Wrapper Methods: Definition

Wrapper methods work by evaluating a subset of features using a machine learning algorithm that employs a search strategy to look through the space of possible feature subsets, evaluating each subset based on the quality of the performance of a given algorithm.

These methods are called greedy algorithms because they aim to find the best possible combination of features that result in the best performant model— which will be computationally expensive, and often impractical in the case of exhaustive search.

Practically any combination of a search strategy and a machine learning algorithm can be used as a wrapper.

Wrapper Methods: Advantages

Given the issues we encountered using filter methods, wrapper methods present two main advantages that deal with those issues:

* They detect the interaction between variables
* They find the optimal feature subset for the desired machine learning algorithm

The wrapper methods usually result in better predictive accuracy than filter methods.

Wrapper Methods: Process

Wrapper methods work in the following way, generally speaking:

* **Search for a subset of features:** Using a search method (described next), we select a subset of features from the available ones.
* **Build a machine learning model:**In this step, a chosen ML algorithm is trained on the previously-selected subset of features.
* **Evaluate model performance:** And finally, we evaluate the newly-trained ML model with a chosen metric.
* **Repeat:** The whole process starts again with a new subset of features, a new ML model trained, and so on.

We stop until the desired condition is met, and then we choose the best subset with the best result in the evaluation phase.

Machine learning models are moving closer and closer to edge devices. Fritz AI is here to help with this transition. [Explore our suite of developer tools that makes it easy to teach devices to see, hear, sense, and think.](https://www.fritz.ai/product/platform.html?utm_campaign=buildmodels4&utm_source=heartbeat)

Stopping Criteria

At some point in time, we need to stop searching for a subset of features. To do this, we have to put in place some pre-set criteria (often somewhat arbitrary), around when this searching should stop. These criteria need to be defined by the machine learning engineer. Here are a couple of examples of these criteria:

* Model performance decreases.
* Model performance increases.
* A predefined number of features is reached.

The pre-set criteria, for example, can be metrics like ROC-AUC for classification or RMSE for regression.

Search methods

* **Forward Feature Selection:** This method starts with no feature and adds one at a time.
* **Backward Feature Elimination:**This method starts with all features present and removes one feature at the time.
* **Exhaustive Feature Selection:** This method tries all possible feature combinations.
* **Bidirectional Search:**And this last one does both forward and backward feature selection simultaneously in order to get one unique solution.

Next, we’ll explore each method in-depth and alongside simple Python implementations.

Library to install

Let's start by first installing a library that will help us use these wrapper methods. This library, called **[mlxtend](http://rasbt.github.io/mlxtend/),**containsuseful tools for a number of day-to-day data science tasks.

Here’s the command for both pip and conda:

# command for pip.  
pip **install** mlxtend# command for conda.  
conda **install** mlxtend --channel conda-forge

Machine learning is rapidly moving closer to where data is collected — edge devices. [Subscribe to the Fritz AI Newsletter to learn more about this transition and how it can help scale your business.](https://www.fritz.ai/newsletter?utm_campaign=fritzai-newsletter-scale6&utm_source=heartbeat)

**Forward Feature Selection**

Also known as step forward feature selection (or sequential forward feature selection — SFS), this is an iterative method in which we start by evaluating all features individually, and then select the one that results in the best performance.

In the second step, it tests all possible combinations of the selected feature with the remaining features and retains the pair that produces the best algorithmic performance.

And the loop continues by adding one feature at a time in each iteration until the pre-set criterion is reached.

Here is a code sample:

This snippet basically says that we’re using an object called [SequentialFeatureSelector](http://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/)**,** with the following parameters:

* k\_features: The maximum feature to be reached when starting from 0— this is the**pre-set criteria**.
* forward: Use this object for step forward or step backward feature selection.
* floating: Use a variant of the step forward selection called step floating forward selection—we will talk about this more later.
* scoring: The scoring function by which we evaluate model performance. Be sure to set the right one for regression and classification cases, respectively.
* cv: The number of folds of K-fold cross-validation—no cross-validation if cv is 0 or False.

**Backward Feature Elimination**

This is also known as step backward feature selection (and sequential backward feature selection — SBS). As opposed to the first method, we start with all the features in the dataset, and then we evaluate the performance of the algorithm.

After that, at each iteration, backward feature elimination removes one feature at a time, which produces the best performing algorithm using an evaluation metric. This feature can be also described as the least significant feature among the remaining available ones.

And it continues, removing feature after feature until a certain criterion is satisfied.

Here is a code sample:

This snippet is just like the previous one, just with the following changes:

* k\_features: The minimum feature to be reached when starting from N (the number of total features)—this is the**pre-set criteria**.
* forward: Use this object for step forward or step backward feature selection.
* floating: Use a variant of the step backward selection called step floating backward selection.

**Exhaustive Feature Selection**

Finally, this method searches across all possible feature combinations. Its aim is to find the best performing feature subset—we can say it’s a brute-force evaluation of feature subsets.

It creates all the subsets of features from **1** to **N**, with **N** being the total number of features, and for each subset, it builds a machine learning algorithm and selects the subset with the best performance.

The parameters that you can play with here are the 1 and N, which can be described as the **minimum number of features** and the **maximum number of features.** That way, we can reduce this method’s computation time if we choose reasonable numbers for these parameters.

Here is a code sample for exhaustive search:

In this code sample, we’re using an object called [ExhaustiveFeatureSelection](http://rasbt.github.io/mlxtend/user_guide/feature_selection/ExhaustiveFeatureSelector/)**,** which has approximately the same parameters as the previous object [SequentialFeatureSelector](http://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/), except k\_features, floating, and the following:

* min\_features: As its name indicates, this is the lower bound of the number of features to search from.
* max\_features: The upper bound of the number of features to search from.

So the object will search for the combination of features of 4,5,6,7,8,9,10 and select the best combination that gives the higher roc\_auc score.

Limitations of Step Forward/Backward Selection

Besides the drawbacks of computation costs, there are some other points to pay attention to when working with SFS & SBS:

* Since we know that SFS adds features at each iteration, a problem can occur when we add up a feature that was useful in the beginning, but after adding more ones, is now non-useful. At this point, there’s no way to remove this kind of feature.
* The same thing happens to SBS but in the reverse direction—this is because of the inability of SBS to see the usefulness of a feature after being removed from the feature set.

For those reasons, and for more generalization for SBS and SFS, there are two methods that can solve such an issue: LRS and sequential floating.

**LRS or Plus-L, minus-R**

LRS uses two parameters **L** and **R** (both integers) to repeatedly add and remove features from the solution subset. Here’s an explanation of how it works:

If L>R, LRS starts from the empty set of features and:

* Repeatedly adds **L** features
* Repeatedly removes **R** features

If L<R, LRS starts from the full set of features and:

* Repeatedly removes **R** features
* Repeatedly adds **L** features

LRS attempts to compensate for the weaknesses of SFS and SBS with some backtracking capabilities, but this method requires you to carefully set L and R parameters since there’s a lack of theory behind choosing the optimal values.

**Sequential Floating**

For both backward and forward selection, sequential floating is an extension to LRS. Rather than choosing values for L and R to add and remove features, we determine this from the data directly.

During the search, the size of the subset will be floating up and down since we are adding and removing features.

How it works

The way this method works is quite simple to understand. Let’s explore it in the context of both methods:

* **Step floating forward selection:** After each forward step, **SFFS** performs backward steps as long as the objective function increases.
* **Step floating backward selection:** After each backward step, **SFBS** performs forward steps as long as the objective function increases.

Here’s a detailed explanation of how **SFFS**works in practice:

1. Start from an empty set.
2. Select the best feature as we usually do in SFS and add it.
3. Then select the worst feature from this subset.
4. Evaluate and check whether the objective function improves or not by deleting this feature. If it’s improving, we delete this feature; otherwise, we keep it.
5. Repeat from step 2 until the stop criterion is reached.

The same steps should be followed with SFBS, except that instead of starting with an empty set, we start with a full one, then proceed with normal SBS. Then, instead of deleting features in step 4, we add one that can improve the objective function.

Implementation

To use the floating method in code, mlxtendprovides a simple parameter to set in the previous object of SFS and SBS. This parameter is called floating and can be set to true or false depending on if you want to use this variant as a selection method.

Other Search Methods

**Bidirectional Search (BDS):**BDS applies SFS and SBS concurrently—SFS is performed from the empty set of features and SBS is performed from the full set of features.

But this can lead to an issue of converging to a different solution. To avoid this and to guarantee SFS and SBS converge to the same solution, we make the following constraints:

* Features already selected by SFS are not removed by SBS.
* Features already removed by SBS are not added by SFS.