced. So, some examples are: *Bagging methods, Forests of randomized trees*, but still exist more classifiers;
- Instead, in *Boosting Methods*, base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble. Hence, some examples are: *AdaBoost, Gradient* Tree Boosting,but still exist more options.

9.2 Random Forests Properties

Learning Technique	Type of Learner	Type of Learning	Classification	Regression	Ensemble Family
RandomForest	*Ensemble Method (Meta-Estimator)*	*Supervised Learning*	*Supported*	*Supported*	*Averaging Methods*

The **sklearn.ensemble module** includes two averaging algorithms based on randomized decision trees: the RandomForest algorithm and the Extra-Trees method. Both algorithms are ***perturb-and-combine techniques***, specifically designed for trees. This means a diverse set of classifiers is created by introducing randomness in the classifier construction. The prediction of the ensemble is given as the averaged prediction of the individual classifiers.
In random forests (see RandomForestClassifier and RandomForestRegressor classes), each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set.
The main parameters to adjust when using these methods is *number of estimators* and *maxima features*. The former is the number of trees in the forest. The larger the better, but also the longer it will take to compute. In addition, note that results will stop getting significantly better beyond a critical number of trees. The latter is the size of the random subsets of features to consider when splitting a node. The lower the greater the reduction of variance, but also the greater the increase in bias.
Empirical good default values are maxima features equals to null, that means always considering all features instead of a random subset, for regression problems, and maxima features equals to “sqrt”, using a random subset of size sqrt(number of features)) for classification tasks, where number of features is the number of features in the data. The best parameter values should always be cross-validated.
We note that the size of the model with the default parameters is $O(M * N * log(N))$, where M is the number of trees and N is the number of samples.

[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)
```

[6]:
```
# 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_random_forest = {
    'n_estimators': list(range(2, 10)),
    'criterion':('gini', 'entropy'),
    'bootstrap': (True, False),
    'min_samples_leaf': list(range(1,5)),
    '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]
```

[7]:
```
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)
)
```

[8]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

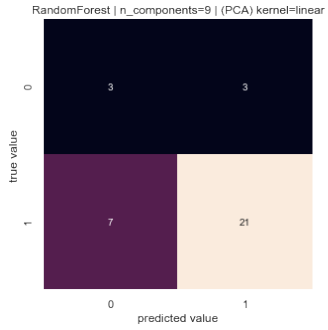
```
<IPython.core.display.Javascript object>
```

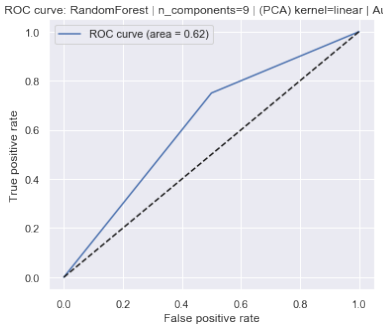
Learning Technique	Type of Learner	Type of Learning	Classification	Regression	Ensemble Family
RandomForest	*Ensemble Method (Meta-Estimator)*	*Supervised Learning*	*Supported*	*Supported*	*Averaging Methods*

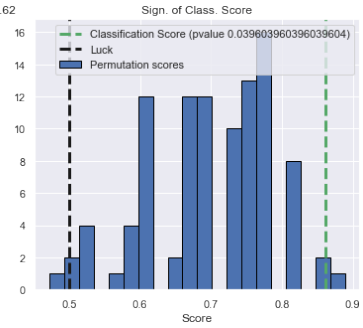
[9]:
```
plot_dest = os.path.join("figures", "n_comp_9_analysis", "grid_search")
X = rescaledX

df_gs, df_auc_gs, df_pvalue = grid_search_all_by_n_components(
    estimators_list=estimators_list[6], \
    param_grids=param_grids[0],
    estimators_names=estimators_names[6], \
    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
```

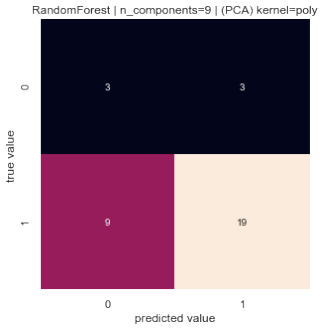
```
Kernel PCA: Linear | RandomForest
=====================================================
              precision    recall  f1-score   support

     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
                 0.89                0.71  0.62  0.03960
```

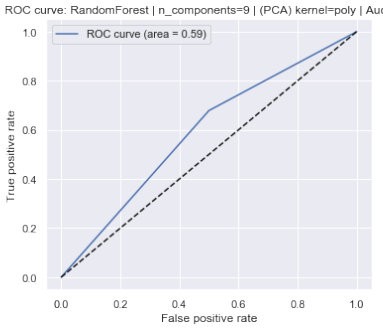







```
Kernel PCA: Poly | RandomForest
=====================================================
              precision    recall  f1-score   support

     class 0       0.25      0.50      0.33         6
     class 1       0.86      0.68      0.76        28

    accuracy                           0.65        34
   macro avg       0.56      0.59      0.55        34
weighted avg       0.76      0.65      0.68        34

Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.94                0.65  0.59  0.02970
```

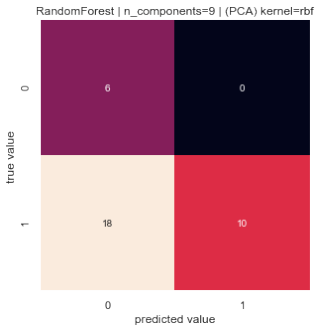


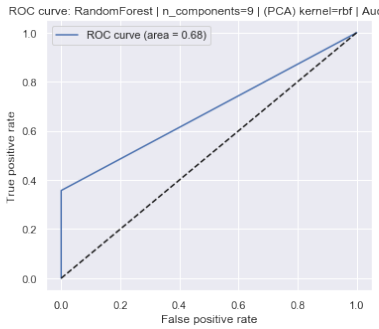


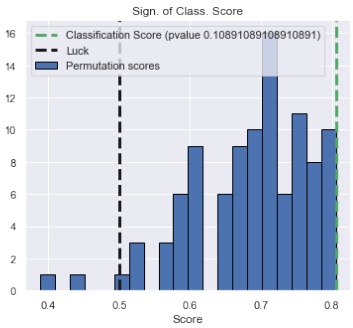
Kernel PCA: Rbf | RandomForest

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

     class 0       0.25      1.00      0.40         6
     class 1       1.00      0.36      0.53        28

    accuracy                           0.47        34
   macro avg       0.62      0.68      0.46        34
weighted avg       0.87      0.47      0.50        34
```

Best Score (CV-Train)	Best Score (Test)	AUC	P-value
0.97	0.47	0.68	0.10891



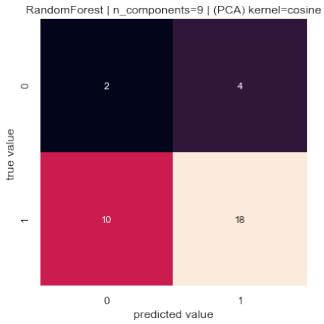


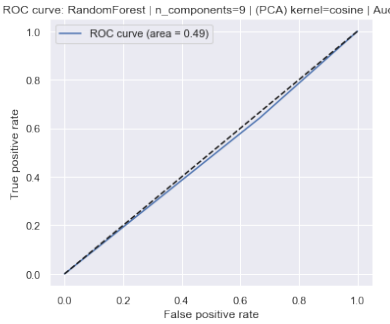


Kernel PCA: Cosine | RandomForest

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

     class 0       0.17      0.33      0.22         6
     class 1       0.82      0.64      0.72        28

    accuracy                           0.59        34
   macro avg       0.49      0.49      0.47        34
weighted avg       0.70      0.59      0.63        34
```

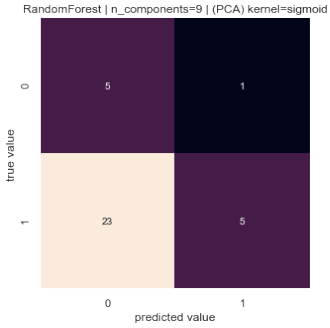
```
Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.89                0.59  0.49  0.35644
```

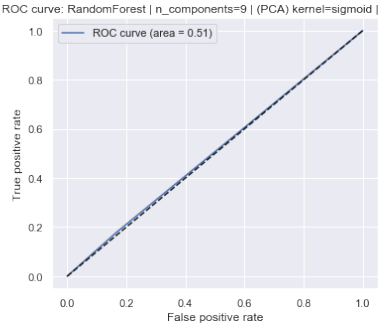


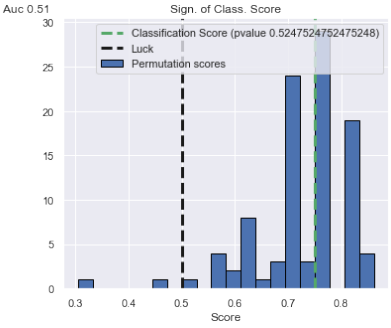



```
Kernel PCA: Sigmoid | RandomForest
=====================================================
              precision    recall  f1-score   support

     class 0       0.18      0.83      0.29         6
     class 1       0.83      0.18      0.29        28

    accuracy                           0.29        34
   macro avg       0.51      0.51      0.29        34
weighted avg       0.72      0.29      0.29        34
```

```
Best Score (CV-Train) Best Score (Test)   AUC  P-value
                 0.86                0.29  0.51  0.52475
```







Looking at the results obtained running *RandomForest 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 for which just two cases out of five are really meainingful and get important performances, that are classifiers corresponding to *Linear, Polynomial kernel Pca based RandomForest Classifiers*. While in all the remaining cases we notice that the models fine-tuned do not perform either better than random classifier or

just worst than a random classifier when adopting a default threshold. However the main description for each classifier is the following:

1. Speaking about **Linear kernel Pca based RandomForest 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 *89%* at train step, while the Auc score reaches a value of *62%* with a Roc Curve that shows a behavior for which the model obtained a Auc score value that isnot too much higher than the Auc score of a random classifier that is *.5*. However for a large set of increasing threshold the model let the TPR to grow quicklier than the FPR, but the latter still increases as well while modifying the discriminative threshold. The model, generally speaking has high recall and precision related to class 1, even if we have to note that our model is a unbalanced model with respect to at least targe class values. While, analysing the performance of class 0, we can observe that the model is characterized by .5 recall and very low precision, so as we already knwo from previous results, the model works hard to classify with low uncertaininty samples when predicting as target label class 0.
2. Observing **Polynomial kernel Pca based RandomForest Estimator**: 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 *94%* at train step, while the Auc score reaches a value of *59%*. With such a model obtained as the best model after having carryed out grid-search phase for training step while adopting as kernel-trick a polynomial feature for kernel-Pca unsupervised technique, we can simply saying that the resulting model has performance that is slighlty lower than the preciding one, however looking at the difference between train and test accuracy we can rougly say that the model seems to overfit against train data, so that it get worst performance metrics scores. Fuurthermore, the Auc score, due to overfit do not reach an appreciable value, in fact it is just more or less 10 percentage point above that of random classifier. In other words the Roc Curve is somewhat similar to that of the previou model expect that appears a little bit more flattened or approaching the Roc Curve related to Random Classifier.
3. Review **Rbf kernel Pca based RandomForest Classifier**: we can notice that such a model exploiting a default threshold of *.5* reaches an accuracy of *47%* at test time against an accuracy of *97%* at train step, while the Auc score reaches a value of *68%*. The model we have obtained here for such a trial, adopting Rbf-Trick for kernel-Pca unsupervised learning technique shows a fine-tuned classifier that largely overfit to the train data, because the test accuracy score is even worst than the random classifier, beacues the model misclassifyes the most of class 1 examples even if goes well with those samples that indeed belong to class 0. So even if the model is characterized from very high precision of class 1 and recall of class 0, and due to the large number of samples that belongs to class 1 but were classified as class 0, it does not gain satisfactory performance, suggesting to increase the default threshold in order to improve classification performance. In fact the related Roc Curve suggest to adopt thresholds that are higher than the default in order to let the trained estimator to misclassify a lower number of samples from class 1 at cost of wrogly predict labels of some samples from class 0.
4. Looking at **Cosine kernel Pca based RandomForest 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 *89%* at train step, while the Auc score reaches a value of

49%. This model together with the last one that is illustrated a little bit later, are the two amongst the Decision Trees based learning models which have been trained by means of training phase exploiting grid-search approach as for all other models, but lead to very poor performances that are even worst than random classifier when looking at Roc Curve graphics. In more details, the model misclassifyes most of the samples from class 0, and a large number of samples from class 1 also haven wrongly predicted in terms of class labels, so the model has both low precision and recall related directly to class0, however also analog results we have obtained for same metrics instead related to class 1.

5. Finally, referring to **Sigmoid kernel Pca based RandomForest Model**: we can notice that such a model exploiting a default threshold of *.5* reaches an accuracy of *29%* at test time against an accuracy of *89%* at train step, while the Auc score reaches a value of *29%*. As we have already said from the preiviosu analysis, this model together with *Cosine kernel Pca based RandomForest Classifier* that was illustrated just before, are the two amongst the Decision Trees based learning models which have been trained by means of training phase exploiting grid-search approach as for all other models, but lead to very poor performances that with a great effort and fatigue seem slightly better than a random classifier model. In particular, we can soon notice that this classifier correctly predicts most of samples that indeed belong to class 0, however misclassifyes the majority of samples that indeed belong to class 1, so it gained high precision and recall for class 1 and class 0, respectively, but low recall and precision for class 1 and class 0, as well. This model with a threshold value does not get better performance than a random classifier and even chaning the default threshold we do not obtain better performance, since the model tend to predict examples as class 0 most of the time.

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. As we already know from the description given for each fine-tuned classifier trained by means of Decision Trees based learning technique, we can state that only two results out of five, which are the first two classifier trained represented by *Linear and Polynomial kernel Pca based RandomForest Estimators* we have obtained important values for performance metrics in fact the corresponding p-values are lower than the fixed *significance levels*. This means that those two models are the estimators to be retained for future inferences, while the other remaing classifiers that are *Rbf, Cosine, and Sigmoid kernels Pca based RandomForest Learners* the p-value scores obtained are not significant to let us state that them are meaningful or exploitable models for further inferences against unknown examples, so rejecting Null-Hypothsi for such models will lead us to *Type II Error*, as well as, refusing to reject *Null-Hypothes* for the first two models leads to *Type I Error*.

[10]:
```
# 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)
```

[10]:

	AUC(%)	P-Value(%)	Acc Train(%)	Acc Test(%)	bootstrap
RdmFrst linear	**0.62**	**3.96**	**0.89**	**0.71***	**True**

RdmFrst poly	0.59	2.97	0.94	0.65	False
RdmFrst rbf	0.68	10.89	0.97	0.47	True
RdmFrst cosine	0.49	35.64	0.89	0.59	True
RdmFrst sigmoid	0.51	52.48	0.86	0.29	True

	class_weight	***criterion***	***max_depth***	***max_features***	***min_samples_leaf***	***n_estimators***
RdmFrst linear	**balanced**	**gini**	**None**	**sqrt**	**3**	**7**
RdmFrst poly	balanced	entropy	None	sqrt	2	2
RdmFrst rbf	balanced	gini	None	sqrt	2	6
RdmFrst cosine	balanced	gini	None	sqrt	1	6
RdmFrst sigmoid	None	gini	None	sqrt	2	4

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 *Rbf, Cosine, and Sigmoid Tricks* or less overfit solution such as in the case of *Linear, Polynomial, and Trick*. Speaking about the hyper-parameters, we can say what follows:

- Looking at **bootstrap hyper-parameter**, which aims at exploiting bootstrap samples, used when building trees. If False, the whole dataset is used to build each tree. We can clearly notice that the majority of classifiers preferred a *boostrap strategy enabled*, and only *Poly kernel Pca based RandomForest Classifier* instead decided to disable such a strategy to get greater performances. Since just two cases out of five are the most interesting and amongst these two best models the boostrap technique is not uniformly adopted we can state that the choice of a particular kernel-method for executing Kernel-Pca prep-processing affects the final setting of the resulting Random Forest classifier.
- Viewing **class_weight hyper-param**, referes to weights associated with classes and if not given, all classes are supposed to have weight one. 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)). The *"balanced_subsample"* mode is the same as "balanced" except that weights are computed based on the bootstrap sample for every tree grown. We can clearly notice that the majority of classifiers preferred a *class_weight strategy enabled*, and only *Sigmoid kernel Pca based RandomForest Classifier* instead decided to disable such a strategy to get greater performances. However, since the latter model is not of particular interest so that it can be discarded without loosing a good classifier, we can state that at the end all the models follows the same policy for weighing examples at training time.
- Reviewing **criterion parameter**, whcich represents the *function to measure the quality of a split* where supported criteria are *"gini"* for the Gini impurity and *"entropy"* for the information gain and this parameter is *tree-specific*. The analysis we are going to carry out for discussing such a hyper-param follows more or less the same knowledge we

got from viewing *bootstrap hyper-parameter*. So we can end up saying that the choice of a particular kernel-trick affected the related selection of the appropriate criterion technique, and just looking at the best model from Decision Trees based learning in half of the cases the model preferred Gini Index while in the remaining part the best performance metrics have been foound by means of entropy strategy.

- Looking at **max_depth hyper-param**, it reflects 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*. Here all the model uniformely adopted the value for such a param that corresponds to the highest degree of freedom possible for growing the several trees estimators, so that we can obtain random forest models which single trees learners might be somewhat wide and therfore the resulting models might be costly both from memory and perfomance point of views.
- Describing **max_features hyper-param**, a hyper-param as this referes to the number of features to consider when looking for the best split, and supported choicea are *"auto", "sqrt", "log2"*. If "auto", then max_features=sqrt(n_features). If "sqrt", then max_features=sqrt(n_features) (same as "auto"). If "log2", then max_features=log2(n_features). And, finally If None, then max_features=n_features. If None, then max_features=n_features. We notice that the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max_features features. Briefly, this hyper-param allows us understand that the choice of a particular kernel-trick does not affect the subsequent selection at train time of the max-number of features to adopt for build a learner tree.
- Looking at **min_samples_leaf hyper-param**, describes the minimum number of samples required to be at a leaf node. In othr words, a split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branche. In general the current choice of a kernel-trick affect at least in a certain part the number of minimum samples for allowing the tree adding a node and not to complete the branch with a leaf, in fact in the majority of the case however the value for such a param was set with a somewhat low value, meaning that this parameter will not affect importantly the depth of the trees built a traiing time. However, the most constrained classifier were the first two which also represent the best models, while the remaining were less affected by min_samples_leaf param so that they are more free of increasing trees depth.
- Looking at **n_estimators hyper-parameter**, which refers to the number of trees in the forest. For performing our analyses we do not consider a large number of trees in the forest, due to overfit issues and also because the training set as well as the dataset itself are somewhat small. Looking at the value obtained for such classifiers we can end up saying that if we just consider the best two models the choice of a kernel-trick for kernel pca is significant and widely affect the final size of the model, in particular that param leads to larger linear kernel-Pca based estimator than the one based on polynomial kernel, which means that the former requires more computation time for both carrying out train step and inference tasks, than the latter which is a more compact model.

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 RandomForest classifier, for sure, we could employ the classifier found from the first two 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.

9.2.1 Random Forest Classifiers References

- (**Ensemble, Non-Parametric Learning: RandomForest**) https://scikit-learn.org/stable/modules/ensemble.html#forest
- (**Gini Index**) https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifi
- (**Entropy**) https://en.wikipedia.org/wiki/Entropy_(information_theory)

10 Summary Results

10.1 Summary Results

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

# 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)
columns_2_avoid = ['ERECTED', 'LENGTH', 'LOCATION']
# Make distinction between Target Variable and Predictors
#
  ↪----------------------------------------------------------------------------
  ↪#
rescaledX, y, columns = prepare_data_for_train(dataset,
  ↪target_col=TARGET_COL)
```

```
None
Summary about Target Variable {target_col}
--------------------------------------------------
2    57
1    13
Name: T-OR-D, dtype: int64
shape features matrix X, after normalizing:  (70, 11)
```

[2]:
```
# Parameters to be tested for Cross-Validation Approach
# ----------------------------------------------------
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]
```

[3]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

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

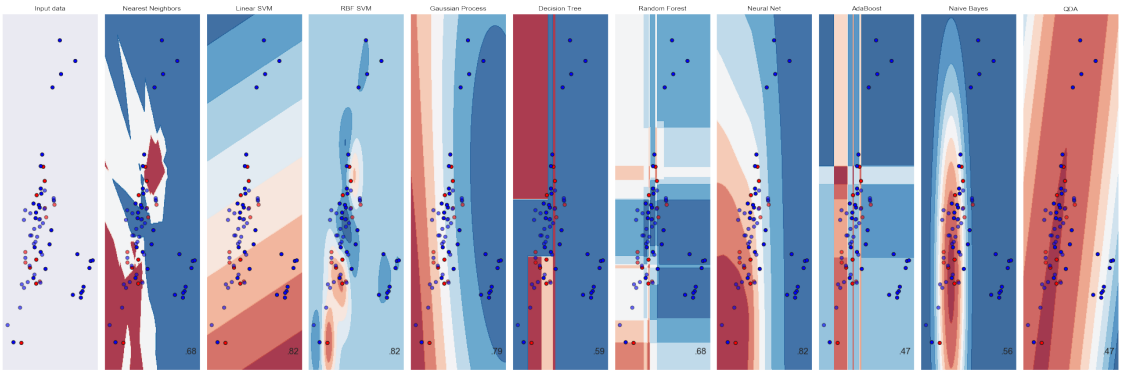
10.1.1 Summary Tables about Analyses done by means of different number of included Pricipal Components

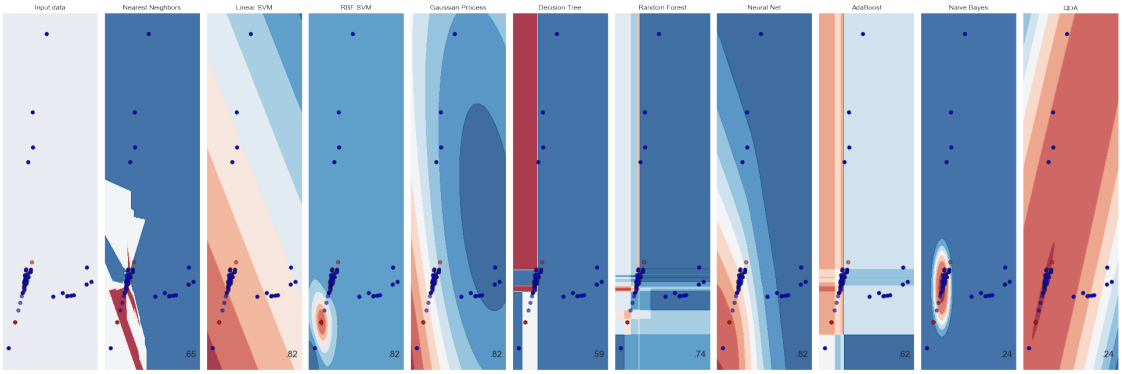
[4]:
```
# df_9_, df_12_ = reshape_dfs_acc([df_9, df_12], num_col=N_KERNEL,␣
  ↪n_cp_list=[9, 11])

# res = create_widget_list_df_vertical([df_9_, df_9_auc]); display.
  ↪display(res)
# res = create_widget_list_df_vertical([df_12_, df_12_auc]); display.
  ↪display(res)
```

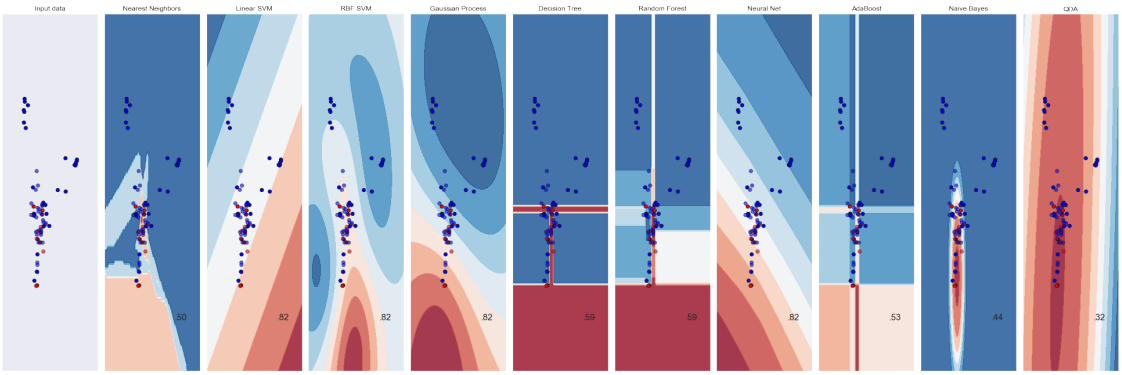
10.1.2 Summary Test

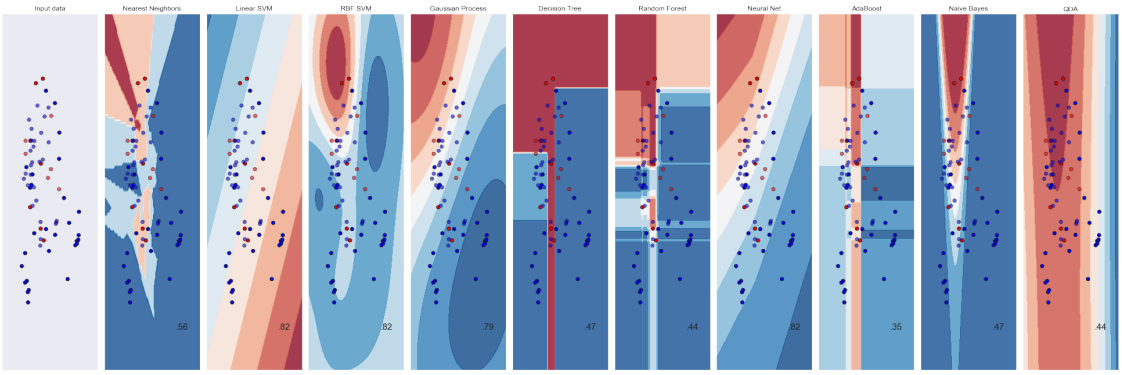
Here, in the following section I'm going to emulate a test in which I will test the different possible kinds of kernel trick, in other sense techniques, available for a Principal Component Analysis, shortly PCA, unsupervised statistical learning technique in order to remap the original features into a new N-dimensional reference system by means of the kernel approach adopted during the computation.
Once the new N-dimensional feature space is available and ready, I will experiment a bounch of selected machine learning methods and procedures applied directly on the first two most informative principal components, that is, also referred to as PCA1 and PCA2, respectively, in order to display a sequence of decision boundaries and contours retrieved after having runned each method on the selected dataset, which has been divided into halves, ofd the same size, and with the same proportion of the two classes of the target variable.
What follows is the related code, to the desciption given just above, and the results are also available through several rows of images that represent the contour and decision boundaries obtained thank to the several combinations of PCA's kernel trick and machine learning method for fitting a classifier:

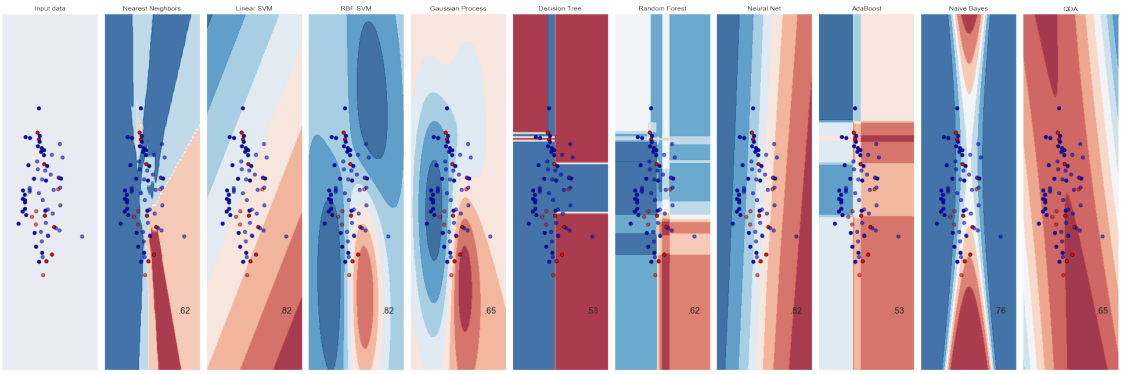
[5]:
```
kernel_pca = ['linear', 'poly', 'rbf', 'cosine', 'sigmoid'] # linear, poly,␣
  ↪rbf, sigmoid, cosine, precomputed
scaler_techniques = ['StandardScaler', 'Normalize', 'MinMaxScaler']
X = rescaledX
# Trying only StandardScaler approach
err_list = classifier_comparison_by_pca_kernels(X, y, start_clf=0,␣
  ↪stop_clf=10, scaler_technique=scaler_techniques[0], straitified_flag=True,␣
  ↪kernels_pca_list=kernel_pca[:], figsize=(27, 9), by_pairs=False,␣
  ↪singles=False, verbose=0, record_errors=True, avoid_func=False,)
```




Input data
Nearest Neighbors
Linear SVM
RBF SVM
Gaussian Process
Decision Tree
Random Forest
Neural Net
AdaBoost
Naive Bayes
QDA
.65
.82
.82
.82
.59
.74
.82
.62
.24
.24


Input data
Nearest Neighbors
Linear SVM
RBF SVM
Gaussian Process
Decision Tree
Random Forest
Neural Net
AdaBoost
Naive Bayes
QDA
.50
.82
.82
.82
.59
.59
.82
.53
.44
.32


Input data
Nearest Neighbors
Linear SVM
RBF SVM
Gaussian Process
Decision Tree
Random Forest
Neural Net
AdaBoost
Naive Bayes
QDA
.56
.82
.82
.79
.47
.44
.82
.35
.47
.44


Input data
Nearest Neighbors
Linear SVM
RBF SVM
Gaussian Process
Decision Tree
Random Forest
Neural Net
AdaBoost
Naive Bayes
QDA
.62
.82
.82
.65
.53
.62
.82
.53
.76
.65

Describing the several pictures we have obtained throughout the combination of kernel tricks available for kernelPCA technique together with different supervised machine learning techniques for building classifiers and models, we can end up saying what follows.
Looking at the first picture of each row of graphs, that is those pictures showing just data points witout any kind of decision regions as well as decision boundaries, what we understand is that the data points that are the data examples will group creating different shapes accordingly to the kind of kernel trick adopted for kernelPCA method, in particular, we can immediately see that the two categories, where blue points stands for THROUGH-like bridges while red points for DECK-like bridges, are not equally in numbers, but blue points are the greater among the two, moreover the two categories does not seem to seaprate very well but the picture is crowded with both types of categories that are strctlu closed one another. More precisely we can see that: - using *linear* kernel trick for kernelPCA procedure data points seem to be widely spread along vertical axis, and group mostly near the center of the picutre; - using *poly* kernel trick for kernelPCA procedure instead data points are mostly clusterd near the left bottom corner where seem to form a straight line and there are few examples on the upper side and some less points on the right side of the sdame picture; - while using *rbf* kernel trick for kernelPCA procedure data points seem to spread as the data points represented in the first picture so in the middle of the area but are tightly related so that are less spread along the horizontal axis; - when exploiting *cosine* kernel trick for kernelPCA method the data points are widely spread and tend to reach the top of the picture; - fianlly, when adopting *sigmoid* kernel trick for kernelPCA we can see that data points are mostly clusterd in the center of the graph.
Speaking about decision boundaries and decision regions about the selected and fitted to the data machine learning methods, what we can say is the following: - Looking at **Nearest-Neighbor Method** graphs for describing decision boundaries and decision regions we notice that in the majority of cases the decision boundaries and decision regions are prominent for the THROUGH-like bridges, sometimes the area referring to DECK-like samples are sourranded by the decision regions of the other class and the transition to the two decision boundaries is very sharp, not easilly describable. - While, looking at **Linear SVM Classifier**, and knowing the fact that we are fitting a linear classifier to the data, we are aware and so it's clear that the expected decision regions follow a pattern made from several strips of shifting shades of colors from dark red to dark blue. More precisely, three out of five Linear Svm classifiers, in paritcular those that correspond to classifiers fitted when kernel trick for Kernel PCA was set to *'rbf', 'cosine', 'sigmoid'* respectively and one at a time, show more or less the same pattern, so this classification technique combined with these kernel tricks for KernelPCA seems to behave more or less at the same way. Instead Liner Svm combined with poly kernel trick seems to lead the classifier and the resulting decisin boundaries to follow a symmetric patter with respect to the vertical axis. Finally the first combination of KernelPca and Linear Svm technique, that is linear kernel trick plus linear svm, leads to a less aggressive or finer slope of the linear decision regions. We can end up saying that in the majority of cases the transition from one extreme or edge to the other of the shade of color is smoother and continue with respect to the Nearest-Neighbor Approach. - Speaking about **RBF kernel SVM** combined with a preprocessed datset with the various kernel tricks for kernelPca Procedure we can ob-

serve that the attempt of finding decision regions on one side advantages the more numerous class that is the class corresponding to those data points classified as THROUGH-like bridges, while penalize the other which is referred to smaller region. However it seems that the classifieris able to correctly classify the data points corresponding to the less numerous class while the data points of the other class sometimes are misclassified more frequently. - Looking at classifiers trained by means of **Gaussian Process technique** we can ascertain that decision boundaries and decision regions seem to follow a straight pattern where the data points are mixing the most, while far from the bigger cluster of points that come from both categories the decison boundaries are assuming higher order so that resemble smooth nonlinear curves. In particular while in all other cases the blue region seems to occupy the left side of the graph, sometimes near the bottom and other times near the top-right, for Gaussian Process technique combined with sigmoid kernel trick for kernelPca procedure we observe that the pattern observed above is the opposite. - Even if these three methods have different characteristics they seems to lead to or provide more or less, and somehow, resulting decision boundaries and decision regions that follow a similar nature that is regions obtained dividing the available two-dimensional plane into subregions that corresponds to square regioons or alternatively irregular regions that are not corresponding to some kind of curve but rather to segmentation of the available area. These methods are respectively **Deciosn Trees, Random Forests, and Adabosts**. Where the two latter can be seen as a improvement of Decsion Tree because often the two latter are based on the decision tree classifier as unit of the overall classifier as are generally described Random Forests, and AdaBost. However Adabost and Random Forests seem to beahve more or less in the same way, in the sense that both show a predominance of reagins and subregions linked to the THROUGH class, even if the transition from one region to the other is mcuh smoother than the transition of the Decision Tree based models. - The **Niave Bayes Classifier**, when applied to the data points once preprocessed using one at a time all the suggested kernel tricks for kernelPca method as a classifier technique, leads to a results in terms of decision boundaries and regions that vary the most from one kernel trick to the other. In particular using the first three proposed kernel tricks that are ‘linear’, ‘poly’, ‘rbf’ the decison regions connected to the Deck-like bridges are concentric with respect to the surrounding area that instead is widely associated to the other class that is THROUGH-like bridges. More Precisely for ‘linear’ kernel the resulting decision boundaries are wide and spread along the vertical axis instead for ‘poly’, ‘rbf’ tend to be narrowe and to be located near the bottom of the graphic. Instead looking at the graphic that referes to the data points when preprocessed by means of cosine kenrel trick for kernelPca method we notice that it seems to lead to a opposite or simmetryc graphic with respect to the horizontal axis when compared with the graphic obtained by means of linear kernel trick. Lastly the sigmoid kernel trick leads to a graphic that seems to classifies data points from THROUGH class associating them to the left and right sides of the piciture while the top and bottom centered horizontal strip seems to be associated with data points from DECK class and more precisely the dark red areas are spotted mosty near either the top or bottom areas. - The last classifier proposed for this thiny and rough experiment is he one known as **Quadratic Discriminant Analysis**, or more shortly *QDA*. The resulting graphics suggest us that by means of such technique we observe that the DECK class is the class among the two which affects mostly the models capabilites, since the decision regions are mostly represented by shades of colors that range in the majority of case around the red color, enabling us to summarize that the DECK class differently from other

preceding models will be the most frequently predicted class with respect to the other class that is the THROUGH class.
Having performed the analyses discussed just above, employing graphics and so qualitaty approach for investigating some of the most known and exploited methods we can summarize that since we adopt jsut two PCS out of eleven possible components for predicting classes among DECK and THROUGH for T-OR-D dependent variable as our predictive or target variable, is is reallyu difficult to correctly classify all the majority of the data samples since the decisoin bundaries vary heavily from one method to the other also due to the fact that we exploit few information and knwoledge and we cannot find patterns that lead to a more precise classification. We need to exploit more features to reach better performance at classification time and find better decision boundaries that allow to separate the data points without mixing them.

10.1.3 Improvements and Conclusions

Speaking about conclusion, what we can say is the following observations and insights we get, gain and fell when analyzing *Pittisburgh's Bridge Dataset*. In general the model - that arose from training phase where we have largely and heavily adopted a *grid-search approach* with the help of *cross-validation technique* for selecting the best models from a set of hyper-parameters specified for fine-tuning purposes, depending on the kind of *kernel-trick* adopted for performing *KernelPca unsupervised learning algorithm* - leads to different performance and results:

[10]:

	AUC(%)	***P-Value(%)***	***Acc Train(%)***	***Acc Test(%)***
GaussianNB rbf	0.85	34.65	0.84	0.85
LogReg cosine	0.77	81.19	0.92	0.74
Knn rbf	0.74	1.98	0.95	0.79
SGD poly	0.73	0.99	0.92	0.76
SVC linear	0.86	2.97	0.89	0.71
Tree linear	0.62	15.84	0.84	0.71
RdmFrst linear	0.62	3.96	0.89	0.71

[10]:

	DECK-Bridge(class = 0)	***THROUGH-Bridge(class = 0)***

GaussianNB linear	*	
GaussianNB poly	*	
GaussianNB cosine	*	
GaussianNB sigmoid	*	
LogReg linear	*	
LogReg poly	*	
LogReg rbf	*	
LogReg sigmoid	*	
Knn cosine	*	
Knn sigmoid	*	
Knn rbf	*	
SGD linear	*	
SGD rbf	*	
SGD cosine	*	
SGD sigmoid	*	
SVC poly	*	
SVC rbf		*
SVC cosine	*	
Trees poly	*	
Trees rbf		*
Trees cosine		*
Trees sigmoid		*
RdmFrst rbf	*	
RdmFrst cosine	*	
RdmFrst sigmoid	*	

Total	21/25	4/25
Total linear	3/7	0/7
Total poly	4/7	0/7
Total rbf	**4/7**	**2/7**
Total cosine	**5/7**	**1/7**
Total sigmoid	**5/7**	**1/7**

- We have worked upon a dataset which is made of a small number of features as well as samples, moreover, amongst these features we do not observe pairs of attributes showing high correlation, in fact in the majority of cases the features have been characterized from a weak correlation when computing correlation matrix. However, even if the features seem somehow weakly correlated, more often them are negatively weakly correlated. Furthermore, we have seen just 5 out 12 features that also correlate with other features in a moderate way among them, which are *CLEAR-G, ERECTED, LANES, RIVER, SPAN*. Also looking at moderate correlation we can end up saying that because of the way we have coded the categorical features those moderately correlated pairs of features are indeed, again, negatively moderately correlated.

- Some have been more precise at classifying and predicting correctly class labels for most of examples that indeed belong to class-1, that is THROUGH-like bridges, while at the same time being more uncertain when estimating all target label class-0, that is DECK-like bridges. The main reason is that we worked on a small dataset which is also unbalanced with respect to the number of instances for the feature selected as the target variable to be predicted. Among the fine tuned models the following have been the best selected just for their performance metrics, without considering other metrics such as p-value: *Rbf-trick based Gaussian, Cosine-trick based Logistic Regression, Rbf-trick based Knn Classifier, Polynomial-trick based Sgd classifier, Linear-trick based Decision Trees, and Linear-trick based Random Forest*. What we can state is that Rbf kernel was adopted for two cased out of fives and in particular with a generative model such as Gaussian Naive Classifier and a instance-based model such as Knn classifier, that amongst the others are the two with the lower number of hyper-parameters. While the linear-kernel was preferred by again two out of five classifier, which both are based on decision trees-like data structure in fact are respectively Decision Trees and Random Forests Classifiers. Finally Logistic Regression adopted a Cosine kernel, so that the similarities and difference between the extracted features have been detected exploiting measures of angles and re-projecting data examples onto unit spheres, whereas the Sgd Classifier adopted a linear-kernel which enable such a model to obtain good performance since the similarities and differences between extracted features have been obtained also via a cross-features measures.

- While taking into account also p-value analysis we can end up saying that few models was able to provide at least one classifier that let us rejecting Null-Hypothesis, which are: *Rbf-trick based Knn; Polynomial-trick based Sgd classifier and Rbf-Polynomial-trick based Sgd classifier, Linear-,Rbf-, and Cosine-trick based SVMs; Linear-trick based Random Forest and Polynomial-trick based Random Forest*. While speaking about Gaussian, Logistic Regression and Decision Trees models, them have not been able to provide us at least a classifier that enables us rejecting Null-Hypothesis.

- Some models, due to overfit, mispredict most samples that indeed belongs to a given class while other samples have been correctly classified in the majority of cases. In particular we have to make distinction between those models that classified all data examples as belonging to class-1, from the others that instead predicted all examples as belonging to class-0. In fact, the former appear as to be somewhat estimators with quite good performances, indeed them just obtained such acceptable performance just because those scores are affected by the fact that the model is unbalanced amongst the number of samples from different classes. While the latter without any doubt show directly worst performances, since the number of samples that belongs to class-1 but that have been misclassified account for a large portion so that the performance scores go down dramatically.

Speaking about further extensions, 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.
- Using *Rank correlation coefficients*, such as *Spearman's rank correlation coefficient and Kendall's rank correlation coefficient* which allow us to discover whether or not we can take into account nonlinearity in more robust way when analyzing the degree of correlation among all the pairs of possible features.

10.2 References section

10.2.1 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
  - (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#

10.2.2 Others References

- Plots:
  - (Python Plot) https://www.datacamp.com/community/tutorials/matplotlib-tutorial-python?utm_source=adwords_ppc&utm_campaignid=898687156&utm_adgroupid=48947256715
    299261629574:dsa-473406587955&utm_loc_interest_ms=&utm_loc_physical_ms=1008025&gclid
    _j1BRDkARIsAJcfmTFu4LAUDhRGK2D027PHiqIPSlxK3ud87Ek_lwOu8rt8A8YLrjFiHqsaAoLDEALw
- Markdown Math part:
  - (Math Symbols Latex) https://oeis.org/wiki/List_of_LaTeX_mathematical_symbols
  - (CheatSheet) https://www.ibm.com/support/knowledgecenter/SSHGWL_1.2.3/analyze-data/markd-jupyter.html
  - (Tutorial 1) https://share.cocalc.com/share/b4a30ed038ee41d868dad094193ac462ccd228e2/Home
    %20Markdown%20and%20LaTeX%20Cheatsheet.ipynb?viewer=share
  - (Tutorial 2) https://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Typesetting%20

List of Figures

List of Tables