University of Glasgow

Title: Feature Engineering Case Study 1

Name: Omar Jabri, Yifan Xie, Tomas Simutis

ID: 2519359j, 2509996x, 2603015s

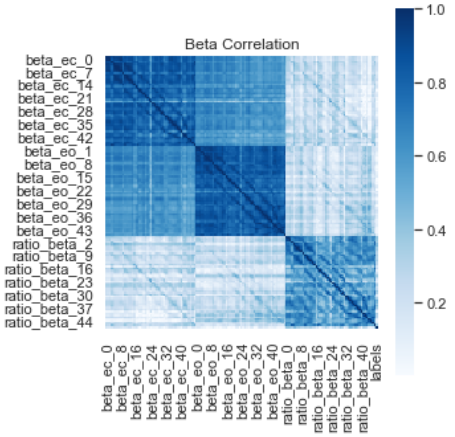
Introduction:

- Approximately 50% of people with Spinal Cord Injury (SCI) have Central Neuropathic Pain (CNP). There is currently no treatment for CNP, only prevention. However, manual assessment is time-consuming and prone to errors. There is evidence that brain Electroencephalogram (EEG) data can be used to predict whether a patient will develop CNP.
- The provided dataset has data from 18 patients, 10 of which have developed CNP after six months and 8 have not. There are 10 repetitions per patient, and the features in the dataset are the recordings of the brain's electrical activity at 250 Hz from 48 electrodes. The data provided has already been preprocessed a bit.
- Electroencephalogram (EEG) signals contain a lot of information, and some of it might not be as important as others for predicting if a patient will develop CNP. The idea is that if we select only the features that contain meaningful data for prediction and discard the others, the classifier will be more accurate. This study explores different methods for feature selection and then compares their performance using various metrics. The feature selection methods include filtering, wrapping and embedded techniques. They try to select a subset of features that describe the data most efficiently and lead to a better classification solution while considering the nature of the data (i.e. medical).
- Thus, this experiment aims to find the best method that reduces the features into a simpler set and fit a classifier with high sensitivity and AUC scores. This will lead to establishing a robust pipeline that reduces the risk of misclassifying the positive class while maintaining high performance.

Methods:

1- Data Analysis:

- The first step was to check for any outliers within the data to clarify which scaler to use. The Z score of the dataset describes the data points' relationship with the standard deviation and mean of the whole set. The features with values far away from zero (a threshold of + or - 3 was applied) are described as outliers. In this study, several features were, in fact, outliers themselves. Hence, the Robust Scaler, being "robust" against outliers, was used to scale the feature data points.
  Moreover, it is vital to check for any high correlations between the filters. Hence a correlation matrix (using Spearman's correlation) was plotted for further analysis:



  Looking at the correlation matrix above, it is obvious that some features are highly correlated together (e.g. ratio_beta_2 and ratio_beta_8).

2- Classifier:

- Since the data is crucial and needs to be handled with care, a model that is robust against overfitting needs to be chosen. Thus, the model chosen was: **Stochastic Gradient**

Classifier. This classifier uses regularised linear models with stochastic gradient descent learning, which allows good defence against overfitting and is **fixed and used for all methods.**

- The only hyperparameter that needs tuning is alpha, the learning rate. This has been successfully completed using Grid Search technique on a wide range of alpha values starting from 0.001 to 1.0. **For each feature engineering method** used in this study, **grid search** has been performed again on the new reduced dataset to achieve higher performance whilst keeping an eye on overfitting.
- Moreover, an important parameter called `early_stopping` is set to `True`. The latter is a regularization parameter that is used to avoid overfitting as well.

3- Feature Engineering:

A. High Correlation Filter:

- First method implemented using Spearman's correlation. Features that are highly correlated approximately hold the same type of information. Hence, by cancelling out the least important data points, the model will fit on a simplified version of the dataset that describes the relationship better or on the same level.
- This method has been developed by using a `VarianceThreshold` sklearn function. The latter drops all features that have a variance > **95%** of the whole dataset. The threshold used was carefully tested and settled at **0.01**.
- Afterwards, features that are **quasi-constant**, (i.e. with 0 variance), that have no information to give to the model are dropped as well.

B. Mutual Information Filter:

- The aim of this filter is to find $\hat{s}$: Optimal subset of features that maximises the mutual information between the subset X_S and the labels Y. Hence, **this method is viable** since it reduces the features into a smaller subset that describes the data better.
- The only parameter to tune is the k number of features that maximise the mutual information using `SelectKBest` sklearn function. By using Grid Search cross validation, this parameter is easy to find by trying a number of k features from 1 to 432 (with a step of 20 features) and finding the best number $\hat{k}$ **that maximises the performance.**

C. Anova (f-Test) Filter:

- Known as the standard analysis of variance, Anova feature selection method tests the relationship between predictors and target variables using the **f-Test**. **Variance is an important metric** when it comes to feature selection since **it determines how much a feature impacts the target variable and how much information it gives to the dataset**.
- This filter is used with `SelectKBest` function as the one in Mutual Information, alongside Grid Search cross validation to find the best **k parameter** subset of features that represent the data best (having the highest variances) while enhancing the model's performance.

D. Recursive Feature Elimination (RFE) Wrapper:

- This type of feature engineering method differs from the previous ones in the sense that it chooses the **optimal subset of features** that enhance the estimator's performance.
- The important parameter to note is the **k optimal number of features** that the algorithm calculates recursively.

- This method has been developed using sklearn's function `RFECV` that returns the k best features to transform the dataset.

E. Ridge/Lasso Regularization Embedded:

- Introducing a new type of feature engineering, Embedded Methods. The latter, combine Filtering and Wrapper methods into one. They extract feature importance based on the model's performance during training while using the regularization technique to penalize the parameters of said model for avoiding overfitting.
- The regularization functions used in this study are `Ridge()` and `Lasso()`.
- Both of these methods were used as parameters of the sklearn function `SelectFromModel()`, which selects the **best k features** based on the models used.
- The parameter to tune in both regularization methods is the learning rate **alpha**, which was carefully tested and chosen to enhance the model's performance and its robustness against overfitting.

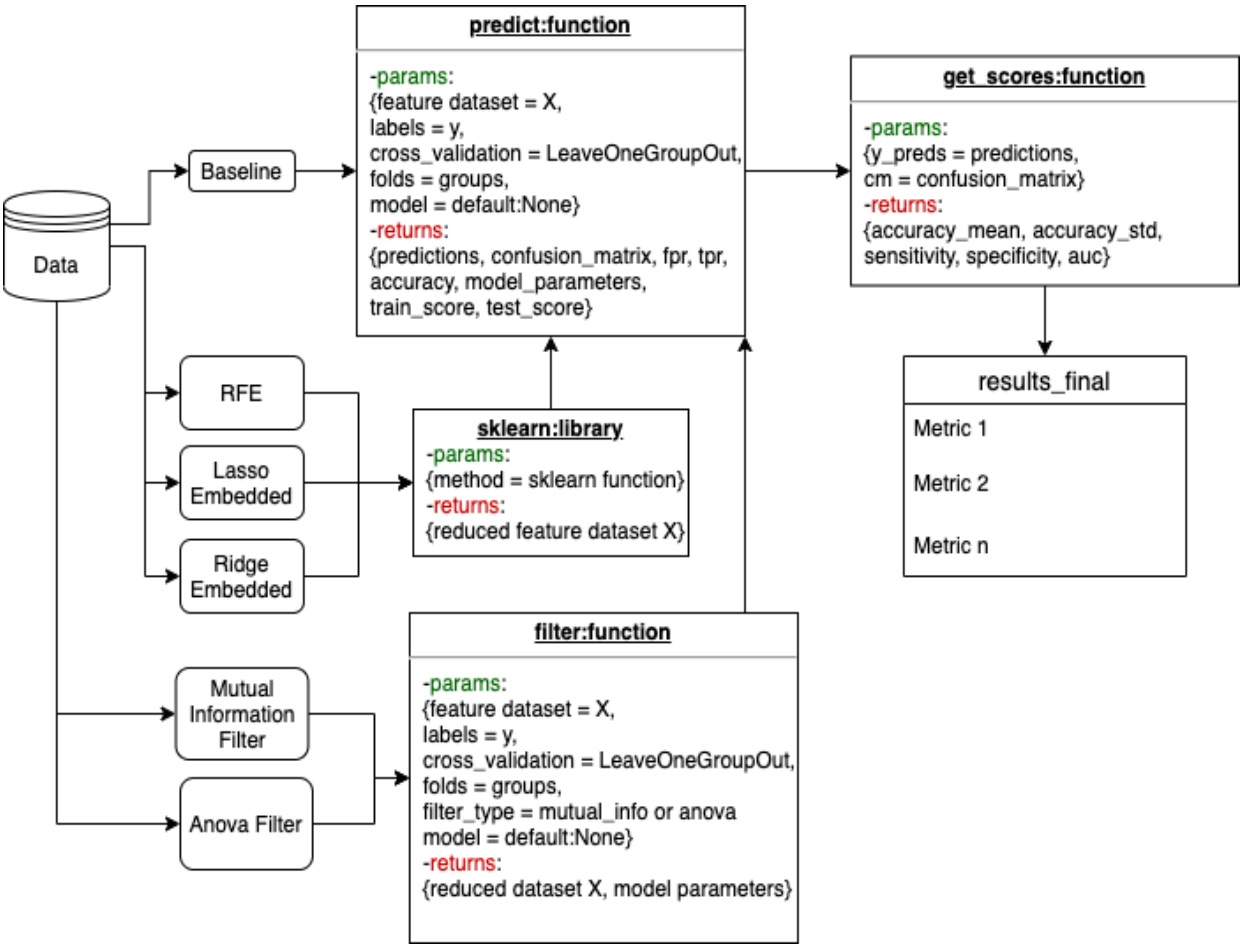
Experiments and Results:

- These methods were implemented to demonstrate how they differ from each other in terms of feature reduction and their effect on performance. By choosing techniques from **filters, wrappers and embedded methods**, the experiment is then fortified by comparing all of them and choosing the **optimal methodology** that suits this problem best.
- The expectations taken into account were the following:
    1. The wrapper method is going to be **heavy** in terms of **computation** and **execution time**.
    2. The filtering methods are going to be **fast** to compute but **not as accurate** as the rest.
    3. The embedded methods are going to be as **fast** as the filtering, and as **accurate** as the wrapper. Hence, why it is expected to have the **best performance**.
- This dataset is fragile, in the way that it holds EEG readings of patients that do or do not have CNP. Hence, training and testing should not be treated lightly. **Cross validation** is a known technique to split datasets in a systematic way that allows models to achieve their best performances while avoiding overfitting. Since every **9 rows in the dataset belongs to one subject**, `LeaveOneGroupOut` cross validation method has been implemented. It generates a group of folds based on the subject and splits the data accordingly.
  This group has the following format:

```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  3,  3,  3,  3,
        3,  3,  3,  3,  3,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  5,
        5,  5,  5,  5,  5,  5,  5,  5,  6,  6,  6,  6,  6,  6,  6,  6,
        6,  6,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  8,  8,  8,  8,  8,
        8,  8,  8,  8,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9, 10, 10,
       10, 10, 10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11, 11,
       11, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 13, 13, 13, 13, 13, 13,
       13, 13, 13, 13, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 15, 15, 15,
       15, 15, 15, 15, 15, 15, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
       17, 17, 17, 17, 17, 17, 17, 17, 17, 17])
```

  Where each number in the array represents the subject id. The first 9 elements are for subject number 0, the next are for subject 1 and so on… Hence, the model trains on 11 subjects and tests on the remaining subject. **At each fold, the test set is different.**

- The notebook is structured in a way to execute each method sequentially. First is Baseline, then filter, wrapper and embedded methods respectively. This allows the reader to compare the methods on the go (more information in the ipython notebook).
- To provide a clear and reproducible code, some functions were implemented:
    1. `get_scores`: returns important metrics: **Specificity**, **Sensitivity** and **AUC** . It takes as parameters: the **confusion matrix** and **predictions** (used by all methods).
    2. `predict`: returns the **predictions**, **false/true positive** rates for roc curves, **accuracy**, and **train/test scores.** It takes as parameters: **Feature dataset**, **labels**, **LeaveOneGrouptOut** cross validation and **groups** to split the folds**.** A default parameter called **model** is set to None. If a model is passed to the function, then the SGDClassifier fitted to the data uses the parameters of the given model. Otherwise, Grid Search cross validation is performed to find the hyperparameters of the classifier that fit the feature dataset best.
    3. `filter`: returns **reduced data** after filtering and **model parameters** that fit the dataset best. It takes as parameters: **Feature dataset**, **labels**, **LeaveOneGroupOut** cross validation, **filter type** and the **groups** to split the folds (used by Mutual Information and Anova methods).
- The experiment pipeline is conducted as follows:



- The diagram above explains clearly how the pipeline works, however for more clarification:
    - The baseline method uses all 432 features directly and is used as a basis of performance to compare all other methods on it.

- RFE and Lasso/Ridge Embedded methods directly use the sklearn functionalities and libraries to reduce the data and then follow the pipeline.
- Anova and Mutual Information filters however first send their library type to the function `filter` which reduces the data using the sklearn libraries. This has been done to remove redundant code and combine the methods into one function.
- `results_final` represents the table that contains all metrics from all methods used (Table 1).

- With all parameters tuned, the notebook was executed and provided the following results with the necessary metrics:

	Baseline	High Correlation Filter	Mutual Information Filter	Anova (f-Test) Filter	RFECV	Ridge	Lasso
Accuracy Mean	0.855556	0.872222	0.866667	0.905556	0.927778	0.877778	0.911111
Accuracy Std	0.125708	0.093128	0.105409	0.107869	0.080316	0.122726	0.080890
Specificity	0.855263	0.880000	0.878378	0.931507	0.913580	0.853659	0.900000
Sensitivity	0.855769	0.866667	0.858491	0.887850	0.939394	0.897959	0.920000
AUC Score	0.851250	0.867500	0.861250	0.900000	0.927500	0.877500	0.910000
Execution Time (s)	6.067074	2.572357	209.172318	24.790698	20.713762	1.327449	1.252481
Number of Features	432.000000	350.000000	360.000000	300.000000	161.000000	175.000000	136.000000

This table represents a good summary of the performances of each method used and comparing them with each other and against baseline.
But, how to choose the most suitable metric to use?

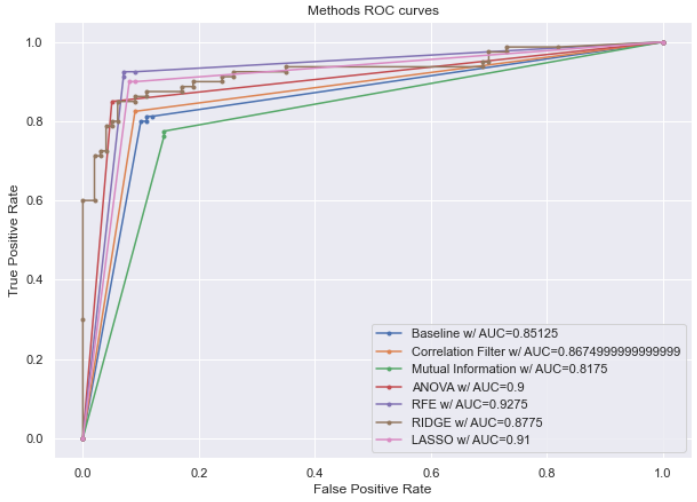
1. Accuracy is not a good choice since the classes are imbalanced. It will not be an appropriate representation of performance.
2. Specificity and Sensitivity are crucial to consider since, the aim of this study is to achieve a high sensitivity score with a slightly lower specificity score. The latter is due to the dataset being of a medical background, hence prioritizing sensitivity over specificity is a must. For example, it is always better to misclassify a healthy person as sick (lower specificity) than misclassifying a sick person as healthy (lower sensitivity).
3. The AUC score is the most important metric. Since the classes are imbalanced, it is good to have a measure of how good the classifier distinguishes between the positive and negative class. The higher the AUC the better the distinction and classification. Thus, the better the performance.

Hence, choosing a high AUC with a high sensitivity score is a good measurement for selecting an optimal method. The model will be assessed on how well it distinguishes between both classes and how correctly it classifies the positive class.
This table contradicts some of the expectations and reinforces others:

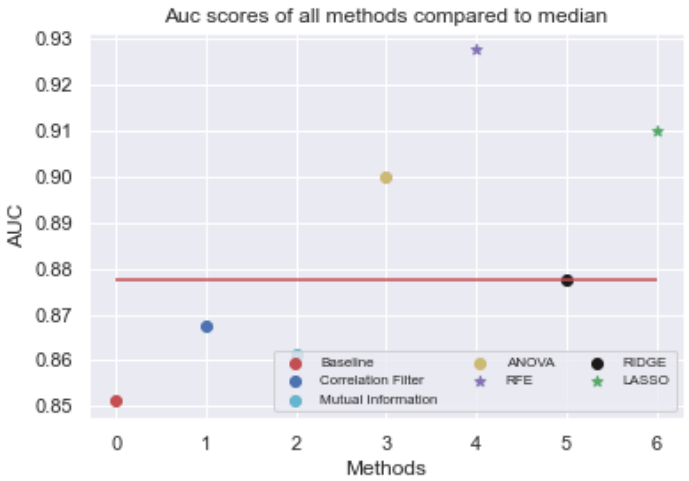
1. The Mutual Information Filter turned out to be computationally expensive and close to the baseline performance with higher features than expected. Failed the requirements of high AUC and large sensitivity (higher than specificity) scores.
2. Even with highly correlated features in the dataset, the high correlation filter did not increase performance or decrease the number of features as expected. Failed the requirements of high AUC and large sensitivity (higher than specificity) scores.
3. The Anova filter performed well, however the sensitivity score is lower than the specificity one which does not meet the set requirements. Moreover, for a filtering method, the execution time is not appropriate. The AUC score is higher than the previous methods and beats baseline with a better performance. Failed the requirements of large sensitivity (higher than specificity) scores.

4. The RFE method beats the expectations by scoring high on both the AUC and sensitivity scores. Moreover, it turned out to be faster than both the Anova and Mutual Information filters. Plus, it reduced the number of features to less than half the original size. Passed the requirements of high AUC and large sensitivity (higher than specificity) scores.
5. The Ridge method on the other hand presented acceptable results. It has fast execution time, high sensitivity scores and a slightly higher AUC score than the filters and baseline. Passed the requirements of high AUC and large sensitivity (higher than specificity) scores.
6. The Lasso technique reinforced the expectations by achieving short execution time, higher AUC scores than all filter, Ridge and baseline methods, and an excellent sensitivity score. Passed the requirements of high AUC and large sensitivity (higher than specificity) scores.

Further observations can be extracted from the following graph:



There are three methods left to choose from. The graph above represents the ROC curves of each method at different thresholds. This allows assessing the model's performance with sensitivity and false positive rate tradeoffs over a set of thresholds of all methods.
Based on the requirements set, a high tradeoff between TPR/FPR is appreciated. And since sensitivity is of the utmost importance, looking at the graphs above, the curves of choice need to be heading towards the upper left corner. Thus having a high sensitivity score and a high TPR/FPR tradeoff. This said, one can drop the Ridge method and only accept **Lasso** and **RFE** as models of optimal performance for this case study.



Based on the graph above, It is clear that the RFE and Lasso methods stand out in terms of performance where they both have scores higher than any other method.

Discussion:

Based on the results above, one could finalize the project and start testing on new subjects. However, there are more analysis factors that need to be taken into account that could result in one last optimum method.

The **Lasso** method has been chosen for the following reasons:

1- Execution time is much faster than RFE:

- The RFE method took approximately 20 seconds to execute whilst the Lasso one took only 1.25 seconds, being the least among all methods. Time is an important metric because what if the dataset was much larger (e.g. greater than 100k samples) ? Then the RFE would take ages to compute while the Lasso will have an appropriate execution time.

2- Embedded methods have the higher advantage:

- The lasso method of embedded nature is a better choice over the RFE method since it interacts with features such as wrapper methods, is faster and more accurate than filtering methods (evidence in Table 1) and it extracts features while the model is training.

3- Lower features than RFE:

- The Lasso reduced the dataset down to 136 features while the RFE method worked with 161 features. This difference of 25 features is important since the Lasso method was able to still maintain high performance while reducing the data into a simpler version with better description. However, the RFE might expect a lower performance if its features have been reduced to 136.

4- Much less prone to overfit:

- Due to the regularization found in the Lasso model, the latter is more robust against overfitting than RFE. This can clearly be shown by the following output:

```
Lasso Train mean score: 0.9316993464052289
Lasso Test mean score: 0.911111111111111
```

```
RFE Train mean score: 0.9568627450980389
RFE Test mean score: 0.9277777777777779
```

- First of all, the ratio train/test scores for the Lasso method is lower than the RFE one, which shows that the model generalizes better with the Lasso method. Second of all, the graphs show how the RFE train scores have less oscillations than the Lasso one which are signs that the model is more prone to overfit while using RFE.

With much evidence considered, the Lasso Embedded feature reduction technique is the ideal solution for this Case Study. It has shown to be fast, maintainable, robust and results in high performance (both AUC and sensitivity scores). Given medical data, it is important to consider all methods and narrow it down to one optimal approach. Hence why this experiment is structured by evaluating all examples alongside grid search and cross validation to ensure best performances from all described pipelines. However, some improvements can be taken into account. Most importantly, more data can be gathered and extracted to ensure the model generalizes better to

new test subjects. By adding more subjects to the experiment, cross validation can be adapted to more groups and enhance the model’s performance while degrading its generalization error.