



LOFO (Leave One Feature Out) Importance calculates the importances of a set of features based on a metric of choice, for a model of choice, by iteratively removing each feature from the set, and evaluating the performance of the model, with a validation scheme of choice, based on the chosen metric.

LOFO first evaluates the performance of the model with all the input features included, then iteratively removes one feature at a time, retrain the model, and evaluates its performance on a validation set. The mean and standard deviation (across the folds) of the importance of each feature is then reported.

If a model is not passed as an argument to LOFO Importance, it will run LightGBM as a default model.

Install

LOFO Importance can be installed using

```
1 pip install lofo-importance
```

Advantages of LOFO Importance

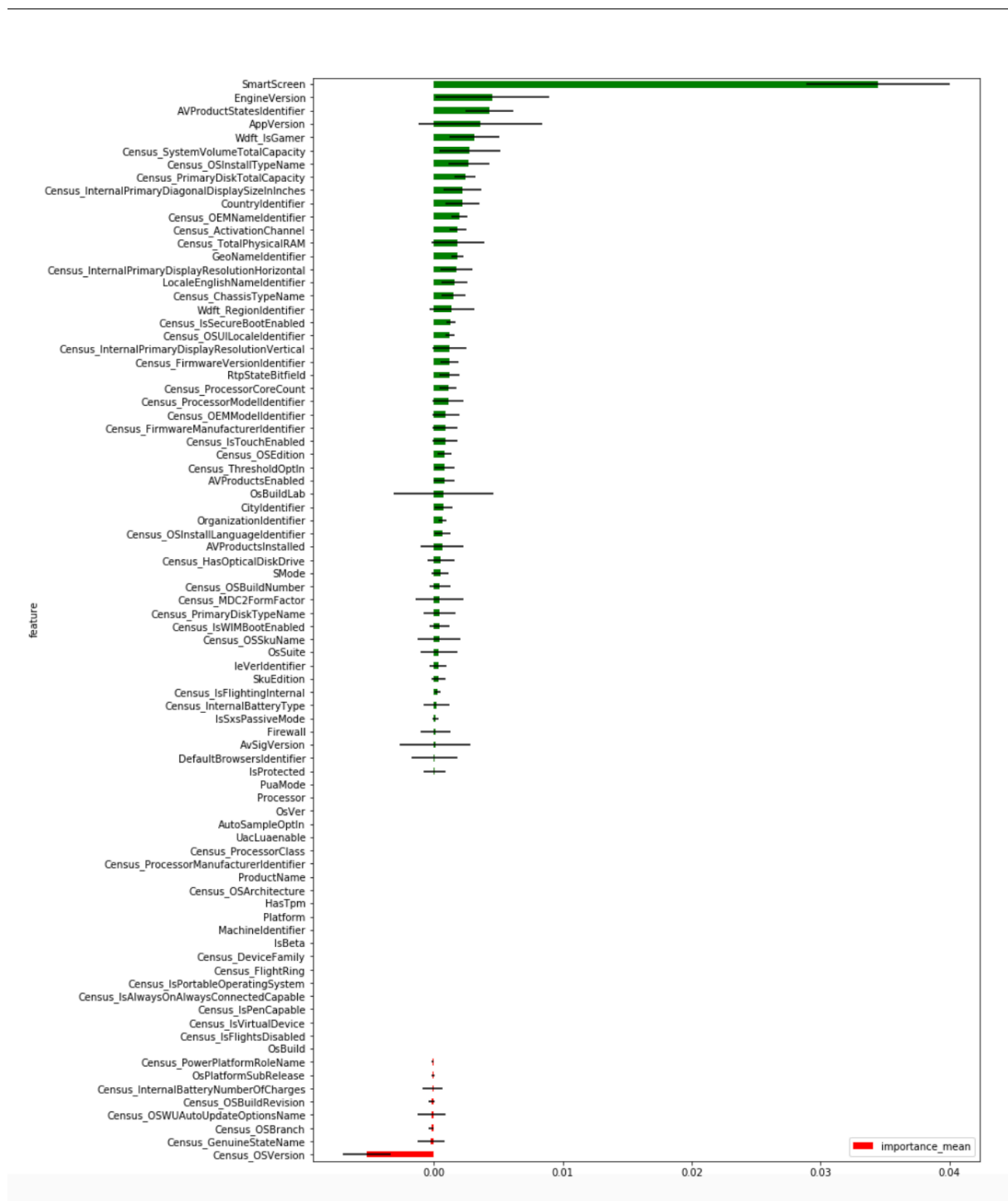
LOFO has several advantages compared to other importance types:

- It does not favor granular features
- It generalises well to unseen test sets
- It is model agnostic
- It gives negative importance to features that hurt performance upon inclusion
- It can group the features. Especially useful for high dimensional features like TFIDF or OHE features.
- It can automatically group highly correlated features to avoid underestimating their importance.

Example on Kaggle's Microsoft Malware Prediction Competition

In this Kaggle competition, Microsoft provides a malware dataset to predict whether or not a machine will soon be hit with malware. One of the features, `Centos_OSVersion` is very predictive on the training set, since some OS versions are probably more prone to bugs and failures than others. However, upon splitting the data out of time, we obtain validation sets with OS versions that have not occurred in the training set. Therefore, the model will not have learned the relationship between the target and this seasonal feature. By evaluating this feature's importance using other importance types, `Centos_OSVersion` seems to have high importance, because its importance was evaluated using only the training set. However, LOFO Importance depends on a validation scheme, so it will not only give this feature low importance, but even negative importance.

```
1  import pandas as pd
2  from sklearn.model_selection import KFold
3  from lofo import LOFOImportance, Dataset, plot_importance
4  %matplotlib inline
5
6  # import data
7  train_df = pd.read_csv("../input/train.csv", dtype=dtypes)
8
9  # extract a sample of the data
10 sample_df = train_df.sample(frac=0.01, random_state=0)
11 sample_df.sort_values("AvSigVersion", inplace=True) # Sort by time for
    time split validation
12
13 # define the validation scheme
14 cv = KFold(n_splits=4, shuffle=False, random_state=None) # Don't
    shuffle to keep the time split split validation
15
16 # define the binary target and the features
17 dataset = Dataset(df=sample_df, target="HasDetections", features=[col
    for col in train_df.columns if col != "HasDetections"])
18
19 # define the validation scheme and scorer. The default model is
    LightGBM
20 lofo_imp = LOFOImportance(dataset, cv=cv, scoring="roc_auc")
21
22 # get the mean and standard deviation of the importances in pandas
    format
23 importance_df = lofo_imp.get_importance()
24
25 # plot the means and standard deviations of the importances
26 plot_importance(importance_df, figsize=(12, 20))
```

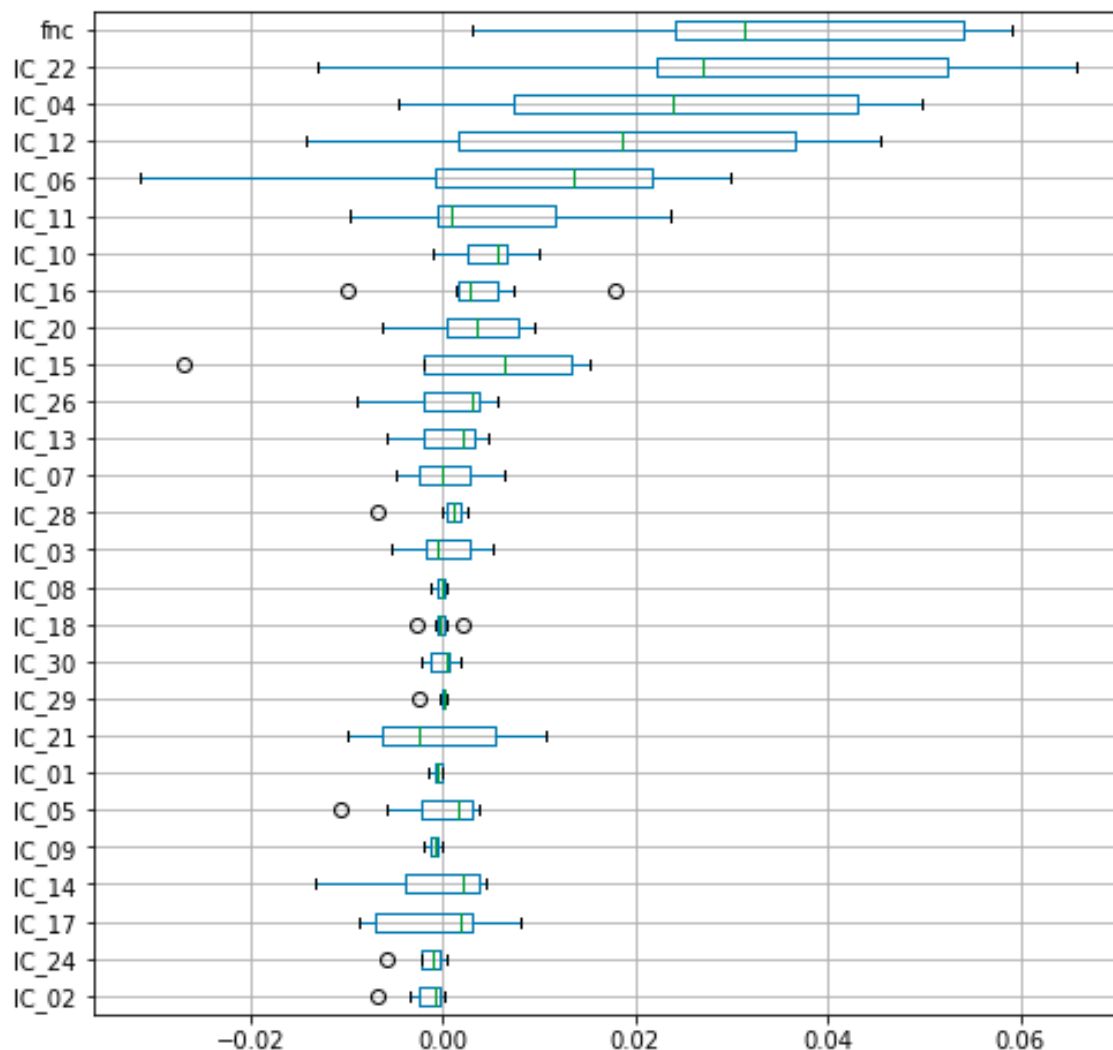


Another Example: Kaggle's TReNDS Competition

In this Kaggle competition, participants are asked to predict some cognitive properties of patients. Independent component features (IC) from sMRI and very high dimensional correlation features (FNC)

from 3D fMRIs are provided. LOFO can group the fMRI correlation features into one.

```
1 def get_lofo_importance(target):
2     cv = KFold(n_splits=7, shuffle=True, random_state=17)
3
4     dataset = Dataset(df=df[df[target].notnull()], target=target,
5                       features=loading_features,
6                           feature_groups={"fnc": df[df[target].notnull()][
7                               fnc_features].values
8                           })
9
10    model = Ridge(alpha=0.01)
11    lofo_imp = LOFOImportance(dataset, cv=cv, scoring="
12        neg_mean_absolute_error", model=model)
13
14    return lofo_imp.get_importance()
15
16 plot_importance(get_lofo_importance(target="domain1_var1"), figsize=(8,
17    8), kind="box")
```



Flofo Importance

If running the LOFO Importance package is too time-costly for you, you can use Fast LOFO. Fast LOFO, or FLOFO takes, as inputs, an already trained model and a validation set, and does a pseudo-random permutation on the values of each feature, one by one, then uses the trained model to make predictions on the validation set. The mean of the FLOFO importance is then the difference in the performance of the model on the validation set over several randomised permutations. The difference between FLOFO importance and permutation importance is that the permutations on a feature's values are done within groups, where groups are obtained by grouping the validation set by $k=2$ features. These k features are chosen at random $n=10$ times, and the mean and standard deviation of the FLOFO importance are calculated based on these n runs. The reason this grouping makes the measure of

importance better is that permuting a feature's value is no longer completely random. In fact, the permutations are done within groups of similar samples, so the permutations are equivalent to noising the samples. This ensures that:

- The permuted feature values are very unlikely to be replaced by unrealistic values.
- A feature that is predictable by features among the chosen $n \times k$ features will be replaced by very similar values during permutation. Therefore, it will only slightly affect the model performance (and will yield a small FLOFO importance). This solves the correlated feature overestimation problem.