Modelling

What are we going to learn today?

- Optimal clustering
- Feature Importance
  - With Substitution Effects
  - Without Substitution Effects
- Hyper-parameter Tuning with Cross-Validation
  - Grid Search
  - Randomized Search
  - Scoring

Optimal Clustering

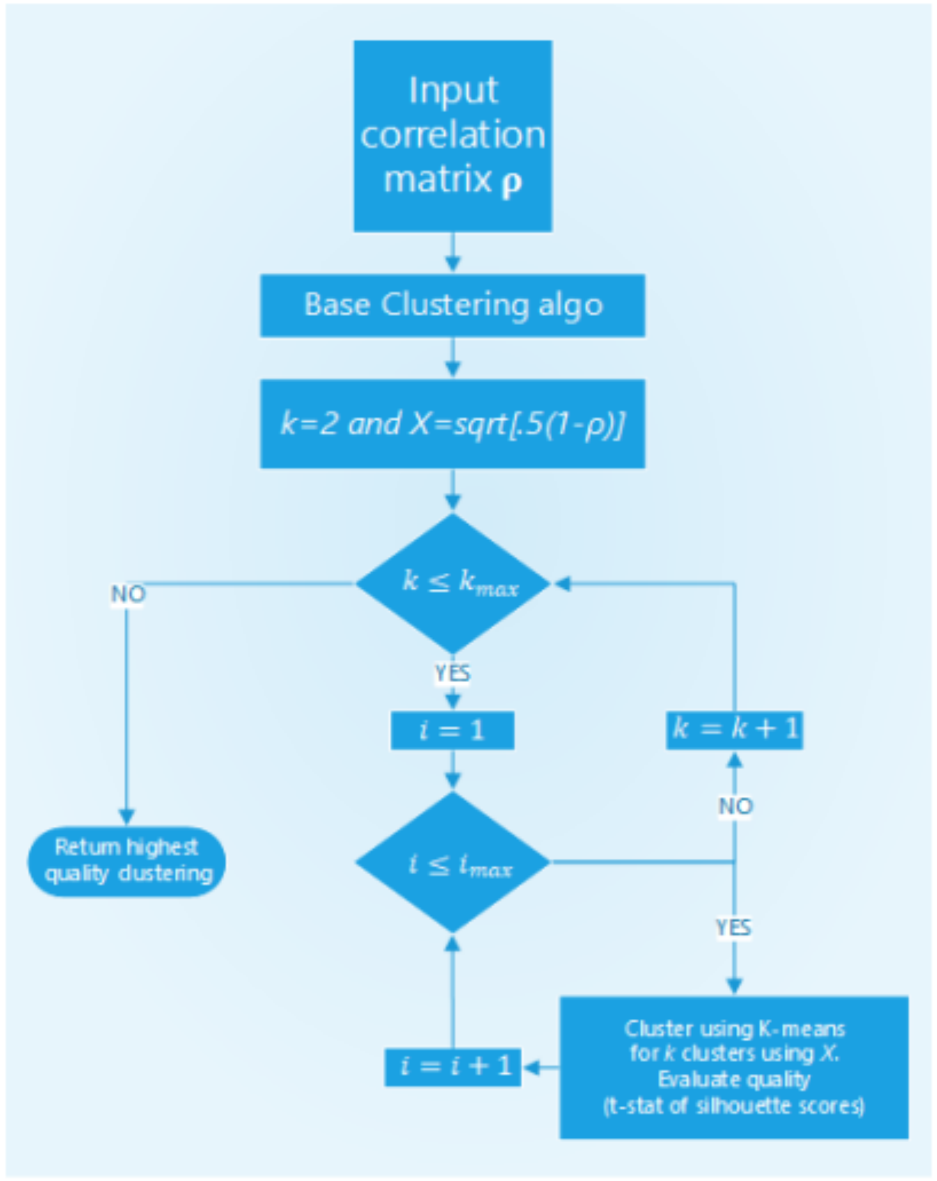
The ONC Algorithm

- Clustering steps:
  1. Base level clustering
  2. Recursive improvements to clustering

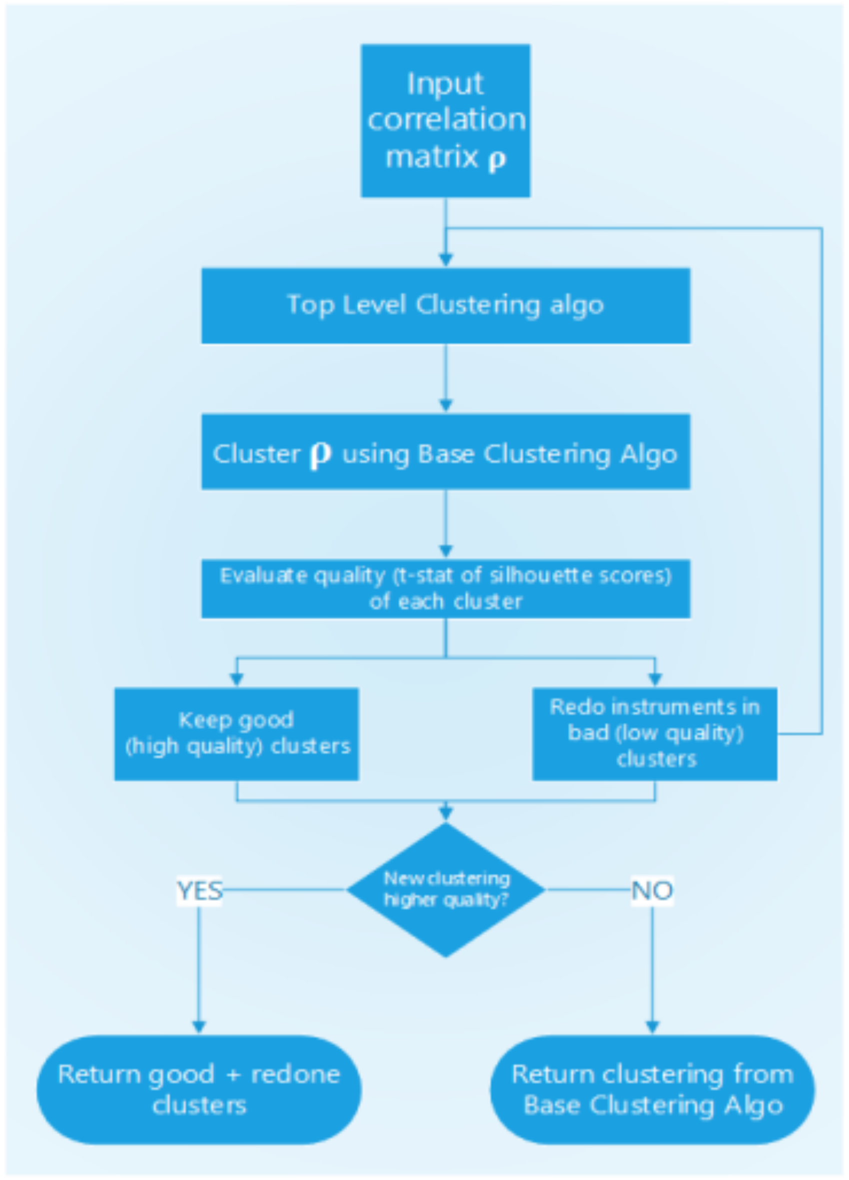
- Base Clustering:
  1. For each k in $2, \dots, N-1$, and l in $1, \dots, n_{init}$:
     - Apply Kmeans to extract k clusters using distance $\widetilde{D}$
     - Evaluate silhouette scores S_n, $n = 1, \dots, N$, for the clustering
     - Evaluate quality score $q_{k,l} = \frac{\mathrm{E}[S_n]}{\sqrt{\mathrm{V}[S_n]}}$
  2. Choose optimal clustering, with $K = \mathrm{argmax}_k \left\{ \max_l \{ q_{k,l} \} \right\}$

- Recursive improvement to clustering:
  1. Run Base Clustering to derive K
  2. Evaluate quality score q_k for each cluster $k = 1, \dots, K$
  3. Fix clusters k where $q_k \geq \mathrm{E}[q_k]$
     - Recursively rerun Base Clustering for rest of clusters
     - Keep new clustering only if re-clustering improves average quality score

Structure of the ONC Algorithm


Input correlation matrix ρ
Base Clustering algo
k=2 and X=sqrt[.5(1-ρ)]
$k \leq k_{max}$
NO
YES
$i = 1$
$k = k + 1$
NO
Return highest quality clustering
$i \leq i_{max}$
YES
$i = i + 1$
Cluster using K-means for k clusters using X. Evaluate quality (t-stat of silhouette scores)

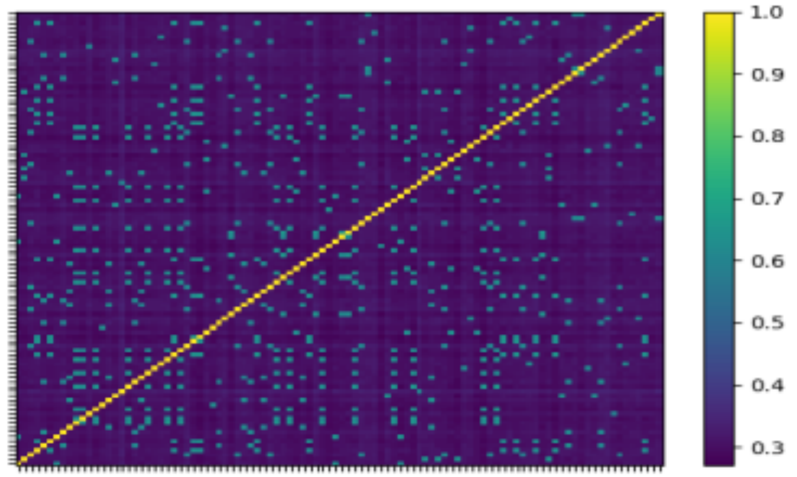
Base clustering


Input correlation matrix ρ
Top Level Clustering algo
Cluster ρ using Base Clustering Algo
Evaluate quality (t-stat of silhouette scores) of each cluster
Keep good (high quality) clusters
Redo instruments in bad (low quality) clusters
New clustering higher quality?
YES
NO
Return good + redone clusters
Return clustering from Base Clustering Algo

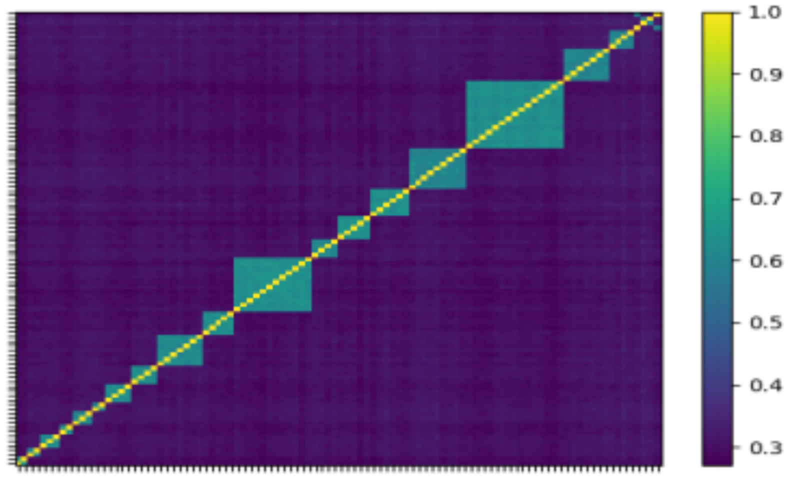
Higher-level clustering

Numerical Example

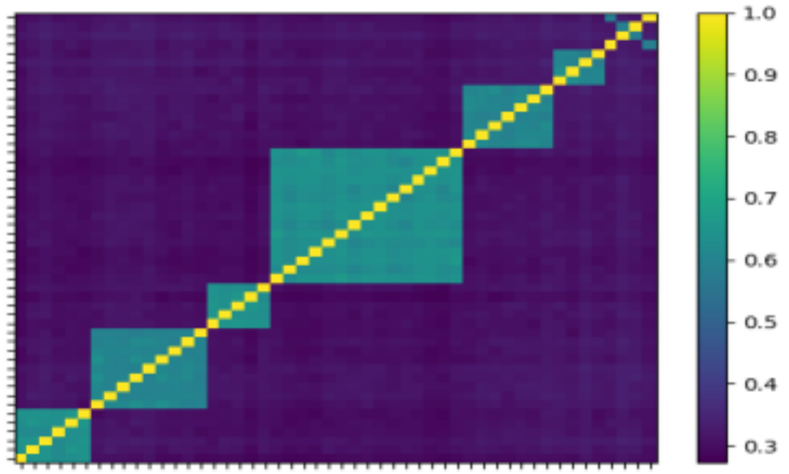
Initial Scrambled Random Block Covariance



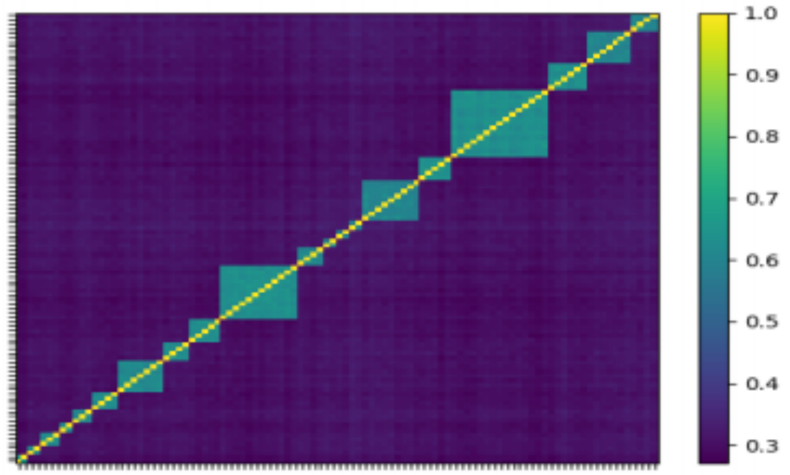
After Base Clustering



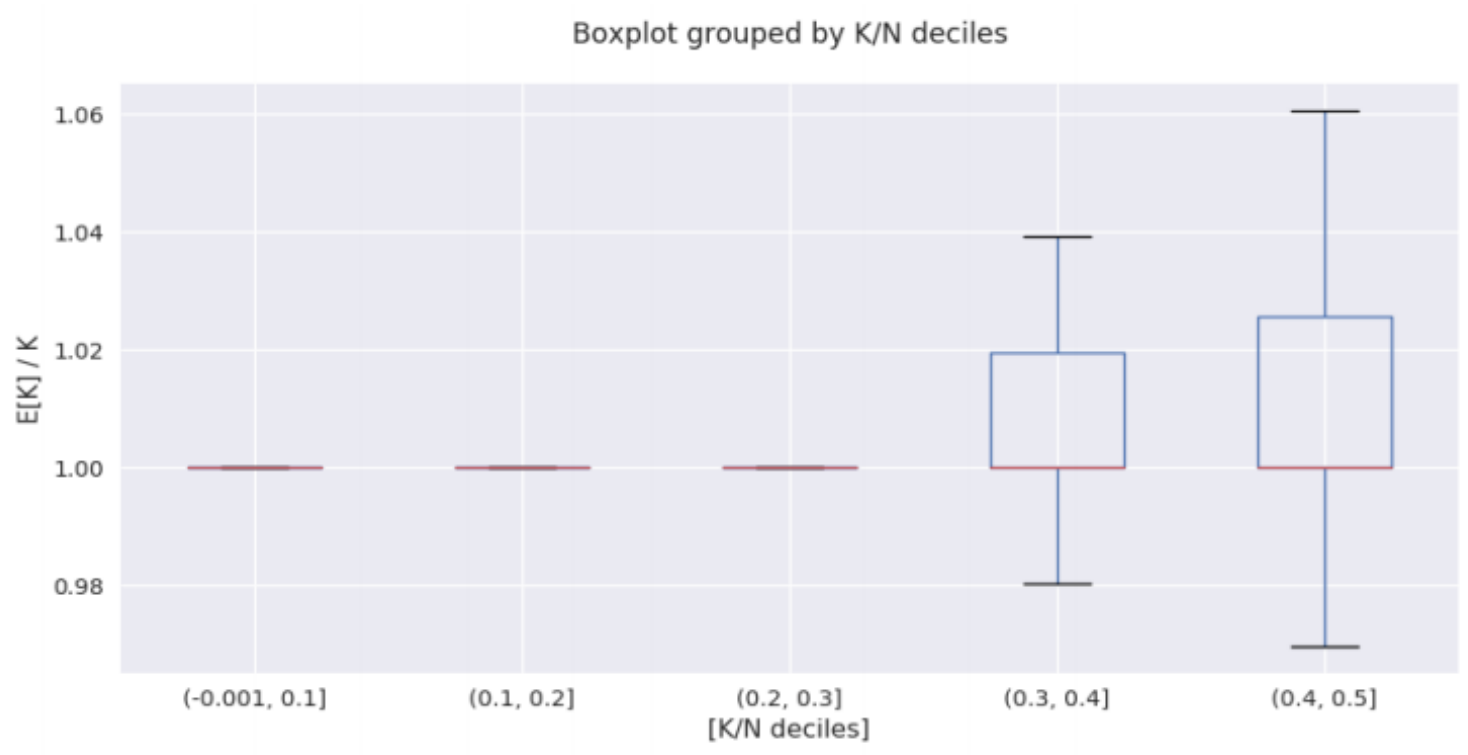
Recursively Re-evaluate Clustering



Final Clustering



Testing ONC's AI



- Create a random block covariance matrix
    - Size NxN
    - K block of random size
- Add global noise
- Run Clustering
- Compare expected cluster count K to number of estimated clusters

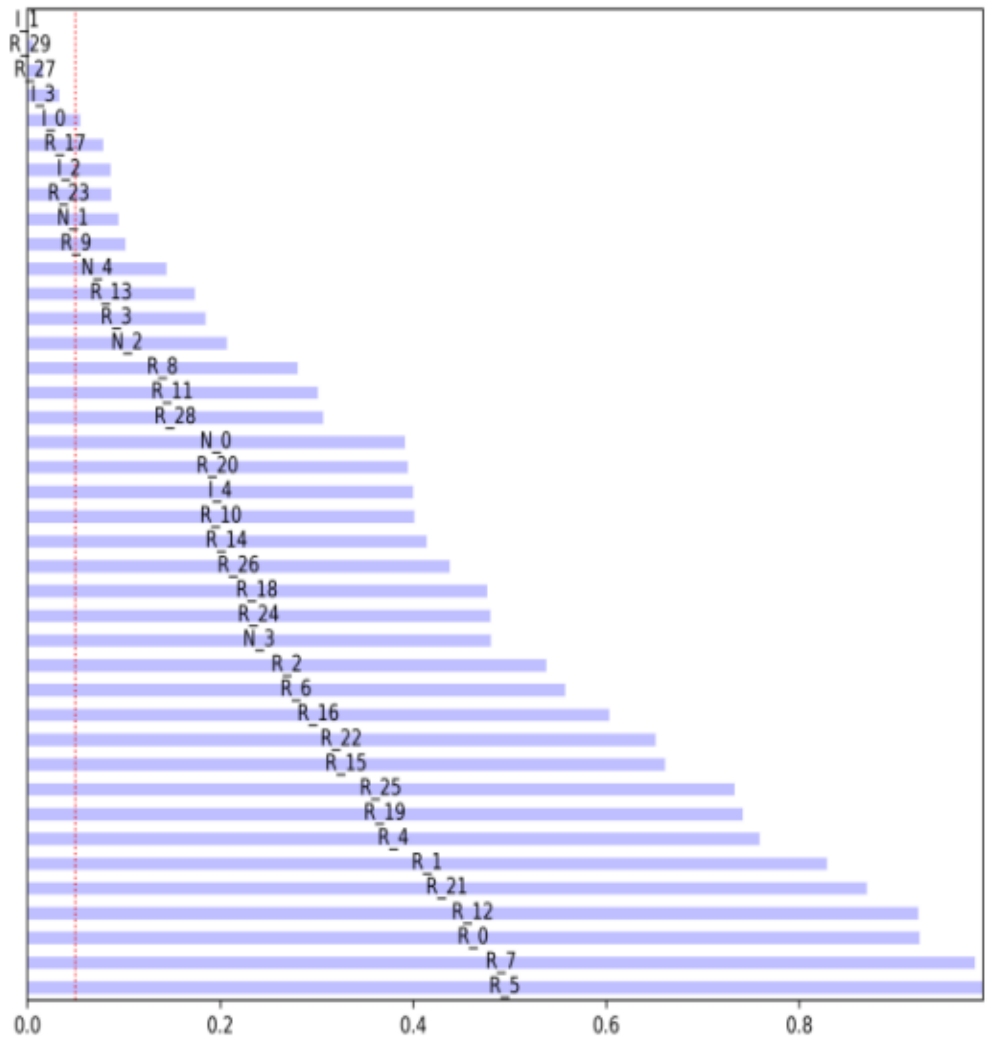
The boxplots show the results from these simulations. In particular, for $\frac{K}{N}$ in a given decile, we display the boxplot of the ratio of K predicted by the clustering to the actual $\mathrm{E}[K]$ predicted by clustering. Ideally, this ratio should be near 1. Simulations confirm that this clustering procedure is effective.

Feature Importance

The Importance of Feature Importance

- **Backtesting is not a research tool**
  - It is trivial to fit an ML algorithm to maximize performance in a backtest.
- **Feature Importance is a research tool**
  - We should always try to understand what drives the reason a ML algorithm.
  - In particular, in the context of financial markets, ask yourself: *"What is the risk-based or behavioral reason that explains that a market participant will systematically lose money to my advantage?"*
- Feature Importance methods can largely be classified depending on whether they control for substitution effects or not.
- A substitution effect takes place when the estimated importance of one feature is reduced by the presence of other related features.
- Substitution effects are the ML analogue of what the statistics and econometrics literature calls "multi-collinearity."
- One way to address linear substitution effects is to apply PCA on the raw features, and then perform the feature importance analysis on the orthogonal features.
- A better approach is to cluster the raw features and conduct the analysis at the cluster level.

A Numerical Example



Consider a binary random classification problem composed of 40 features, where 5 are informative, 30 are redundant, and 5 are noise.

Informative features (marked with the "I_" prefix) are those used to generate labels. **Redundant features** (marked with the "R_" prefix) are those that are formed by adding Gaussian noise to a randomly chosen informative feature. **Noise features** (marked with the "N_" prefix) are those that are not used to generate labels.

This plot shows the *p*-values that result from a logit regression on those features. The horizontal bars report the *p*-values, and the vertical dash line marks the 5% significance level. Only four out of the 35 non-noise features are deemed statistically significant: I_1, R_29, R_27, I_3. Noise features are ranked as relatively important (9, 11, 14, 18, and 26). Fourteen of the features ranked as least important are not noise.

In short, these *p*-values misrepresent the ground truth.

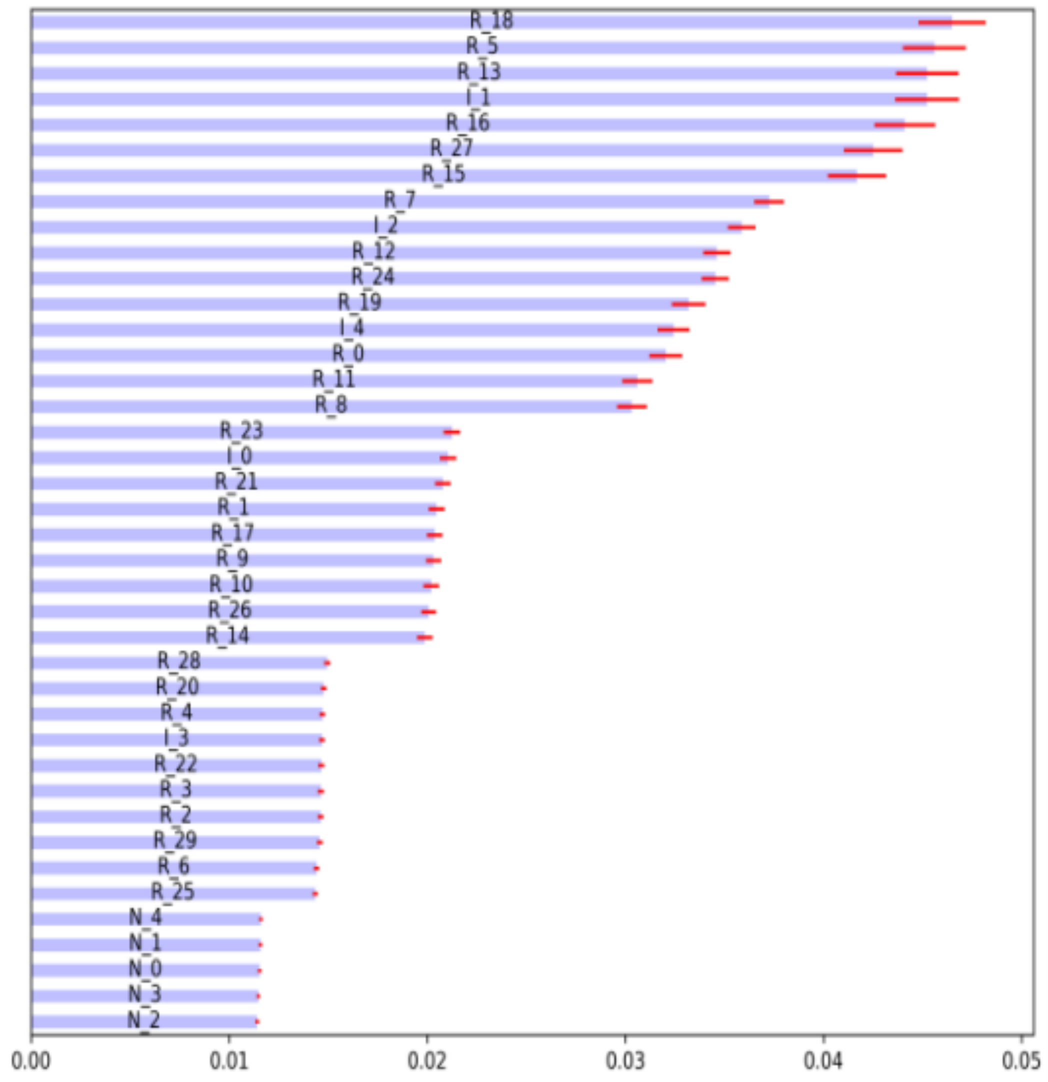
Mean Decrease Impurity (1/2)

Mean decrease impurity (MDI) is a fast, explanatory-importance (in-sample, IS) method specific to tree-based classifiers, like RF. At each node of each decision tree, the selected feature splits the subset it received in such a way that impurity is decreased. Therefore, we can derive for each decision tree how much of the overall impurity decrease can be assigned to each feature. And given that we have a forest of trees, we can average those values across all estimators and rank the features accordingly.

There are some important considerations you must keep in mind when working with MDI:

- By construction, MDI has the nice property that feature importances add up to 1, and every feature importance is bounded between 0 and 1.
- MDI dilutes the importance of substitute features, because of their interchangeability: The importance of two identical features will be halved, as they are randomly chosen with equal probability.
- MDI cannot be generalized to other non tree-based classifiers.
- Masking effects take place when some features are systematically ignored by tree-based classifiers in favor of others. In order to avoid them, set `max_features=int(1)`.

Mean Decrease Impurity (2/2)

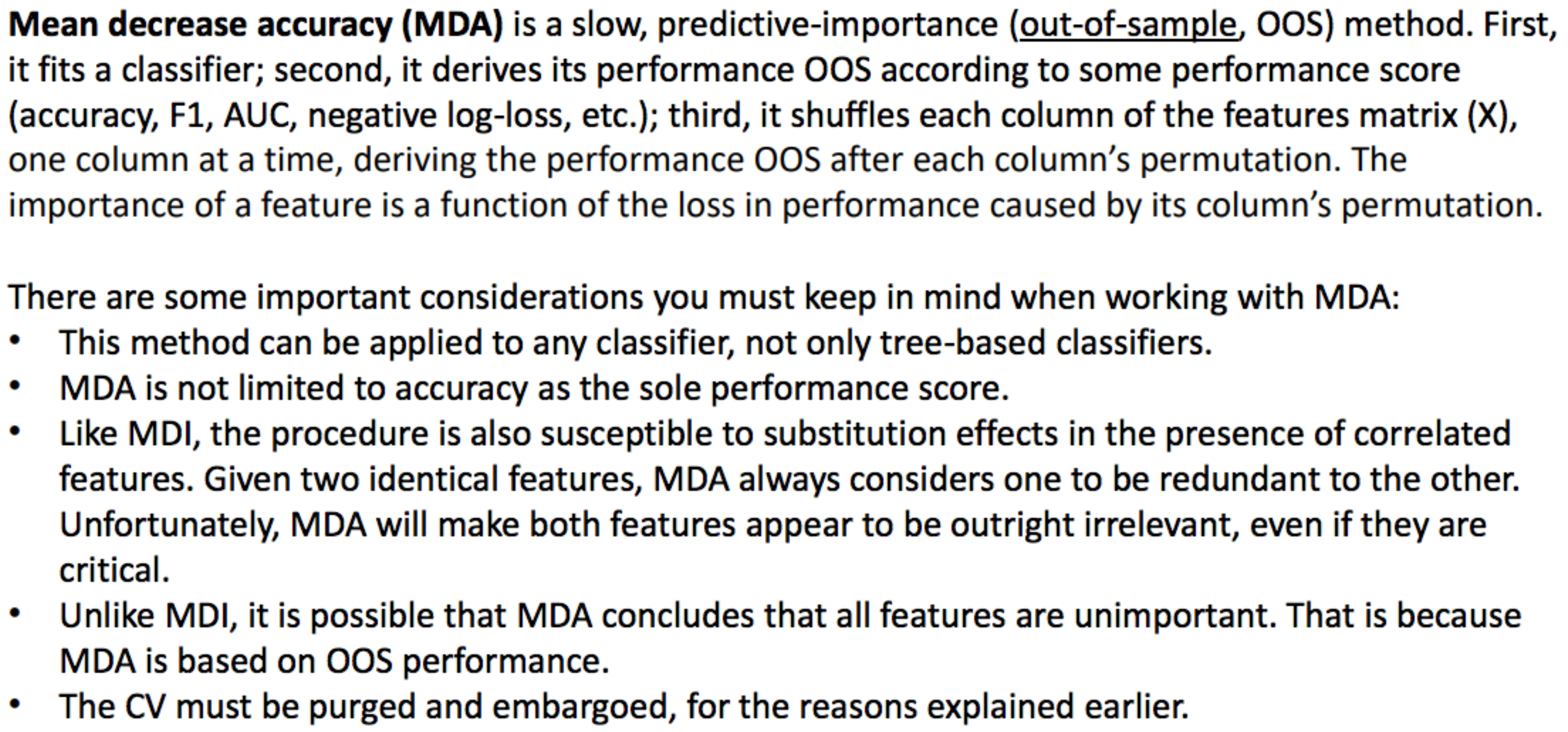


This plots shows the result of applying MDI to the same random classification problem discussed earlier.

The horizontal bars indicate the mean of MDI values across 1,000 trees in a random forest, and the lines indicate the standard deviation around that mean. The more trees we add to the forest, the smaller becomes the standard deviation around the mean.

MDI does a good job, in the sense that all of the non-noisy features (either informed or redundant) are ranked higher than the noise features. Still, a small number of non-noisy features appear to be much more important than their peers. This is the kind of substitution effects that we anticipated to find in the presence of redundant features.

Mean Decrease Accuracy (1/2)

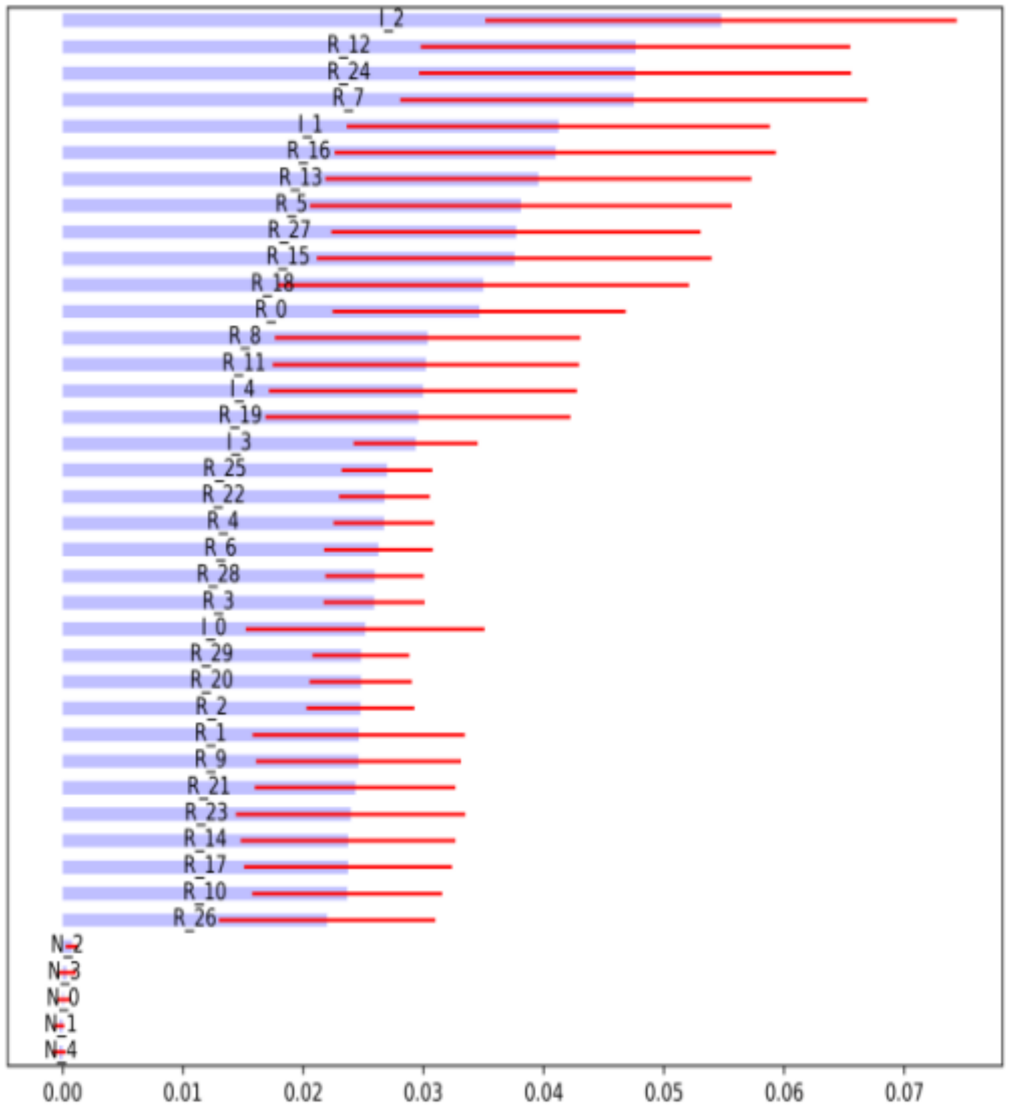


Mean decrease accuracy (MDA) is a slow, predictive-importance (out-of-sample, OOS) method. First, it fits a classifier; second, it derives its performance OOS according to some performance score (accuracy, F1, AUC, negative log-loss, etc.); third, it shuffles each column of the features matrix (X), one column at a time, deriving the performance OOS after each column's permutation. The importance of a feature is a function of the loss in performance caused by its column's permutation.

There are some important considerations you must keep in mind when working with MDA:

- This method can be applied to any classifier, not only tree-based classifiers.
- MDA is not limited to accuracy as the sole performance score.
- Like MDI, the procedure is also susceptible to substitution effects in the presence of correlated features. Given two identical features, MDA always considers one to be redundant to the other. Unfortunately, MDA will make both features appear to be outright irrelevant, even if they are critical.
- Unlike MDI, it is possible that MDA concludes that all features are unimportant. That is because MDA is based on OOS performance.
- The CV must be purged and embargoed, for the reasons explained earlier.

Mean Decrease Accuracy (2/2)



This plot shows the result of applying MDA to the same random classification problem we discussed earlier.

We can draw similar conclusions as we did in the MDI example.

First, MDA does a good job overall at separating noise features from the rest. Noise features are ranked last.

Second, noise features are also deemed unimportant in magnitude, with MDA values of essentially zero.

Third, although substitution effects contribute to higher variances in MDA importance, none is high enough to question the importance of the non-noisy features.

Single Feature Importance

Single feature importance (SFI) is a cross-section predictive-importance (out-of-sample) method. It computes the OOS performance score of each feature in isolation.

A few considerations:

- This method can be applied to any classifier, not only tree-based classifiers.
- SFI is not limited to accuracy as the sole performance score.
- Unlike MDI and MDA, no substitution effects take place, since only one feature
- is taken into consideration at a time.
- Like MDA, it can conclude that all features are unimportant, because performance
- is evaluated via OOS CV.

Orthogonalized Feature Importance

One way to partially address substitution effects is to **orthogonalize the features before applying MDI and MDA**. Consider a matrix $\{X_{t,n}\}$ of stationary features, with observations $t = 1, \dots, T$ and variables $n = 1, \dots, N$:

1. We compute the standardized features matrix Z, such that $Z_{t,n} = \sigma_n^{-1}(X_{t,n} - \mu_n)$, where μ_n is the mean of $\{X_{t,n}\}_{t=1,\dots,T}$ and σ_n is the standard deviation of $\{X_{t,n}\}_{t=1,\dots,T}$.
2. We compute the eigenvalues Λ and eigenvectors W such that $Z'ZW = W\Lambda$, where Λ is an NxN diagonal matrix with main entries sorted in descending order, and W is an NxN orthonormal matrix.
3. We derive the orthogonal features as $P = ZW$. We can verify the orthogonality of the features by noting that $P'P = W'Z'ZW = W'W\Lambda W'W = \Lambda$.

The diagonalization is done on Z rather than X, for two reasons: (1) centering the data ensures that the first principal component is correctly oriented in the main direction of the observations. It is equivalent to adding an intercept in a linear regression; (2) re-scaling the data makes PCA focus on explaining correlations rather than variances.

Clustered Feature Importance

- A better approach, which does not require a change of basis, is to cluster together similar features, and apply the feature importance analysis at the cluster level.
- By construction, clusters are mutually dissimilar, hence taming the substitution effects. Because the analysis is done on a partition of the features, without a change of basis, results are usually intuitive.

- Let us introduce one algorithm that implements this idea. The clustered feature importance (CFI) algorithm involves two steps:

  1. finding the number and constituents of the clusters of features. E.g., apply the ONC algorithm to the observed features.
  2. apply the feature importance analysis on groups of similar features rather than on individual features.

Single Feature Importance

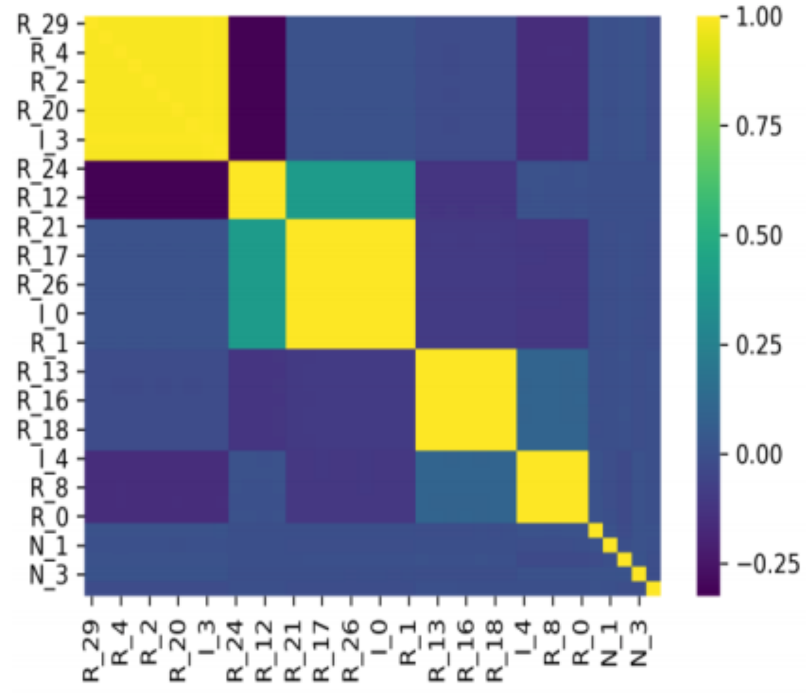
Single feature importance (SFI) is a cross-section predictive-importance (out-of-sample) method. It computes the OOS performance score of each feature in isolation.

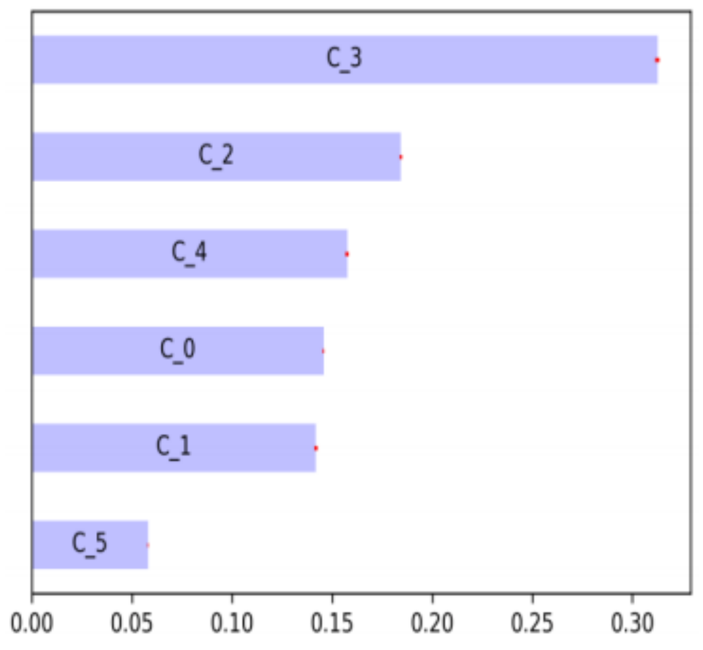
A few considerations:

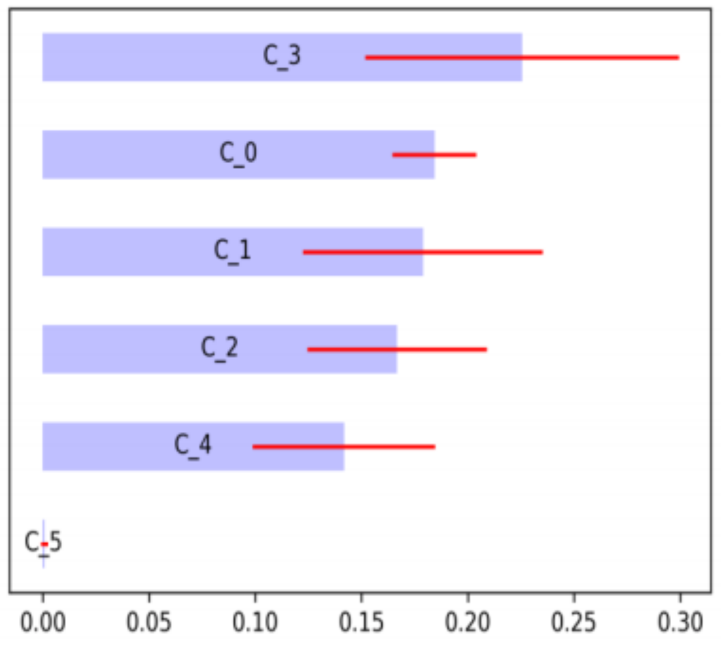
- This method can be applied to any classifier, not only tree-based classifiers.
- SFI is not limited to accuracy as the sole performance score.
- Unlike MDI and MDA, no substitution effects take place, since only one feature
- is taken into consideration at a time.
- Like MDA, it can conclude that all features are unimportant, because performance
- is evaluated via OOS CV.

CFI Experimental Results

- In this experiment we are going to test the clustered MDI and MDA procedures on the same dataset we used on the non-clustered versions of MDI and MDA. We apply CFI to control for substitution effects.
- The left plot shows that ONC correctly recognizes that there are 6 relevant clusters (one cluster for each informative feature, plus one cluster of noise features), and it assigns the redundant features to the cluster that contains the informative feature from which the redundant features were derived.
- MDI (center plot) and MDA (right plot) recognize that C_5 contains uninformative features.







Hyper-parameter Tuning with CV

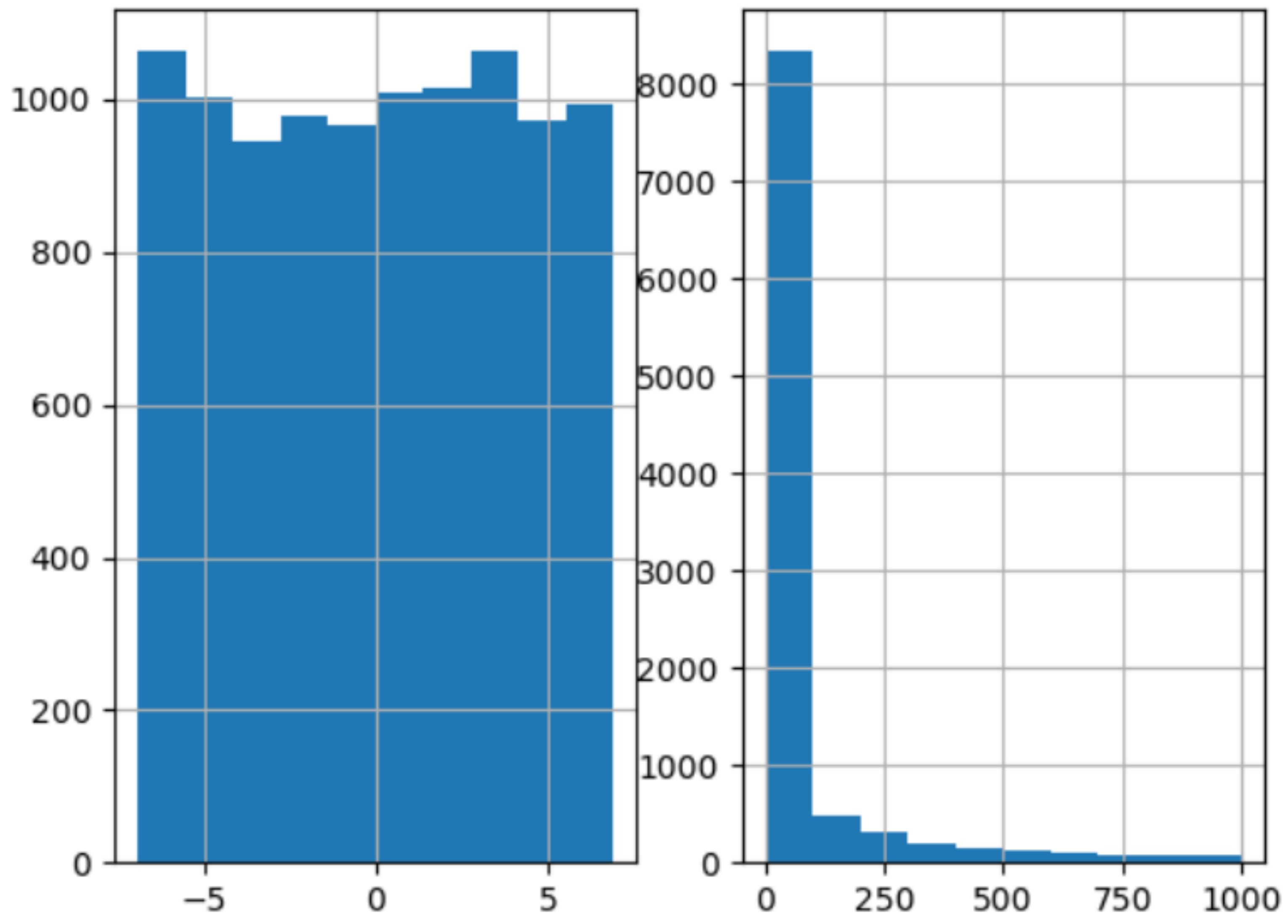
Grid Search CV

- Grid search cross-validation conducts an exhaustive search for the combination of parameters that maximizes the CV performance *on the validation set*, according to some user-defined score function.
- When we do not know much about the underlying structure of the data, this is a reasonable first approach.
- In the context of meta-labeling, you should avoid scoring by accuracy or negative log-loss:
    - Suppose a sample with a very large number of negative (i.e., label '0') cases.
    - A classifier that predicts all cases to be negative will achieve high 'accuracy' or 'neg_log_loss', even though it has not learned from the features how to discriminate between cases.
    - In fact, such a model achieves zero recall and undefined precision.
    - The 'f1' score corrects for that performance inflation by scoring the classifier in terms of precision and recall.
- For other (non-meta-labeling) applications, it is fine to use 'accuracy' or 'neg_log_loss', because we are equally interested in predicting all cases.

Randomized Search CV

- For ML algorithms with a large number of parameters, a grid search cross-validation (CV) becomes computationally intractable.
- In this case, an alternative with good statistical properties is to sample each parameter from a distribution. This has two benefits:
    - We can control for the number of combinations we will search for, regardless of the dimensionality of the problem (the equivalent to a computational budget).
    - Having parameters that are relatively irrelevant performance-wise will not substantially increase our search time, as would be the case with grid search CV.
- It is common for some ML algorithms to accept **non-negative hyper-parameters** only. That is the case of some parameters, such as C in the SVC classifier and gamma in the RBF kernel.
    - We could draw random numbers from a uniform distribution bounded between 0 and some large value, say 100. That would mean that 99% of the values would be expected to be over 1.
    - That is not necessarily the most effective way of exploring the feasibility region of parameters whose functions do not respond linearly. For example, an SVC can be as responsive to an increase in C from 0.01 to 1 as to an increase in C from 1 to 100, so sampling C from a *U* [0, 100] (uniform) distribution will be inefficient.

Log-Uniform Distribution (1/2)



In those instances, it seems more effective to draw values from a distribution *where the logarithm of those draws will be distributed uniformly*.

Log-Uniform Distribution (2/2)

- A random variable *x* follows a log-uniform distribution between $a > 0$ and $b > a$ if and only if $\log[x] \sim U[\log[a], \log[b]]$. This distribution has a CDF:

$$F[x] = \begin{cases} \dfrac{\log[x] - \log[a]}{\log[b] - \log[a]} & \text{for } a \leq x \leq b \\ 0 & \text{for } x < a \\ 1 & \text{for } x > b \end{cases}$$

- From this, we derive a PDF:

$$f[x] = \begin{cases} \dfrac{1}{x \log[b/a]} & \text{for } a \leq x \leq b \\ 0 & \text{for } x < a \\ 0 & \text{for } x > b \end{cases}$$

Note that the CDF is invariant to the base of the logarithm, since $\frac{\log\left[\frac{x}{a}\right]}{\log\left[\frac{b}{a}\right]} = \frac{\log_c\left[\frac{x}{a}\right]}{\log_c\left[\frac{b}{a}\right]}$ for any base *c*, thus the random variable is not a function of *c*.

Cross-Entropy Loss

- Investment strategies profit from predicting the right label with high confidence: Gains from good predictions with low confidence will not suffice to offset the losses from bad predictions with high confidence.
- For this reason, **accuracy does not provide a complete scoring of the classifier's performance**.
- Conversely, log loss (aka cross-entropy loss) computes the log-likelihood of the classifier given the true label, which takes predictions' probabilities into account.

$$L[Y,P] = -\log\left[\mathrm{Prob}[Y|P]\right] = -N^{-1}\sum_{n=0}^{N-1}\sum_{k=0}^{K-1} y_{n,k}\log\left[p_{n,k}\right]$$

where

- $p_{n,k}$ is the probability associated with prediction *n* of label *k*.
- Y is a 1-of-*K* binary indicator matrix, such that $y_{n,k} = 1$ when observation n was assigned label k out of *K* possible labels, and 0 otherwise.

Probability-Weighted Accuracy

- One disadvantage, however, is that log-loss scores are not easy to interpret and compare. A possible solution is to compute the probability-weighted accuracy (PWA) as

$$PWA = \sum_{n=0}^{N-1} y_n(p_n - K^{-1}) \Big/ \sum_{n=0}^{N-1} (p_n - K^{-1})$$

where $p_n = \max_k\{p_{n,k}\}$, and y_n is an indicator function, $y_n \in \{0,1\}$, where $y_n = 1$ when the prediction was correct and $y_n = 0$ otherwise.

- This is equivalent to standard accuracy when $p_n = 1$ for all *n*.

- PWA punishes a high confidence bad prediction more strongly than accuracy, but less strongly than cross-entropy.