Report: Smart Steel feature ranking
Author: Tung Dinh
Stuttgart 01.06.2021

Introduction
Why do we need to rank features?

In the ML system design, pre-processing the dataset is one of the most important steps, in which we try to grasp as many as possible insights from the dataset. Understanding the problem statement of the pipeline, we can get our hands on the feature ranking. Feature ranking could be seen as a part of feature selection. The fact is that not all features are useful for the final task, and adding or removing one feature could change the accuracy of the final model. Choosing the correct features based on their ranks could bring some advantages:

- Faster training model
- Improve accuracy
- Save time and space
- Reduce overfitting

Although tree-based model is the most common among all, we will go through some methods to rank features:

- Filter method
- Tree-based or intrinsic method
- Additional ML method
- Wrapper method

The main problem with the dataset is testing the ranked features.
In order to understand the ranking models and to pick the best performing one, one final (neutral) model should try to predict the classes of sensors based only on the ranked sensors.
Therefore, once we have the ranked sensors, it is very difficult to argue that the ranking is totally correct.

Components

- task_data.csv
- task.txt
- README.md
- svmranker.py: a class built from soft SVM, including other functions for the ease of computation.
- treeranker.py: a class built from tree-model, including with other functions for the ease of computation.
- my_notebook.ipynb: includes the pipeline of the process
- tree_ranking.txt: ranked features from tree model
- svm_ranking.txt: ranked features from soft SVM

- ReportSmartSteel.pdf: the final report.

In each ranker, we have:
- dataLoader: create features and labels, split train-test sets
- Train: perform fit model
- Permutation: verify the ranked feature with permutation importance, it can also be used alone, by adjusting the train-test split to 1
- Report: plot the ranking and returns ranking as list

Experiments

We attempt to pick one model for each method. For the filter method, the very basic one is Correlation matrix, XGBClassifier for tree-based, SVM for ML model and Permutation Importance for wrapper method. Also, to understand if the model performs well under the ranking task, we are tempted to use a wrapper method to verify it. To be specific, we will try to use 80% of the dataset for ranking with tree-based and ML models, then we will use 20% test set to verify if the ranking brings us some insights or not. Since the wrapper method depends solely on the target model (tree-based and ML), it might not be a good idea to compare them unless we have a standard model for the classification task.

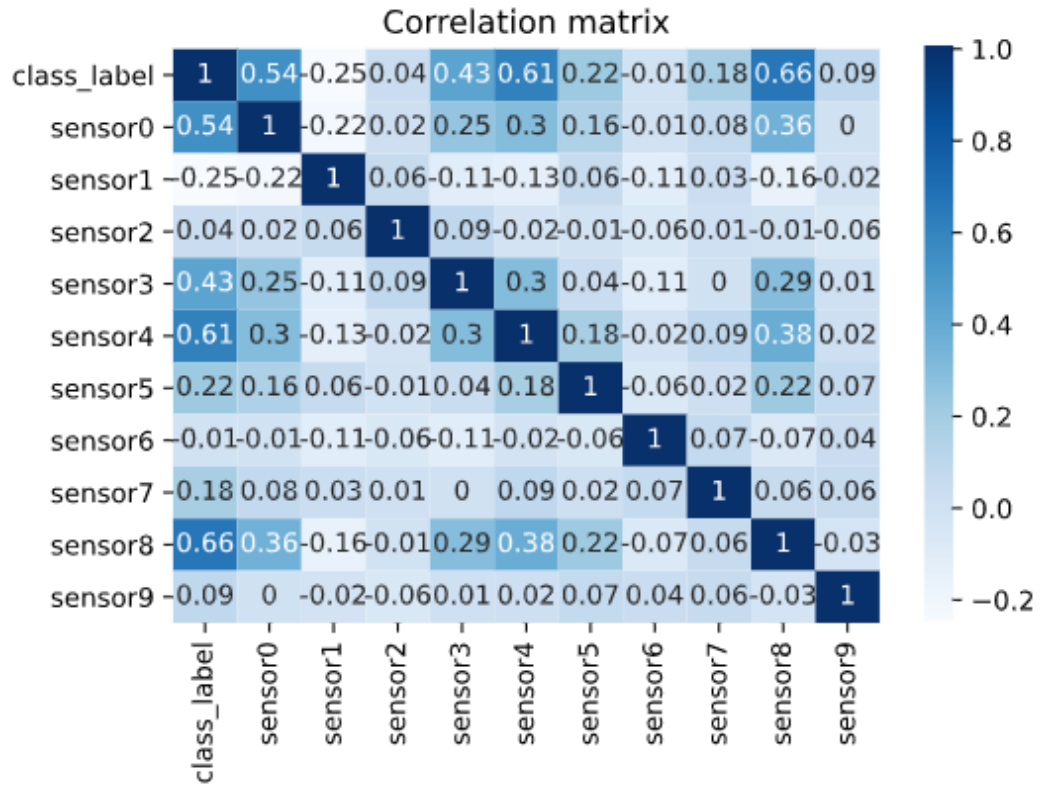
At the end, the author attempted to find the meaning of ranking features, therefore, he used a very simple logistic regression to classify the dataset after ranking. It is the author's idea to multiply the ranking score to the dataset as before feeding to the logistic regression.

Result

Correlation matrix:

It is designed to work best with numerical output, however, we still use it here to give us a general view into our dataset.

They the the general methods, which we expect to pick a subset of features. Using correlation matrix is the most basic filter method. We can see that all 10 sensors show the independence to each other with small correlation values. On the class column, we can see the top three sensors are sensor 8, 4, and 0. Naively, this means the value of the class column depends on 3 sensors more than others.



Advantages:

- Correlation method is simple, easy to calculate, and shows the positive/ negative/ neutral relationship between features.
- It can give an overall look on the relationship of features, and we can build our predictor on its hypotheses.

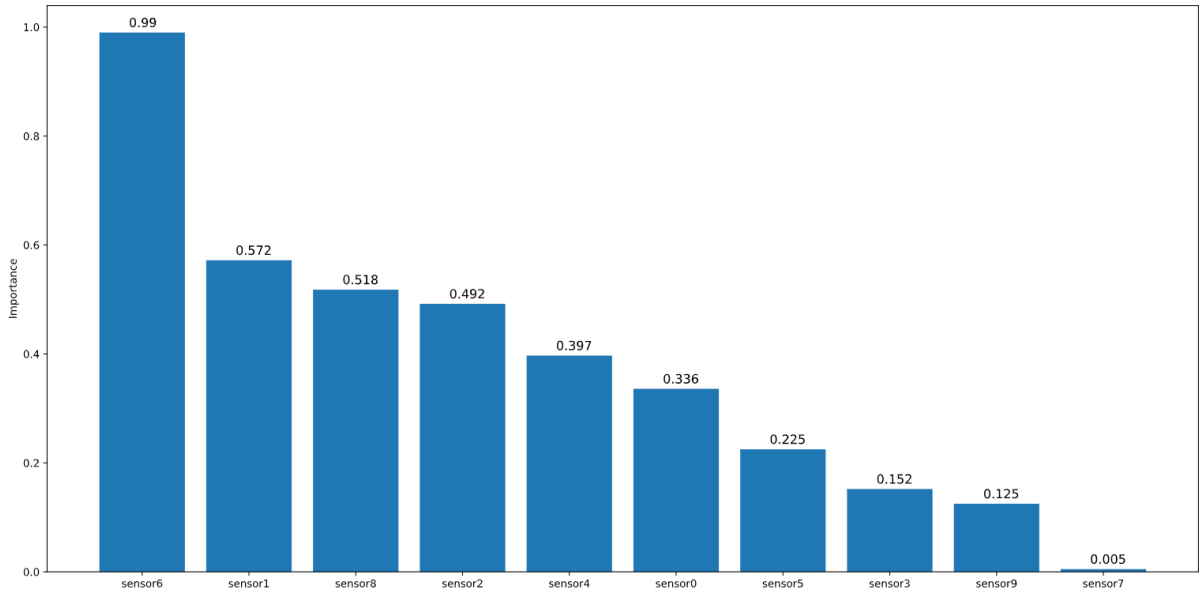
Disadvantages:

- Its assumption is only on linear combination of features, so it is not correct for higher degree of combination.
- Drawing conclusions with the correlation matrix could be also incorrect, it could not tell the causality, and it does not necessarily mean: high correlation value is equal to higher degree of linear combination between the two.
- We can incorrectly assume the relationship between features since the dataset is too small (with small dataset, we can not conclude about its distribution)
- High computational cost

This method is applicable for larger dataset, both in number of samples and number of features. And since it has high computational cost O(#samples * #features^2), it is not scalable.

Other methods, which could be considered here are ANOVA and Predictive Power Score (PPS).
Advantage of ANOVA: it is also simple to calculate, and is used to test the interaction hypotheses between features.
Disadvantage: we should have a standardized dataset to work on, meaning a uniform distributed dataset, with the same mean and standard variance. It is **not** the case in this dataset (please check notebook at df.describe()).

Additional: ranking result with PPS:



With PPS, sensors 6, 1 and 8 are more important than other sensors.
Advantage of PPS: It can figure out the non-linear relationship between features, therefore learning hypotheses is good.
Disadvantage: come from its high computational cost, one should not make any conclusion based on the score alone, it should be enhanced by forward and backward selection (wrapper method)

https://github.com/8080labs/ppscore

XGBClassifier:
With gradient boosting the XGBClassifier performs classification on plit-based and gain-based measure. According to the paper "A working guide to boosted regression tree" by J.Elith et al: "*The measures are based on the number of times a variable is selected for splitting, weighted by the squared improvement to the model as a result of each split, and averaged over all trees".* In simple words, in order to learn a feature importance, we learn the information-gain based on the parent and child impurities (mean decrease of impurities) and we aim for the feature that make the maximal improvement in splitting.

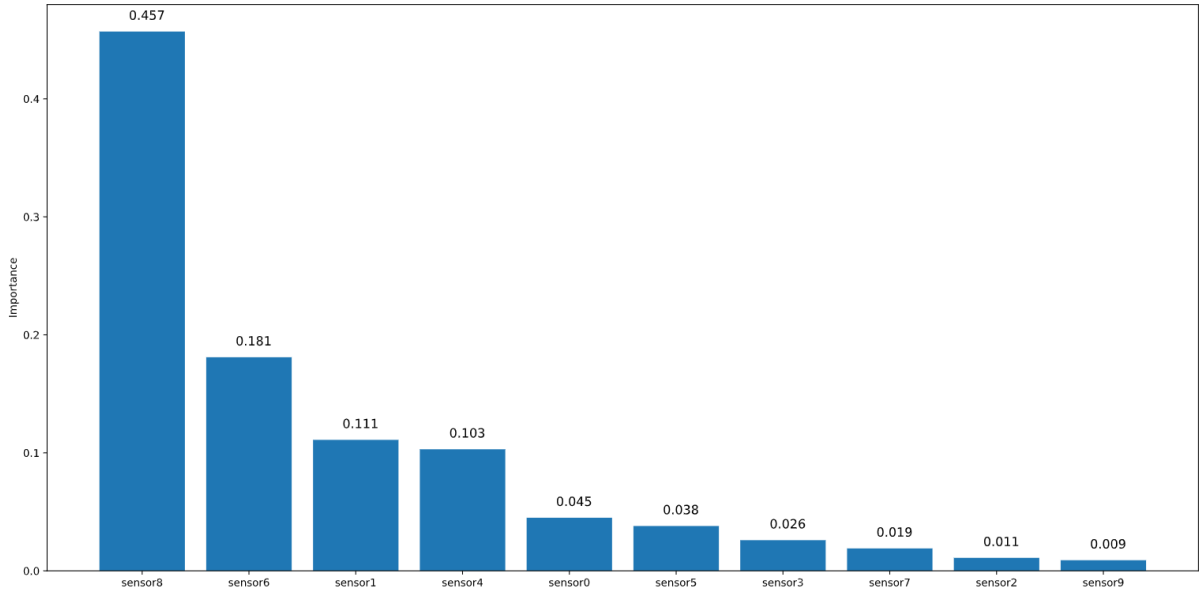
The model shows the importance of features: the most important ones are sensors 8, 6 and 4.

Advantages

- It is immune to multicollinearity of the dataset
- It is the perfect algo for this situation, since we only have 2 classes and small dataset of 400 samples, it also calculates fast
- It can avoid overfitting by pruning trees and using regularization L1.
- It can work with a sparse dataset.

Disadvantages:

- Its downside is when we have a more complicated dataset, with more classes, more features, its performance decreases due to overfitting.
- It also depends on outliners to make decisions, it is due to its nature to try to correct each predecessor tree.
- It uses a sequential process to train trees, so it is time consuming.



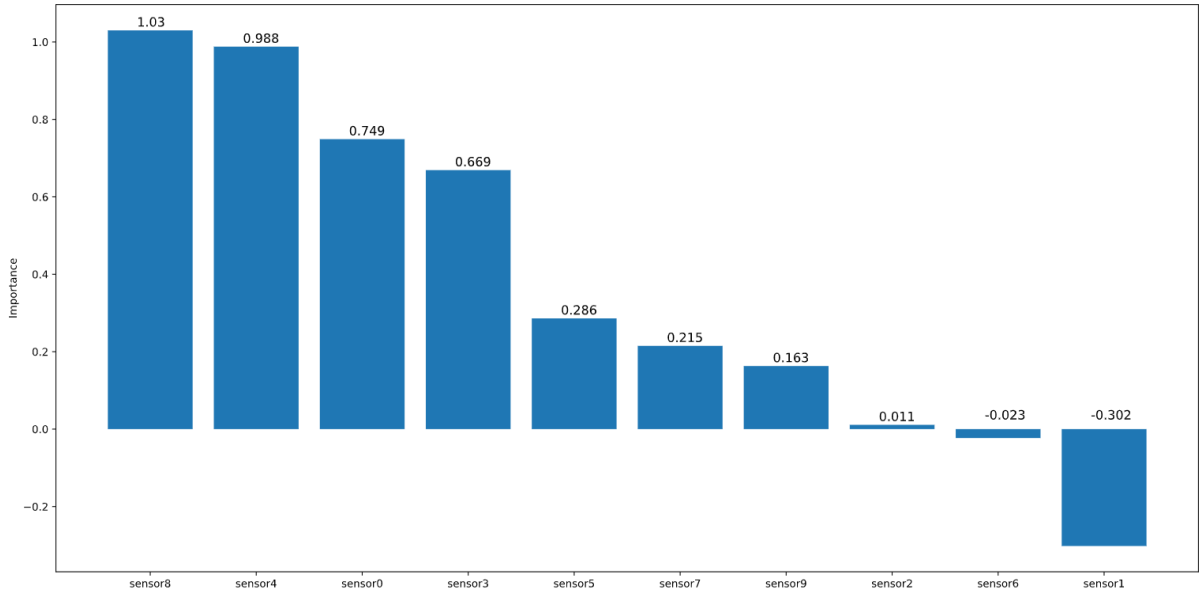
Verification with wrapper method: there is a small conflict in ranking with the dataset, in which the most important ones are sensors 6, 8 and 4. In order to understand this further, and in order to be fair, we should dive into what makes our train and test sets. To be fair, the train and test sets must follow the same distribution. In this case, we random shuffle the dataset, and the problem could be: in our test set, the number of sample depends on feature 6 higher than feature 8. We can see that, with XGBClassifier, we can tell that sensors 8, 6 and 4 are more important than other in classifying the class using XGBClassifier.

XGBClassifier uses bootstrapping idea to generate its dataset, the more samples + the more features => the more datasets, hence it is not scalable in terms of number of samples and number of features.

Another method to be considered here: Random forest tree, it is also included in the code, for the task of calculating feature importances, they share similarity in the idea behind. One point here is it is a bagging model, so it can be better than XGBClassifier in avoiding overfitting. Comparison between the two methods is controversial.

Other ML method

SVM is powerful in classification, and using soft SVM, we try to classify our dataset, based on the 10 sensors. The sensors are ranked on their coefficients.



Verification with wrapper method: there is a small conflict in ranking with the dataset, in which the most important ones are sensors 8, 4 and 0. This problem could be explained similarly as the above XGBClassifier. We can see that, with soft SVM, we can tell that sensors 8, 0 and 4 are more important than other in classifying the class using soft SVM.

Advantages:

- It is relatively powerful in classification tasks, even in case of not-so-clear margin, we can use soft SVM.
- It can avoid overfitting with margin trade off.
- It works well with high dimensional data.

- It is shown to be memory efficient also.

Disadvantages:

- With a larger dataset, we might have a computational problem.
- Also, the output of SVM depends on our choice of kernels.
- And its output is not a probability, so it is not easy to interpret for classification problems.

SVM uses kernel trick to work around with high dimensional dataset, hence it is scalable in term of number of features. However, in term of number of samples, it is not the case, because increasing the number of samples can lead to overlap between regions due to outliers, and the margin is not easy to find.

Wrapper method:

Advantages:

- This method works well in destroying each feature and discovering the accuracy of the model in each turn.
- It could be ideal to verify the ranked feature regarding to the host model.

Disadvantages:

- Since it is a repeated process, time is a factor to be considered, and it might not be a good idea to apply for a dataset with many features.

Wrapper method is an iterative method in interferencing the features, assume the time complexity is O(#feature), it is hence not scalable to the number of features. To the number of samples, I believe increasing the number of sampling would not corrupt the model.

Another wrapper method is Recursive Feature Elimination. This method returns the accuracy of the corresponding models without each feature. It also shares the similarities in advantages and disadvantages. Other ML method: Lasso, with L1, it can punish features, therefore, it is well known for the ability to select good features.

Problems

During working, one mistake encountered is: data is not shuffled. The output of the host model and wrapper method got a huge conflict.

Conclusion

We have been through some methods in all the groups. It is difficult to conclude which ranking is better than others, however, from all the ranking results, we can tell that sensor 8 is important to classify classes.

Additional

	Train accuracy	**Test accuracy**
Default	0.9390681003584229	0.9338842975206612
XGBClassifier	**0.8602150537634409**	0.9256198347107438
Soft SVM	0.9247311827956989	0.9173553719008265

In order to verify if the ranking is useful to the final task of classification, the author idea is to multiply the ranking to the values of sensors, then feed their values to a neutral classifier. The neutral classifier should not be a NN, since with NN, it will not require feature engineering, and the author believes it is fairer with a very basic ML model. So we will go with a logistic regression. With keras, it can be implemented as 2-layer NN. The output layer will have 2 neurons. The model is then tuned with adam optimizer, L2 regularization, the accuracy of all the PPS ranked methods turn to be 1 and 0 error. Since it brings no insights to the ranking systems, we will stop here. The author believes the problem might comes from the habit of tuninng the logistic model.

The next attempt is with the built-in LogisticRegression of sklearn lib, since building one from scratch is time-consuming. And we can find something interesting here :). Without ranking, the logistic regression performs better. It could mean that the dataset is easily separated. Also multiplying the rank back to the value leads to the decrease in their values, and might decrease the level of importance in logistic regression when multiplication with weight matrix and lead to wrong decision.

It can be seen that logistic regression outputs a more convincing result for the soft SVM than XGBClassifier. Test accuracy of XGBClassifier > train accuracy, and it could come from the bad train-test split, or the linear regression is underfitting, either way, the verification of soft SVM brings more insights in this case.
Thank you.