Data Space Report (Official) - Logistic Regression Analysisv1.0.2

June 11, 2020

1 Data Space Report

1.1 Pittsburgh Bridges Data Set

`Andy Warhol Bridge - Pittsburgh.`

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

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

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

1.1.1 Imports Section

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

```
None
```

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

```
[3]: # =========================================================================== #
     # READ INPUT DATASET
     # =========================================================================== #
     dataset_path, dataset_name, column_names, TARGET_COL = get_dataset_location()
     estimators_list, estimators_names = get_estimators()
```

```
[4]: dataset, feature_vs_values = load_brdiges_dataset(dataset_path, dataset_name)
```

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

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

```
Summary about Target Variable {target_col}
--------------------------------------------------
2    57
1    13
Name: T-OR-D, dtype: int64
shape features matrix X, after normalizing:  (70, 11)
```

1.2 Pricipal Component Analysis

[7]:
```
show_table_pc_analysis(X=rescaledX)
```

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

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

Major Pros & Cons of PCA

1.3 Learning Models

[8]:
```
# Parameters to be tested for Cross-Validation Approach
# -----------------------------------------------------

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

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

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

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

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

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

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

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

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

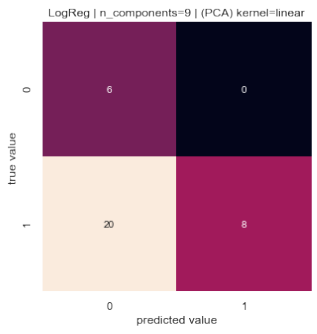
[9]:
```
n_components=9
learning_curves_by_kernels(
# learning_curves_by_components(
    estimators_list[:], estimators_names[:],
    rescaledX, y,
    train_sizes=np.linspace(.1, 1.0, 10),
    n_components=9,
    pca_kernels_list=pca_kernels_list[0],
    verbose=0,
    by_pairs=True,
    savefigs=True,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

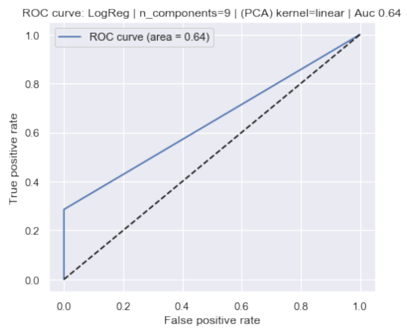
[10]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

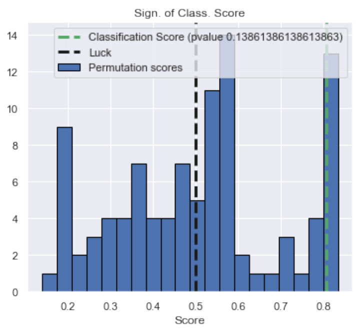
```
<IPython.core.display.Javascript object>
```

[11]:
```
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
```

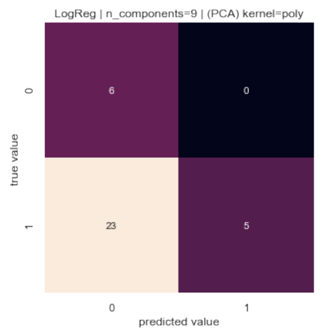


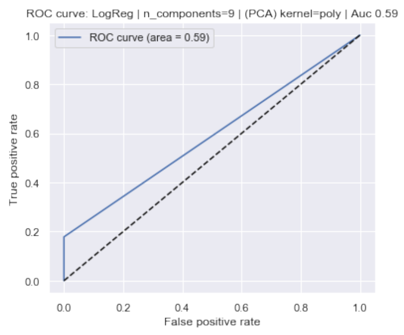


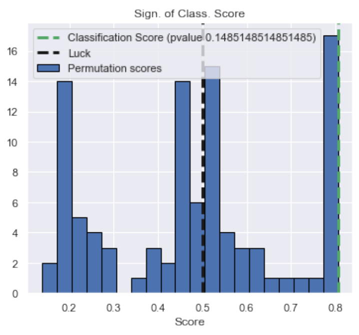


```
Kernel PCA: Poly | LogReg
================================================================================
====================
              precision    recall  f1-score   support

     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
```

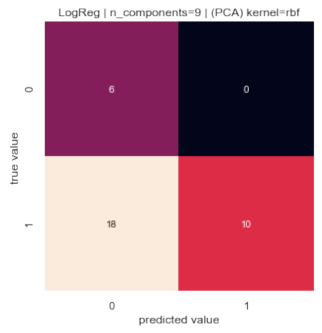
```
                 0.92                0.32  0.59  0.14851
```

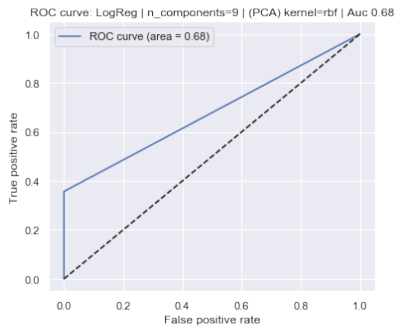


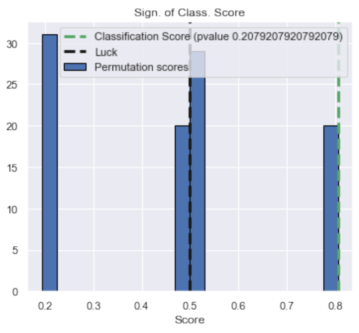




```
Kernel PCA: Rbf | LogReg
================================================================================
====================
              precision    recall  f1-score   support

     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
```

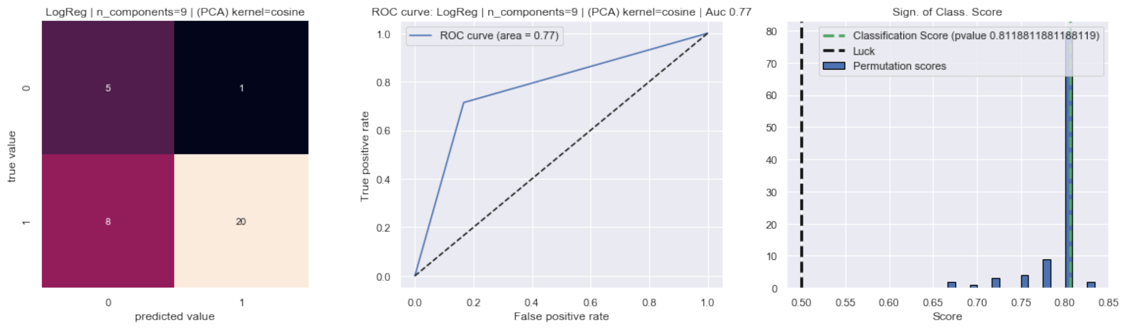




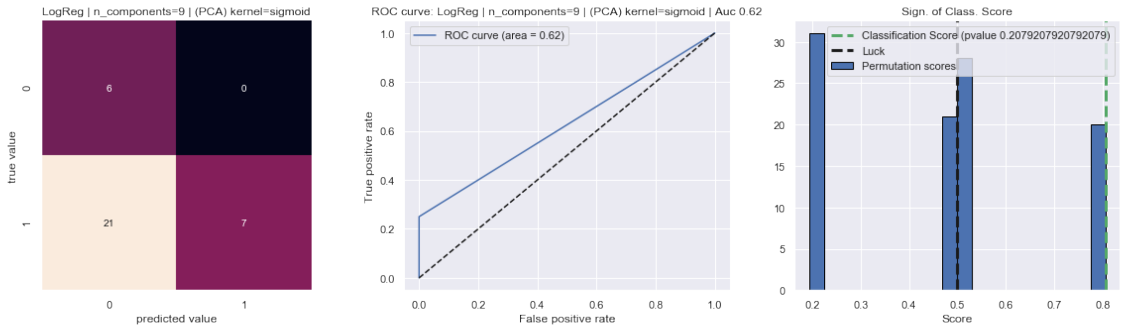


```
Kernel PCA: Cosine | LogReg
================================================================================
====================
              precision    recall  f1-score   support

     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 Logisti 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: - speaking about **Linear kernel Pca based Logisti 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. - observing **Polynomial kernel Pca based Logisti 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. - review **Rbf kernel Pca based Logisti 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. - **Cosine kernel Pca based Logisti Classifier** results to be the best 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 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. - lastly, also **Sigmoid kernel Pca based Logisti 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 hyper-parameters and model's weights for describing the underling model related to the data.

Table Fine Tuned Hyper-Params (Logisti Regression)

[12]:
```
# 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)
```

[12]:
```
               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. - instead* 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.

1.3.1 Improvements and Conclusions

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

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

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

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

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

1.4 References section

1.4.1 Main References

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

1.4.2 Others References

- Plots:
  - (Python Plot) https://www.datacamp.com/community/tutorials/matplotlib-tutorial-python?utm_source=adwords_ppc&utm_campaignid=898687156&utm_adgroupid=48947256715&ut
299261629574:dsa-473406587955&utm_loc_interest_ms=&utm_loc_physical_ms=1008025&gclid=C
_j1BRDkARIsAJcfmTFu4LAUDhRGK2D027PHiqIPSlxK3ud87Ek_lwOu8rt8A8YLrjFiHqsaAoLDEA
- Markdown Math part:

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

```
[ ]:
```