Pázmány Péter Katolikus Egyetem

Információs Technológiai és Bionikai Kar

Data Mining and Machine Learning

Final Project Documentation

Csatári Dominik, FV1TW4

Software Engineer Msc

2024

Contents

Chapter 1

Digging into Data

1.1. Data exploration

Before I have started the reduction of features, I made sure, that I treat missing values accordingly. As the dataset did not have any missing values, I was able to proceed. On Figure 1.1, we can see that the there is a normal distribution of values, represented in the test and train set as well.

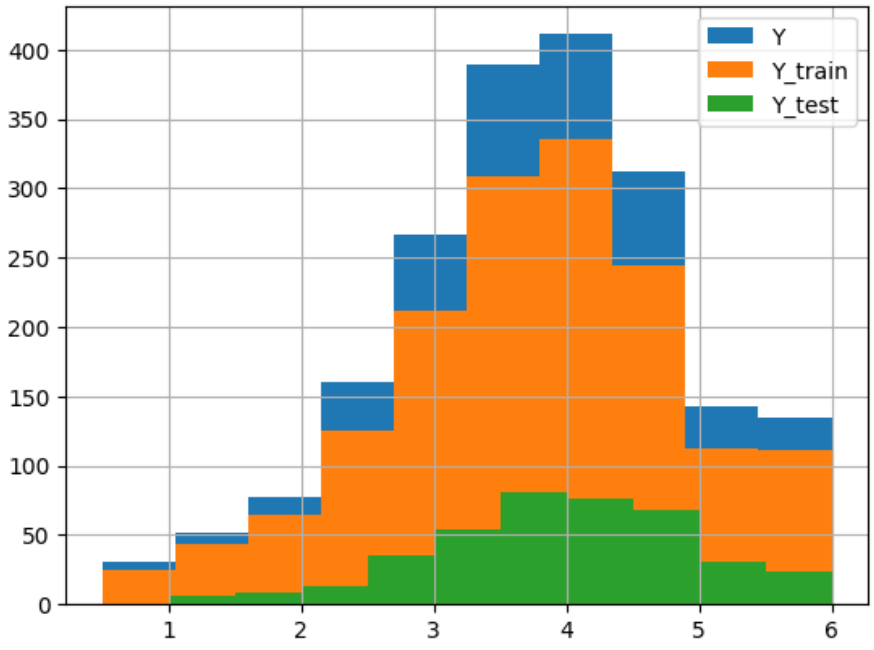

Figure 1.1: Distribution of Y values in original values and test and train set.

With the library *sweetviz*, I have generated report about the model, to have a deeper understanding of the data. As it is shown on Figure 1.2, there were a lot of features, that have little to no variance in them. These features needed to be dealt with at the

preprocessing step, so the model complexity would get too big.

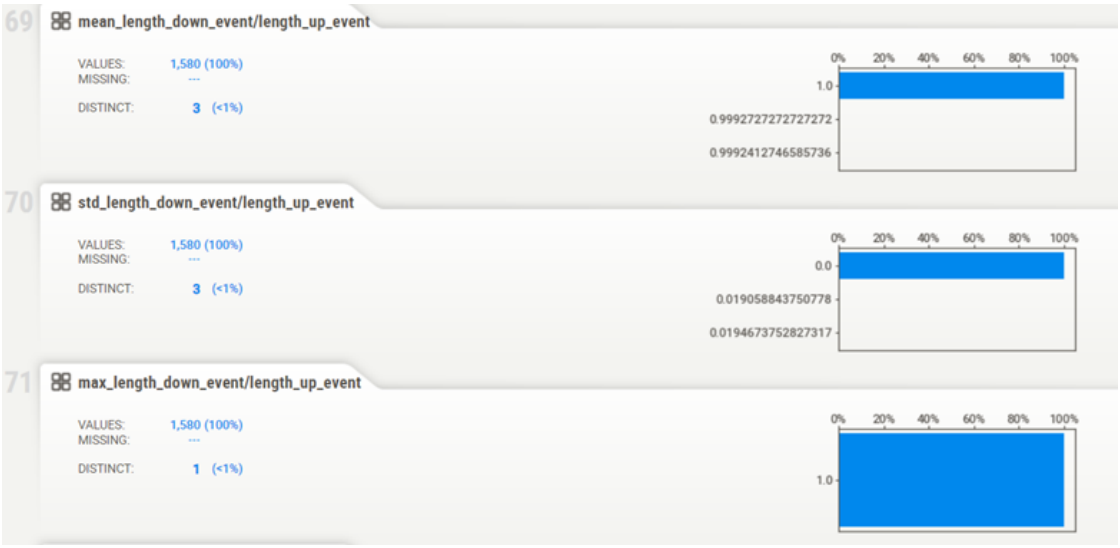

Figure 1.2: Example of features having few distinct instances.

1.2. Preprocessing

As the dataset, we had to work on had a lot of features(468), I have considered a few feature reduction or selection methods to decrease the dimensionality and with it decrease the models complexity as well. Note, that I have only used these as a proof of concept type, as for the majority of these models need more instances to learn.

1.2.1 PCA

With PCA I have reduced the features number from 468 to 349. First I have scaled the train and test set so it would become a valid input for PCA. I have used the parameters

```
pca = PCA(n_components='mle', svd_solver='full'
```

setting, *sklearn* calculated the best number of features to have. I have made calculations with the original dataset modified with PCA. The results will be described at each model, where I have used PCA.

1.2.2 Variance Threshold

With *sklearn* library *VarianceThreshold* and some other code parts, I have created so, that the code removes a feature from the dataset if the variance of its values are below a threshold. Note, that this simple application of variance could remove perhaps, important information, as a threshold value can not be that adaptive, however it still can

provide some interesting results. These results will be discussed at the part where I have implemented it.

1.2.3 K-means

Notebook: *CSATARI_FV1TW4_Third.ipynb*

With Kmeans, I was able to transform my data to a space having less dimensions. On figure 1.3, the results of the pipeline

```
pipeline.make_pipeline(Scaler(),cluster.KMeans(n_clusters = 2)).fit(X)
```

can be seen. This creates two clusters from the dataset. With this transformation we can see outliers. As most of the data is close to the centers, having a more reasonable number of centers according to the feature size it can be a great way for feature reduction [4].

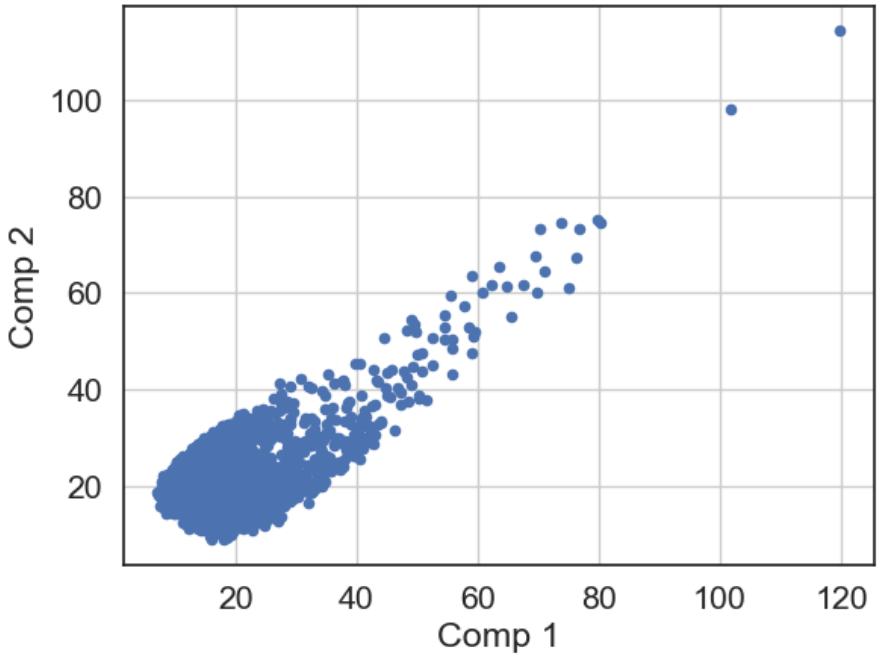

Figure 1.3: Kmeans with two centeroids. **Comp 1** is the distance from **Center 1** and **Comp 2** is the distance from **Center 2**.

1.2.4 Correlation

Notebook: *CSATARI_FV1TW4_FORTH.ipynb*

There are some features in the data that change correspondingly to each other, that is the reasoning behind having highly correlated features on Figure 1.4. As correlated

feature do not provide further information in the data, I have to remove these features (at least one in every pairs) or transform the dataset with listed methods. Further investigating in correlated values with pandas command

```
corr_matrix = X.corr()
```

it is apparent that there are further correlated features. However removing these features might have other consequences as well. For example **Feature A** and **Feature B** indicates a pattern in the data. However **Feature A** and **Feature C** are correlated, we remove **Feature A** from the set. In the process we lost information, but it might still be better with less dimensions. The main factor to follow is having a lower error rate [1].

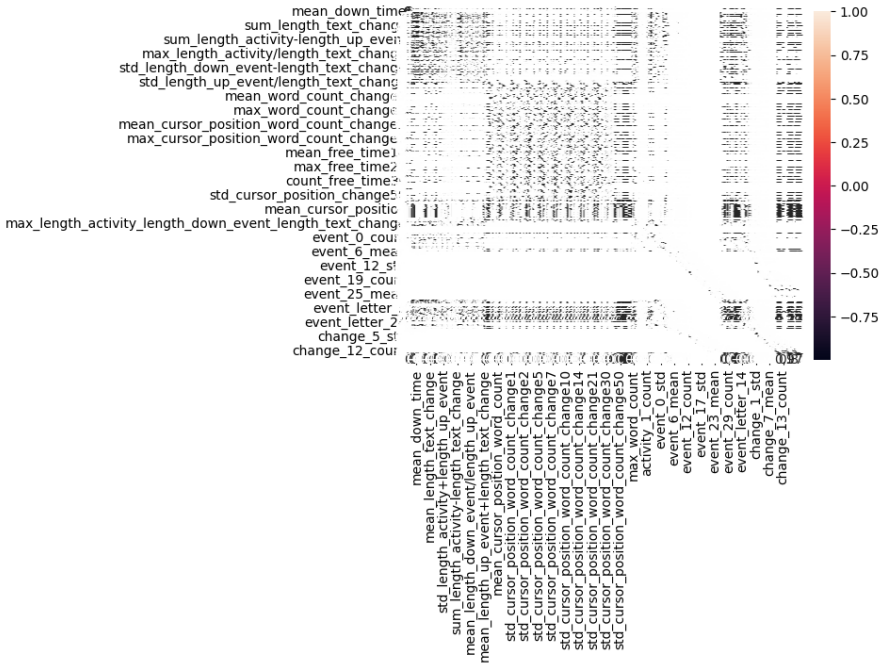

Figure 1.4: Heatmap of the correlation matrix of the original dataset visualized with *seaborn*.

Chapter 2

Models

In this section of the Report, I am going to describe briefly the results and the conclusions drawn from these, according to each model. I have created various models and only uploaded those, that had a really low *MSE* or *RMSE* on my test or validation set.

2.1. SVM

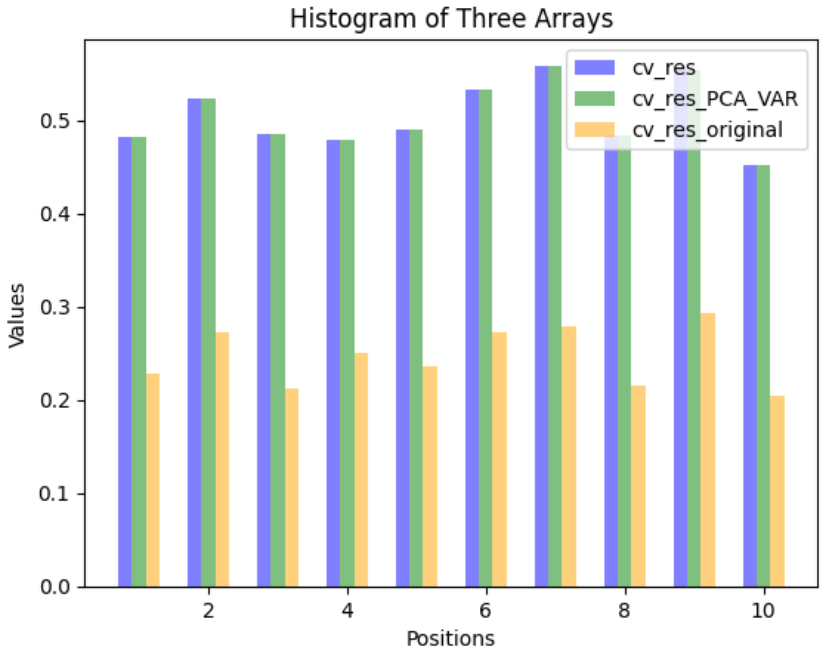

Figure 2.1: Histogram of Error Rates according to 10-fold dataset smaller sets (from 1 to 10). CV_res indicates the variance thresholded dataset, CV_res_PCA_VAR is the PCA and variance controlled and cv_res_original is the original.

Support Vector Machine is usually used for classification tasks, however it can be used for regression as well. *Sklearn* provides a library for it. As SVR can be used on various kind of numeric datasets, I have trained and tested it on the original dataset, and on Variance controlled and PCA transformed, and even Kmeans transformed. The general results were the following:

- **Orignal Dataset:** 0.7241 *Test Set*, 0.919 *Kaggle Result*
- **Grid Search with Feature Reduction:** 0.4115 *Test Set*, 0.684 *Kaggle Result*
- **K-means transformed:** 0.45 *Validation Set*, 1.003 *Test Set*

Note, that on my test set, I checked *MSE* while on *kaggle* we have used RMSE. Feature reduction was done via variance thresholding. The result of the Grid Seach was found to be:

- Variance: 0.001
- Epsilon: 0.3579
- Regularizer C: 1

Note, for grid search I have used the same test and train set. For other parts of the documentation I have used different regularization values. For other considerations, I have created different dataset with variance and epsilon being the grid search values(0.001). I have generated a cross validation process and models on different dataset, them being: X_New being Variance controlled, X_PCA PCA transformed and the original X. On Figure 2.1, we can see, that the best result was fitted on the original model.

The cross validation and the models were created with the command

```
SVR2 = svm.SVR(C=0.1, epsilon=eps)
cv_res_PCA_VAR = cross_val_score(SVR2, X_new_PCA ,Y['score'], cv=10)
```

The Variance threshold controlled and PCA and Variance transformed dataset has the same results, even though the PCA dataset has 269 features and the only variance thresholded dataset has 370. This means, that Support Vector Machine is able to create great decision boundaries.

With K-means transformation I have mainly followed a trial-and-error path, with which I have generated a lot of pipelines with the following command:

```
pipeline.make_pipeline(Scaler(), n_clusters = n, svm.SVR())
```

Where I had tried to find a reasonable number of clusters. It had to be smaller than the original feature number, but reasonably big, so that the dataset would still hold some valuable information. I had not found such a number.

2.2. Neural Network

Notebook: *CSATARI_FV1TW4_NN.ipynb* [2]

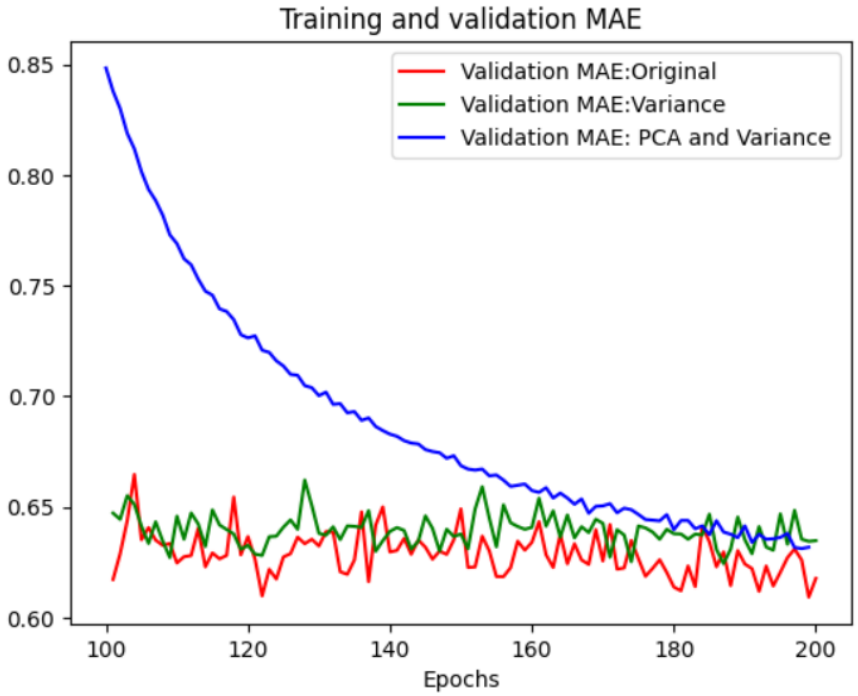

Figure 2.2: Neural Networks accuracy improving with newer epochs. On the Y-axis we can see the validation loss, and on the X-axis the number of Epochs.

In this section I will briefly describe my results in the neural networks I have created. I have worked on the same kind of dataset I have described in Section 2.1. PCA, PCA and Variance and Original. For the network, I have set the loss to MAE, so it would not be sensitive for outliers. I have tried Early Stopping and setting different regularization fro Adams optimizer, but the general 100 epochs size seemed as one of the best solution. Early Stopping would have helped in case of overfitting. Note, that I have set the epochs for 200 only for visualisation purposes. This high number was only needed in case of

PCA and Variance transformed dataset. The 3 model contains similar layers, with the same activation. However, as there are less features, in reduced datasets, I have applied this knowledge to the number of neurons and layers accordingly. As the Figure 2.2 shows, the validation lost is small for Validation transformed and original dataset. For PCA and Variance transformed, it needs "time"(epochs) to adjust, as it has way fewer features to learn from. But at the end in generalizes, and we get small errors in all of the cases. On *Kaggle* the result was 0.812. With more learning data and early stopping, it would be a great choice.

2.3. Linear Regression

Notebook: *CSATARI_FV1TW4_Third.ipynb*

I have tried Linear Regression in it's own, however it did not give a great result. After that I have tried it with fitting it on the data transformed with K-means. With K-means I have got better results. With fine-tuning this might work, however I still was not able to get the loss below 1.

2.4. XGBoost

Notebooks: *CSATARI_FV1TW4.ipynb*, *CSATARI_FV1TW4_Second.ipynb*, *CSATARI_FV1TW4_Third.ipynb*

With this method I was expecting great values because of the boosting method it uses. First, I have implemented it simply with the command

```
XGBRegressor(n_estimators = 300, learning_rate = 0.001, max_depth = 10).
```

My error *MSE* estimated on my test set was 0.51 and with *RMSE* on *Kaggle* 0.767, which is great result, given the dataset. Next I have changed the parameters the following way:

```
XGBRegressor(n_estimators = 200, learning_rate = 0.1, max_depth = 20).
```

Which ended up with a worse test *MSE*, being 0.57.

Transforming the data Kmeans having 30 cluster centers gave me the results of *RMSE* being around 1.

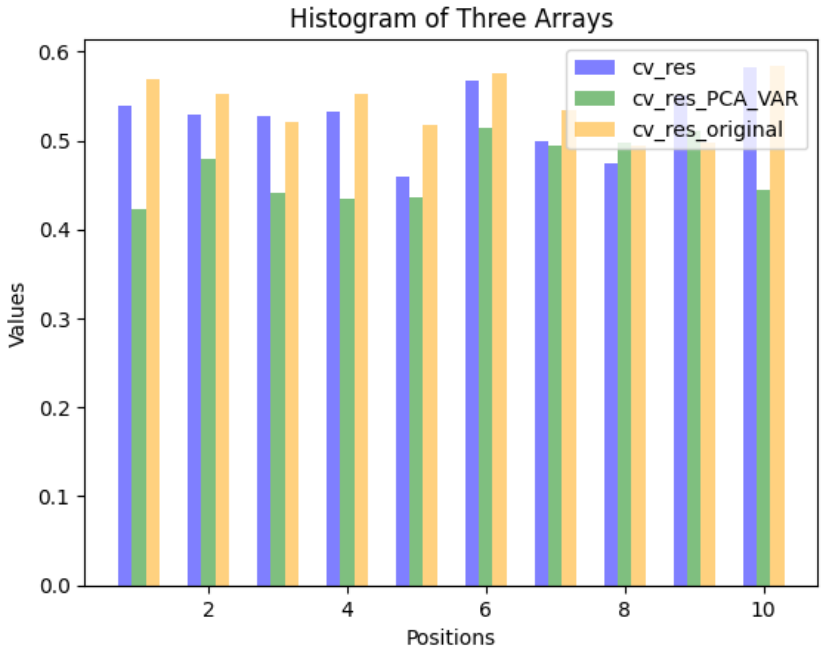

Figure 2.3: 10-fold cross validation results of XGBoost on the 3 datasets. CV_res indicates the variance thresholded dataset, CV_res_PCA_VAR is the PCA and variance controlled and cv_res_original is the original.

I have also done a cross-validation for different datasets as before, and as Figure 2.3 shows, usually the modified dataset is getting better results proving we gave a better initial feature set for the model, by feature selection.

Also, I have implemented sklearn's GradientBoostingRegressor(). With this 200 estimators and a 0.1 learning rate, I had a 0.9371 *RMSE* on my test set.

2.5. AdaBoost

Notebooks: *CSATARI_FV1TW4.ipynb*, *CSATARI_FV1TW4_Second.ipynb*, *CSATARI_FV1TW4_Third.ipynb*

Firs in the first notebook I have created a grid search to find the best values for Adaboost with this mdoel. For the basic dataset it seemed to be

```
AdaBoostRegressor(learning_rate=0.311, n_estimators=81)
```

Having a result of 0.645 *MSE* on my dataset. On *kaggle* it was 0.672. After this in the Second Notebook I have created a model with a learning rate of 0.1 and with n_estimators being 150 on the PCA dataset, where my results was worse, 0.7162 *RMSE*.

In the Third Notebook, I have implemented 3 model having values

`AdaBoostRegressor(n_estimators=80, learning_rate=0.08, loss='linear').`

On this I have created a 10-fold cross validation with the 3 datasets described in details in Chapter 1. As you can see from the results on Figure 2.4, we similar results as in the case of XGBoost (Figure 2.3) described in in Section 2.4, having the transformed dataset performing the best.

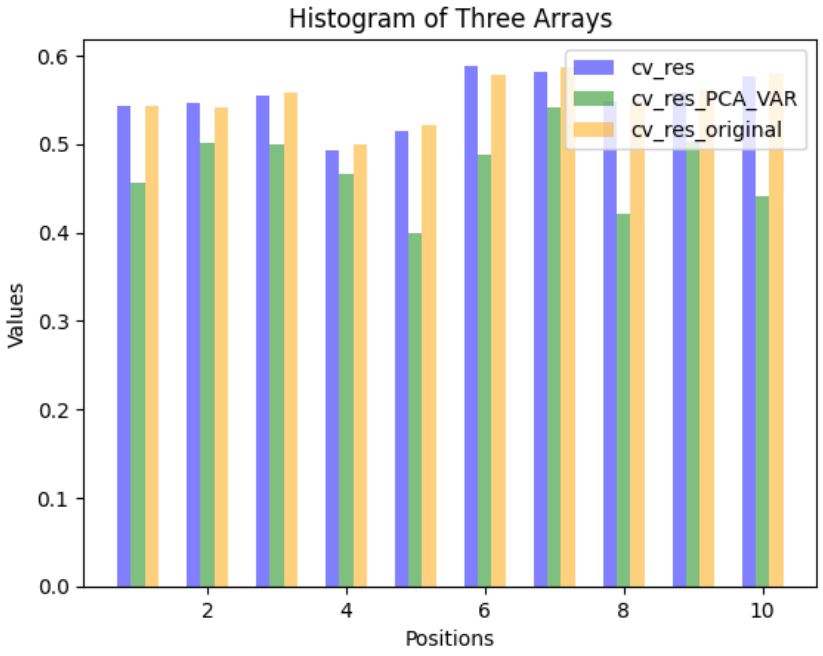

Figure 2.4: 10-fold cross validation results of Adaboost on the 3 datasets. CV_res indicates the variance thresholded dataset, CV_res_PCA_VAR is the PCA and variance controlled and cv_res_original is the original.

2.6. CART

Notebook: *CSATARI_FV1TW4_Third.ipynb*

I have implementing a decision tree with the help of *sklearn DecisionTreeRegressor*. With this I have tried cross validation on the whole dataset, and results on the modified datasets described in the Chapter 1 (PCA an Variance thresholded datasets). I have used absolute error so the fitting phase will not be penalized upon its outliers. Both models trained on the modified datasets gave around the same *RMSE* (around 0.75) neither really stood out. I did a 10-fold cross-validation as well it did have a better result, lower loss, however it is only a cause of it having less instances to test the process on.

2.7. RandomForest

Notebooks: *CSATARI_FV1TW4.ipynb*, *CSATARI_FV1TW4_Second.ipynb*, *CSATARI_FV1TW4_Third.ipynb*

As random forest had a lot of optimization capabilities and it gave me the best initial result, I have decided to try to implement it in various scenarios. I the First Notebook, with the parameters being set to

```
RandomForestRegressor(n_estimators=100, random_state=42).
```

On my test set it gave a result of 0.4046 *MSE* and on *Kaggle* a 0.649 *RMSE*. With criterion set to absolute error the *MSE* became 0.4177, higher, which indicated that the prediction loss was not a direct cause of outliers. With Halving Grid Search, I found that on the grid I have tested I got the best results to be

```
min_samples_split': 15, 'n_estimators': 120
```

I have chosen Halving Grid Search as an algorithm because it seemed the most reasonable for me after reading "Tuning the hyper-parameter" page from *sciktilearn*[3]. As I already have a small amount of data, splitting it with SH and racing different estimators seemed like an interesting path to follow, allocating more and more resources for the "winning" estimator Note, that in these search these were the highest values, so these values might not be the best. This is proved by the prediction made on this best estimator having a 0.6289 *RMSE* on my test set.

I have also generated a pipeline transforming the data with kmeans than on that scaled and transformed data I have fitted a random forest regressor. The RMSE of this was 1.21 with 50 clusters and the result average r2 result of the 10-fold cross validation was 0.497.

2.8. KNN

Notebook: *CSATARI_FV1TW4.ipynb*,*CSATARI_FV1TW4_Second.ipynb*, *CSATARI_FV1TW4_Third.ipynb*

With KNN with trial-and-error, I decided to do regression with the the parameters:

```
KNeighborsRegressor(n_neighbors=70,weights='distance').
```

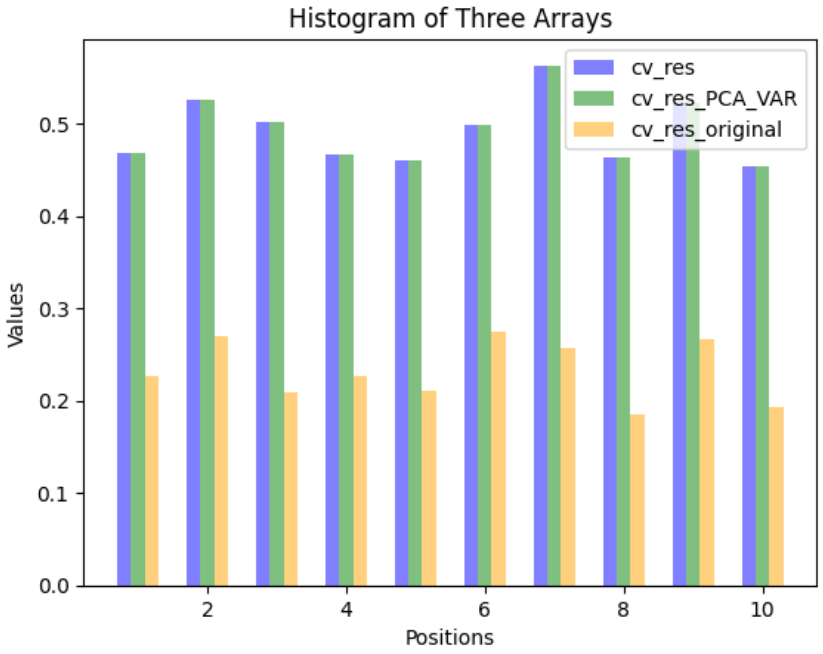

Figure 2.5: 10-fold cross validation results of KNN on the 3 datasets. CV_res indicates the variance thresholded dataset, CV_res_PCA_VAR is the PCA and variance controlled and cv_res_original is the original.

My *RMSE* result was 0.8673 on my own test set and 0.92 on *Kaggle*. With the transformed datasets mentioned in Chapter 1, I have got the results shown on the Figure 2.5. These validation results seem too good, too optimistic. Which is true as the best models MSE value on my test set is 0.7451.

Also, KNN also gave a great result with other parameters, for example with these parameters

```
KNeighborsRegressor(n_neighbors=50,algorithm='auto',weights='distance')
```

on the PCA transformed dataset it gave a 0.68 *RMSE*.

2.9. Lasso and Ridge Regression

Notebook: *CSATARI_FV1TW4_FORTH.ipynb*

In the last Notebook I have implemented to Lasso and Ridge regression models to see, how much the overfitting penalized my results so far. I did a simple grid search for finding the best alpha value for each, which alpha value is

$$\|y - Xw\|_2^2 + \alpha \cdot \|w\|_2^2 \tag{2.1}$$

in Ridge [5] and in Lasso (Lasso documentation writes it bit differently, see at [6]). For the values, I get in the Grid Search, I got 1 as the best for Lasso. This result is the evaluation of the mean of the 10-fold cross validation. It basically means, that the $\|w\|_2^2$ part takes over, as it can make some parameters coefficient zero. This is a kind of self taught feature selection. It also gave a good result on my test data, it had never seen before, being 0.45 *MSE*. However, at the scaling part I might have messed something up, as the *Kaggle* results were bad. It is only true, for lower alpha values, as for alpha=1, on *Kaggle* the RMSE was 0.858. Most probably that was a mistake of the scaling.

References

1. Youtube Video about Feature Selection.
2. How to build a Neural Network Tutorial
3. Tuning the hyper-parameters of an estimator
4. Less Known Applications of k-Means Clustering
5. Ridge Sklearn
6. Lasso Sklearn