Embedded Methods: Definition

Embedded methods complete the feature selection process within the construction of the machine learning algorithm itself. In other words, they perform feature selection during the model training, which is why we call them embedded methods.

A learning algorithm takes advantage of its own variable selection process and performs feature selection and classification/regression at the same time.

Embedded Methods: Advantages

The embedded method solves both issues we encountered with the filter and wrapper methods by combining their advantages. Here’s how:

* They take into consideration the interaction of features like wrapper methods do.
* They are faster like filter methods.
* They are more accurate than filter methods.
* They find the feature subset for the algorithm being trained.
* They are much less prone to overfitting.

Embedded Methods: Process

Any and all embedded methods work as follows:

* First, these methods train a machine learning model.
* They then derive feature importance from this model, which is a measure of how much is feature important when making a prediction.
* Finally, they remove non-important features using the derived feature importance.

The Methods

In this article, we’ll explore a few specific methods that use embedded feature selection: regularization and tree-based methods.

Using Regularization

[Regularization](https://heartbeat.fritz.ai/deep-learning-best-practices-regularization-techniques-for-better-performance-of-neural-network-94f978a4e518) in machine learning adds a penalty to the different parameters of a model to reduce its freedom. This penalty is applied to the coefficient that multiplies each of the features in the linear model, and is done to avoid overfitting, make the model robust to noise, and to improve its generalization.

There are three main types of regularization for linear models:

* **lasso regression**or L1 regularization
* **ridge regression** or L2 regularization
* **elastic nets** or L1/L2 regularization

Let’s look at each in a bit more detail:

* **L1 regularization** has shrinks some of the coefficients to zero, therefore indicating that a certain predictor or certain features will be multiplied by zero to estimate the target. Thus, it won’t be added to the final prediction of the target—this means that these features can be **removed** because they aren’t contributing to the final prediction.
* **L2 regularization**, on the other hand, doesn’t set the coefficient to zero, but only approaching zero—that’s why we use only L1 in feature selection.
* **L1/L2 regularization**is a combination of the L1 and L2. It incorporates their penalties, and therefore we can end up with features with zero as a coefficient—similar to L1.

Here is a code snippet to work with:

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)

Tree-based Feature Importance

Tree-based algorithms and models (i.e. random forest) are well-established algorithms that not only offer good predictive performance but can also provide us with what we call **feature importance** as a way to select features.

Feature importance

Feature importance tells us which variables are more important in making accurate predictions on the target variable/class. In other words, it identifies which features are the most used by the machine learning algorithm in order to predict the target.

Random forests provide us with feature importance using straightforward methods — **mean decrease impurity**and **mean decrease accuracy**.

How it works

A random forest is no more than a group of decision trees. Each of them is established over a random extraction of samples and features from the dataset, so an individual tree isn't able to see all the features or access all the observations.

Furthermore, every node in a decision tree is a condition on one feature—these nodes are designed to split the dataset into two different sets. Similar observation values will be in the same set, and different ones will be in the other.

Thus, the importance of each feature is derived by how “pure” each of the sets is.

The measure based on which optimal condition is chosen is known as an **impurity**. For classification, it’s typically either the Gini impurity or information gain/entropy; and for regression trees, it’s the variance.

Thus, when training a tree, feature importance is calculated as the decrease in node impurity weighted in a tree. The **higher** the value, the more **important** the feature.

Here’s a code sample that abstracts what we said in just a couple of lines:

You can use any other tree-based algorithm the same way we did here. One of the best tree model types is gradient boosting algorithms (like [XGBoost](https://heartbeat.fritz.ai/boosting-your-machine-learning-models-using-xgboost-d2cabb3e948f), CatBoost, and many more) since they provide accurate feature importance.