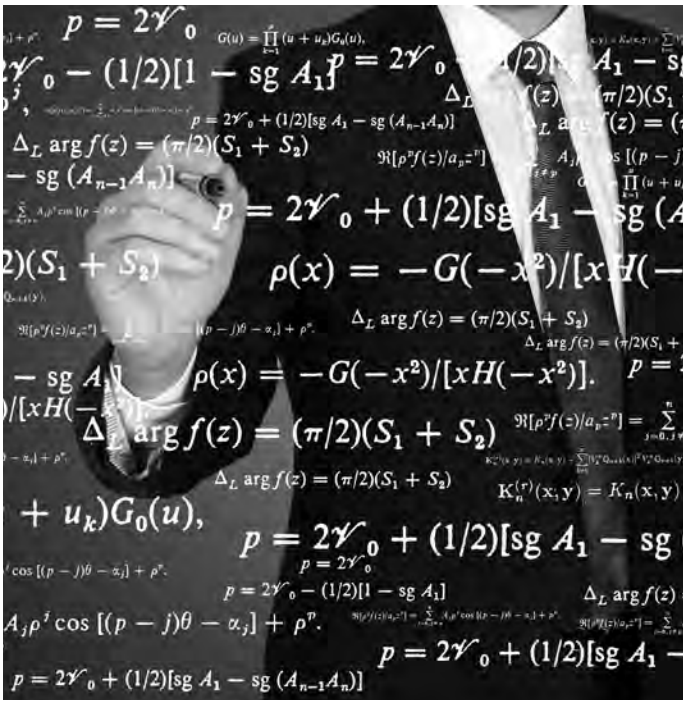
Article from

Predictive Analytics and Futurism

April 2018
Issue 17

Feature Importance in Supervised Training

By Jeff Heaton



Supervised learning is the class of machine learning where a model is trained to produce a specific result for a given input. These inputs and expected outputs form the training data for a model. Because the expected outputs are known, this type of training is referred to as supervised learning. If there are no expected outcomes, then the technique is referred to as unsupervised learning. The process of using these data is called training or fitting. Whether to use supervised or unsupervised learning depends upon the project goal. If the desire is to create a model that can be trained to produce some sort of output from input data, then you are using supervised training. The focus of this article is determining the importance of columns of your input data for supervised training.

In the domain of supervised learning, predictive models accept a feature vector and return a prediction. For example, a model might be asked to accept inputs that specify the face amount, annual premium, term, age of applicant, and other values to predict the likelihood of the policy being lapsed. These inputs are typically referred to as the feature vector or the x-values. The output from the model is typically referred to as the score, prediction or y-hat value. Some of the input features are more important to making an accurate prediction than others. For example, term length might be more important to predicting lapse than the face amount. There are a wide variety of techniques that can be used to measure the importance of the input features.

MODEL-SPECIFIC FEATURE RANKING

Depending on the type of model to be evaluated, there are a number of different ways to evaluate feature importance. These model-specific, feature-ranking techniques will change depending on what model you are using. For example, if you are dealing with a generalized linear model (GLM), the coefficients can provide an importance measure. Similarly, neural network feature importance can be gauged by examining the outbound weights from each of the input neurons.[1] Additionally, the importance of features in tree-based models, such as gradient boosting machines (GBMs), random forests, and classification and regression trees (CARTs) can be determined by evaluating the number and weighting of splits that the given feature was involved in.

Of course, these techniques are only valid for GLMs, neural networks and tree-based models. If you are making use of other model types, such as support vector machines (SVMs), k-nearest neighbors or any other, you will need to use a technique that is specifically designed for that model type. Furthermore, your importance will remain the same over time.

The importance of the model features is generated from the model parameters that were defined when the model was fit. It is not possible to see how important these features are with newer data sets that your model might need to score. Fitting a model and deploying it to production are only the first battles that a data scientist must face. It is important to ensure that your model remains relevant with new data sets and external conditions that might affect the validity of your model. Evaluating the importance of features for your trained model on new data sets can be an important piece of information in ensuring the continued robustness of your deployed model. Most model-specific, feature-ranking algorithms only analyze the model, and not the importance of features in entirely new data sets.

MODEL-AGNOSTIC FEATURE RANKING

Model-agnostic, feature-ranking algorithms consider the intrinsic characteristics of the data in evaluating the fitness of the feature subset. Model-agnostic, feature-ranking techniques do not require a learning algorithm and require fewer computing

resources. Rather, the model-agnostic algorithm makes use of an already trained model and a data set.

Correlation-coefficient feature importance is a very simple model-agnostic, univariate algorithm that calculates the absolute value of the correlation coefficient between each of a model's expected outputs. This value can be used to estimate the importance of each input feature to the model. The higher the correlation coefficient between an input (x) and the target (y), the greater a feature's importance. To calculate this coefficient, the first step is to calculate the covariance (C_{ij}) between the two features i and j. Usually, feature i will be the input feature currently being evaluated and j will be the target value. This is performed by the following equation:

$$C_{ij} = \sum_{i=i}^{n} \frac{(x_i - \bar{X})(y_i - \bar{Y})}{n-1}$$

The value n represents the number of rows in the training data. The value x represents each vector of predictors and y represents the expected value. The Pearson product-moment correlation coefficient is given by the following equation (which makes use of the previous equation):

$$R_{ij} = \frac{C_{ij}}{\sqrt{C_{ii} \cdot C_{jj}}}$$

The resulting value (R) gives the correlation between any of the inputs (i) and the target (j). The absolute value of R indicates how strongly correlated the input is to the target. Higher values are more strongly correlated. We provide a Python implementation of the correlation-coefficient, feature-importance-ranking algorithm that can be used with any Scikit-Learn model.[2]

The input perturbation algorithm[3] is a more complex agnostic, feature-importance algorithm that calculates the loss of a model when each of the input features to the neural network is perturbed by the algorithm. The idea is that when an important input is perturbed the neural network should have a considerable increase in error, that corresponds to the importance of that input. Because the inputs are being perturbed, rather than removed entirely, it is not necessary to train a new neural network for each evaluated feature. Rather, the feature is perturbed in the provided data set. The feature is perturbed in such a way that it provides little or no value to the neural network, yet the neural network retains an input neuron for that feature. No change is made to the neural network as each input is evaluated.

To effectively use feature-perturbation ranking it is necessary to evaluate the loss (E) of a model. If the model is regression, the following equation evaluates the loss between the expected output (y) and the model output ($\hat{y}$) over n data items:

$$E = \frac{\sum_{i=1}^{n} (\hat{y}_i - y_i)^2}{n}$$

If there are multiple outputs, they are simply considered as additional y and $\hat{y}$ values. If the neural network is classification, then a multi-logloss evaluate is performed:

$$E = -\frac{1}{n} \sum_{i=1}^{n} (y_i \log(\hat{y}_i) \cdot (1 - y_i) \log(1 - \hat{y}_i))$$

To successfully perturb a feature for the input-perturbation, feature-importance algorithm two objectives must be met. First, the input feature must be perturbed to the point that it now provides little or no predictive power to the neural network. Secondly, the input feature must be perturbed in such a way that it does not have adverse effects on the neural network beyond the feature being perturbed. Both objectives are accomplished by shuffling, or perturbing, the column that is to be evaluated. By shuffling the column, the wrong input values will be presented for each of the expected targets. Secondly, the shuffle ensures that most statistical measures of the column remain the same, as the column will maintain the same distribution.

To effectively use feature-perturbation ranking it is necessary to evaluate the loss *(E)* of a model.

Feature importance is usually reported as a table that shows the name of each feature, its relative importance, and the error that the model reported when that feature was perturbed. For example, Table 1 might represent the importance of four features:

Table 1
Sample Feature Importance Ranking

Feature Name	Importance	Loss
D	1	5
B	0.6	3
A	0.4	2
C	0.1	0.5

The higher the loss, the more important a feature is. The perturbation effectively removes the feature from the prediction. Removing an important feature will result in a higher loss than

removing a less important feature. Each feature has an importance that is reported as the value of that feature's loss divided by the highest loss. Because of this, the most important feature will always have an importance of 1. The importance values will not sum to 1.0. Rather, the importance values show the relative importance of each feature to the most important feature. We provide a Python implementation of the perturbation-ranking algorithm that can be used with any Scikit-Learn model.[4]

MULTIVARIATE FEATURE RANKING

It is also possible to use the perturbation feature-ranking algorithm to evaluate multivariate features. It is possible that two features are more important together than they are separately. To evaluate this, a pair-wise feature importance could be generated for each of the possible pairs of features, similar to how a covariance matrix is often calculated to determine which feature pairs are strongly correlated to each other.

The generation of a pair-wise multivariate feature importance report is produced similarly to the univariate-perturbation, feature-ranking algorithm presented in the previous section. The primary difference is that two columns will be perturbed at a time, rather than a single column. To perform this, it will be necessary to loop over every combination of features taken two at a time. For example, 10 features result in 45 evaluations. This is because 10 items, taken two at a time, yield 45 combinations.

Visually, this can be thought of as a pair-wise matrix. The diagonal is discarded, because that would consider each feature with itself. Likewise, the upper or lower triangle of the matrix can be discarded because the pair-wise importance of feature-1 and feature-2 is the same as the pair-wise importance of feature-2 and feature-1. Considering triplets, quadruplets and higher multiples would considerably increase the amount of processing that would be necessary.

SUMMARY

Feature-importance ranking is a very important consideration for data science. It can be used to optimize your data set and remove unimportant features to improve the performance of your model. This decreases the computation time needed for your model and often increases the accuracy. Feature engineering also benefits greatly from feature importance evaluation. As additional features are engineered, they can be evaluated to see their relative importance to the model. When using feature importance in conjunction with feature engineering, it is important to remember that the perturbation-ranking algorithm will typically share the importance between two closely correlated features. Because engineered features are mathematical combinations and transformations of the original feature set, the engineered features are usually strongly correlated to the original feature set. Therefore, it is important to keep in mind that the engineered features are usually sharing importance with the original features from which they were constructed. ■

Jeff Heaton, Ph.D., is lead data scientist, Reinsurance Group of America, in Chesterfield, Mo. He can be reached at *JHeaton@rgare.com*.

ENDNOTES

1 Goh, Anthony T. C. 1995. Back-Propagation Neural Networks for Modeling Complex Systems. *Artificial Intelligence in Engineering* 9, no. 3:143–151.

2 Heaton, Jeff, Steven McElwee, and James Cannady. Early Stabilizing Feature Importance for TensorFlow Deep Neural Networks. May 2017. In *International Joint Conference on Neural Networks (IJCNN 2017)*. IEEE.

3 Olden, Julian D., Michael K. Joy, and Russell G. Death. 2004. An Accurate Comparison of Methods for Quantifying Variable Importance in Artificial Neural Networks Using Simulated Data. *Ecological Modelling* 178, no. 3-4:389–397.

4 *Supra*, note 2.