Feature Selection for Machine Learning

**What is feature selection?**

Feature selection defines the process of identifying and selecting a subset of variables from the original data set to use as inputs in a machine-learning model.

A data set usually begins with a large number of features, but we can employ a variety of methods to determine which of these features are actually important in making predictions. Each of the different methods have varying advantages and disadvantages to consider.

**Importance of Feature Selection**

At a glance, it may seem that the more information one feeds to the model, the more accurate it will be. However, simpler models are more effective in a variety of ways, particularly when used in an organization.

Models with fewer features have:

* Better interpretability
* Reduced redundancy
* Shorter training times
* Simpler production code

If a model has ten predictive variables, for example, versus a hundred, people can understand the effect of the variables on the outcome much more clearly. This becomes particularly useful when the model is used in real life, say for example to prevent fraud. When an event or application is flagged as potentially fraudulent by a machine learning model, the fraud investigators will follow it up, and it helps them, if there are a few, clear indicators, the variables in the model, that point them in the right direction.

Oftentimes if too many variables exist in a model, many are redundant. Redundant variables will not add to the model, and large feature subsets may actually reduce the performance of some machine learning models, particularly those derived from tree-based algorithms.

Additionally, simpler models require shorter training times. Less variables used to build the models reduces computational cost. This speeds up model building and allows the models to score the incoming data much faster. If a business intends to use the model to make real time decisions, speed becomes particularly important.

Simpler models require simpler production code and that means less error handling and unit testing, which makes them easier to implement. They also reduce the risk of data dependent errors; if a business uses data collected from a third party API, the more variables in the model, the more exposure they have to any errors or, simply, changes made by the third party, or even worse, a temporary shut-down of the API.

Simpler models tend to generalize better and therefore show reduced overfitting. Too many variables often add noise to the model rather than predictive value. Eliminating noisy and irrelevant features makes the model generalize better to new, unseen data, which gives way for more accurate predictions.

With the following feature selection techniques, an organization can benefit from simpler models while still achieving effective and robust machine learning models.

**How Do We Select Features?**

A feature selection procedure or algorithm involves a combination of search techniques and evaluation measures; here, search techniques propose new feature subsets, and evaluation measures determine the how good the subset is.

In a perfect world, a feature selection method could iterate through all possible subsets of feature combinations and calculate which one results in the best performance. However, computational cost inhibits such a practice in reality. In addition, the optimal subset of features varies between machine learning models. A feature subset that optimizes one model’s performance will not necessarily optimize another’s.

To combat these challenges, there are many feature selection algorithms we can use to find a close-to-optimal subset of features in a manner that is computationally cost efficient. Generally, these methods have characteristics that allow them to group naturally into one of three categories: filter methods, wrapper methods, and embedded methods.

**Filter Methods** describe procedures that rely on characteristics in the data for their selection procedures. They look at each feature individually and assess how important they are in prediction.

**Wrapper Methods** examine all or almost all possibilities of feature combinations to identify the optimal feature set. Because of this, they are known as “greedy” algorithms.

**Embedded Methods** describe selection procedures that occur alongside fitting the machine-learning model. Combining these two processes allows them to consider the interaction between the model and the features.

**Filter Methods:**

A typical filter algorithm consists of two steps: it ranks features based on certain criteria and then chooses the highest-ranking features to train the models.

Filter methods are generally univariate, so they rank each feature independently of the rest. Because of this, the filter methods tend to ignore any interactions that occur between features. Thus, redundant variables will not necessarily be eliminated by filter methods.

However, some multivariate filter selection methods exist as well. These consider features in relations to others in the data set, making them naturally capable of handling redundant features. Their selection criteria scans for duplicated features and correlated features and provide simple but powerful methods to quickly remove redundant information.

The filter methods are basic, but essential. Though they do not usually find the best feature subset, they are model agnostic and are often an easy first step to reduce the feature space without significant computational cost. I will describe some of the popular filter methods here, as well as provide insight about when to use them.

**Constant, Quasi-Constant, and Duplicated**

The most basic and intuitive methods for feature selection consist of removing constant, quasi-constant, or duplicated features.

Constant features only show one value for all the observations in the data set. That is, they show absolutely no variability. Quasi-constant features are similar; if most observations share the same value, and then we would label that feature quasi-constant. In practice, quasi-constant features typically refer to those variables where more than 95 to 99 percent of the observations show the same value.

Duplicated features, as the name indicates, are those that are in essence, identical. That is, for every observation, they show the same value.

Although it sounds obvious and overly simple, many datasets contain many constant, quasi-constant, and duplicated features. In fact, duplicated features often arise when generating new features by one-hot encoding of categorical variables. Removing these features is an easy but effective way to reduce the dimension of the feature space, without losing any significant information.

**Correlation**

Correlation measures the linear association between two or more variables. The higher the correlation, the more linearly associated the variables are. The central hypothesis is that good feature sets contain features that are highly correlated with the target, yet uncorrelated with each other.

If two variables are correlated, we can predict one from the other. Therefore, if two features are correlated, the model only really needs one of them, as the second one does not add additional information.

Though correlated features shouldn’t diminish the accuracy of the model, there are certain detriments to having them. For linear models, multi-collinearity may reduce performance or mask the true contribution of each feature to the prediction. For decision trees and random forests, the model building methods will assign roughly the same importance for both correlated features, but this would be roughly half the importance it would assign if there were just one feature present, detracting from interpretability.

**Statistical Measures**

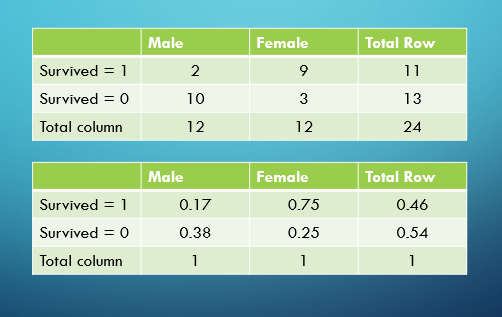
The filter methods discussed up until this point evaluate features individually or compare features to other features. The following methods, however, evaluate features in light of the target. These methods rank the features so we can assess their importance.



**Mutual information** measures the mutual dependence between two variables, in this case, the feature and the target. Mutual information is similar to correlation, but more general; it does not strictly represent linear association. It measures how much knowing one of these variables reduces uncertainty in the other.

**Fisher score** uses the Chi-Square distribution to measure the dependency between two variables and generally works best with categorical or discrete variables.

Fisher score essentially compares the actual frequencies of the values of a variable with the expected frequencies if it had no relation to the target. For example, we can think of Titanic survivors. If there was no relationship between gender and survival, about 50 percent of the survivors would be male and 50 percent female. However, it turns out about 75 percent of the survivors were female, suggesting that gender and survival of the Titanic have a significant relationship. Thus, the variable gender is very likely, a good predictor of the survival.



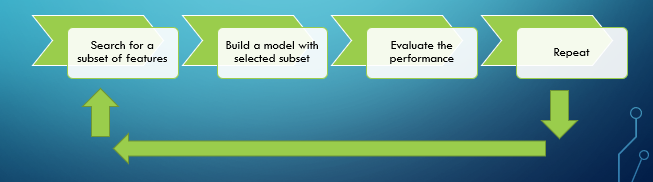
**ANOVA** is another method to measure dependencies between two variables, but unlike the Fisher score, is suited to continuous variables. It requires a binary target, and it essentially compares the distribution of the variable when the target is one versus the distribution when the target is zero. There are implementations that extend this functionality for use with continuous targets.

**ROC-AUC or RMSE** measure the performance of a model for either classification or regression methods, respectively. We can select features by building a machine learning model using only one feature and then evaluating the model’s performance with one of these metrics. After repeating this for every feature, we can rank the features with this metric, and then select the top-ranking ones.

**Wrapper Methods**

In comparison to filter methods, wrapper methods tend to be more computationally expensive, but select a better set of features. Wrapper methods use a specific algorithm in the selection process and choose the best subset of features tailored for that algorithm; however, this subset of features may not be optimal for all algorithms, meaning wrapper methods are not model agnostic.

A typical wrapper method will perform the following:



Wrapper methods start by searching through different subset of features, then creating a model with each. They evaluate these models to select the best one, and afterwards, they iterate to define a new subset based on the previous best subset.

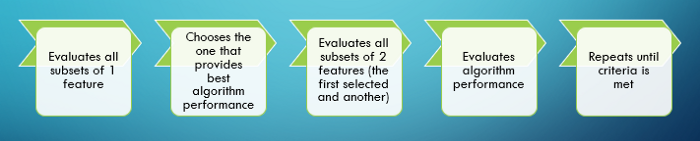
Deciding when to stop this search comes down to monitoring whether the performance does not increase or decrease beyond a certain threshold, depending on what method you are using. These thresholds are often arbitrary and defined by the user.

I will discuss these procedures in more details for specific wrapper methods, including step forward feature selection, step backward feature selection, and exhaustive search.

**Step Forward Feature Selection**

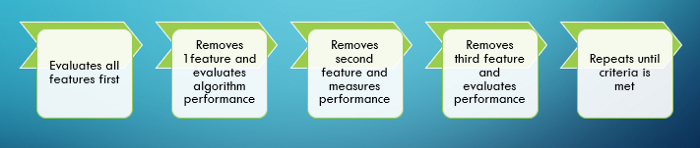
The step forward feature selection procedure begins by evaluating all feature subsets that consist of only one input variable. It selects the “best” feature, afterwards, adds all the other features to it individually, and selects a second feature that creates the new best performing model.

The process repeats repeatedly, adding one feature at a time, until it meets certain criteria. After adding additional features to the subset, if the machine learning model performance does not improve by more than a specific threshold, then we can stop the search and select this feature subset.



**Step Backward Feature Selection**

Step backward feature selection works in an opposite way. It begins by building a machine-learning model with all of the features, and then creates new subsets by removing every feature, one at a time, and training a new machine-learning model. It evaluates each subset to find the best one, and then continues the process of removing a new single feature that results in the best model.

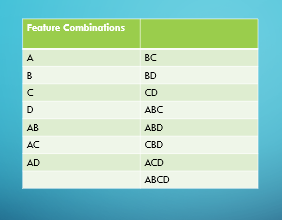


The process stops when an additional feature is removed and the performance does not decrease past a certain arbitrary threshold.

**Exhaustive Search**

An exhaustive search is the most “greedy” method. It tries all combinations of features, for any number of features from one until the maximum number of features available. This method is extremely computationally expensive, almost prohibitively, but would provide the best subset of features in theory.

For example, if there were four features to choose from (A, B, C, and D), an exhaustive search would build and evaluate models for fifteen feature subsets. It would go through all possible groupings: each feature on its own, every combination of paired features, every threesome combination, and the set of all features, as depicted below.



**Embedded Methods**

Finally, we have embedded methods. These encompass the benefits of both the wrapper and filter methods, by evaluating interactions of features but also maintaining reasonable computational cost.

The typical steps for embedded methods involve training a machine-learning algorithm using all the features, then deriving the importance of those features according to the algorithm used. Afterwards, it can remove unimportant features based on some criteria specific to the algorithm, of which I will cover some examples of shortly.

**Lasso**

Lasso belongs to a group of methods used to add penalty to different coefficients to reduce their freedom, thereby reducing its tendency to fit the noise. These methods are called regularization. Regularization improves the model’s generalization, and comes in three different forms: L1, L2, and L1/L2. These are also referred to as Lasso, Ridge, and Elastic net, respectively.

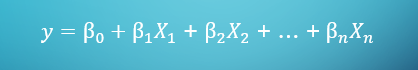
These methods apply penalties to coefficients of predictors in the model; this means the higher the penalty, the less the model will over fit and the better it will generalize to unseen data. Here, I will only go into the Lasso method, as this is the only regularization method that allows for feature selection.

During the Lasso fitting algorithm, the model tries to minimize the difference between the predicted and estimated value of the observation with the penalty. This regularization can shrink some coefficients of the linear regression to zero. This indicates that the predictor can essentially be multiplied by zero to estimate the target and consequently does not add to the overall prediction of the output.

In this way, Lasso regularization helps determine which features can be removed from the model.

**Regression Coefficients**

Linear models aim to predict a target based on given variables by assigning a coefficient to each. They follow the form of this equation:



Here, each variable X is multiplied by a coefficient, beta, to predict the final value of y. The betas here are directly proportional to how much the feature contributes to y. This holds true for any regularized or unregularized linear methods. Consequently, we are able to use these coefficients to select features.

There are a few caveats, however: this method assumes there is a linear relation between the predictors and the target and the scale of the features affects interpretability of the coefficients.

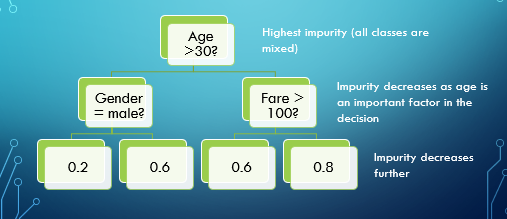
Some features may only contain values between zero and one, while others may range into the thousands or greater, which would affect the size of the coefficient. To truly compare features, we would need to put all the features into similar scales.

If all of these conditions hold and the features are in a similar scale, we can safely infer that the coefficients align with variable importance and therefore remove any that have negligible coefficients.

**Decision Tree Derived Importance**

Random forests and decision trees are generally very popular among the machine learning algorithms because they provide good predictive performance, low overfitting, and easy interpretability. Part of the interpretability encompasses how straightforward it is to derive the importance of each feature on a decision, which makes it one of the better-embedded methods of feature selection.

Random forests are in essence a collection of decision trees. Each decision tree is comprised of a series of “questions” based on different features in the data set. For each feature, the decision tree formulates a question of the type “is the value of the observation bigger than a for feature x?” If the answer is yes, the observation is allocated on one side of the note, if it is no, the data is taken to the other side of the node. The answer leads to the highest possible decrease of impurity, meaning it gives the best possible separation of the class.



Random forests consist of hundreds to thousands of decision trees built over a random extraction of the observations and features from the data set. The random extraction helps train de-correlated trees, and thus improve generalization.

Each node of the decision tree separates the data into two buckets. The importance of the feature is derived by how high the node in which the feature is used, is used is located in the tree, in how many different trees the feature is used, and by how much the impurity is decreased.

We can therefore select features by building a random forest, looking at the importance derived for each feature, and selecting the most important ones.

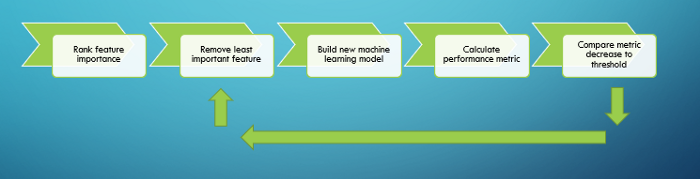
**Hybrid Methods**

As the name might suggest, hybrid methods combine pieces of wrapper methods and embedded methods. They build several algorithms at each round of selection like wrapper methods, but they do not examine every possible feature combination. In addition, they select features based on a specific machine-learning algorithm and evaluate the model performance.

Some common hybrid methods are recursive feature elimination and recursive feature addition.

**Recursive Feature Elimination**

This hybrid method consists of the following steps:



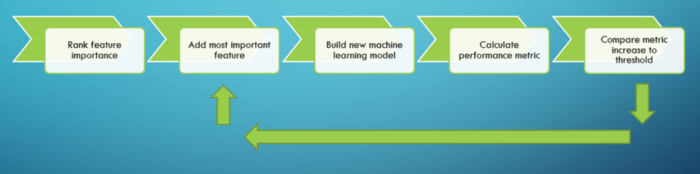
It starts with ranking the feature with importance derived from an embedded method, such as those discussed earlier. Next, we remove the least important feature and build a new machine-learning algorithm. We then calculate a performance metric, such as ROC-AUC, MSE, or RMSE.

If the metric decreases by more than an arbitrarily set threshold, then the feature should be kept. Otherwise, we can remove that feature and repeat this process until a feature removal causes the performance metric to decrease past this threshold.

The difference between this method and the step backward feature selection method is that here, we don’t remove all features first in order to determine which to remove, we simply select the least important feature based on what the machine learning model derives as important. It removes only one feature, rather than all the features at each step.

**Recursive Feature Addition**

The recursive feature addition hybrid method works similarly, but adding features one at a time rather than removing one.



It begins by ranking the features with one of the embedded methods, and then builds a model with only the one most important features. Again, we calculate the performance metric of our choice. Then we add the second most important feature and train a model, and if the metric increases over an arbitrarily set threshold, then we should keep the feature. Otherwise, it can be removed.

This process will repeat until all features have been evaluated. Similar to recursive elimination, it differs from step forward feature selection because it doesn’t add all features to determine what to keep, but instead, just adds the most important one.

This method and the recursive feature elimination method usually work faster than wrapper methods and better than embedded methods. They account for correlations depending on how stringent the threshold is set, but on the downside, this threshold is, again, set arbitrarily; the smaller the threshold, the more features will be selected.

**Random Shuffling of the Values**

Another popular method of feature selection consists in randomly shuffling the values of a specific variable and observing how the permutation affects the performance metric of the algorithm. If the variable is important, the random permutation will decrease the performance metrics dramatically. On the other hand, permuting unimportant variables should have little to no effect on the performance of the model. Thus, this information can be used to determine which variables are predictive enough to keep in the feature subset.

Feature selection in machine learning models allows for accurate models that are simpler and less computationally expensive to implement. Some methods I have outlined in this blog post should serve as a guide for selecting features to optimize a model in a variety of different situations. Though there is not one perfect method, each method provides advantages and disadvantages, and between all the methods available, one should be able to find one to optimize the machine-learning model