**Best Practice to Calculate and Interpret Model Feature Importance**

**With an example of Random Forest model**

Source: [Unsplash](https://unsplash.com/photos/w7ZyuGYNpRQ) (Credit to Kevin Ku)

In machine learning, most of the time you want a model that is not only accurate but also interpretable. One example is customer churn prediction — in addition to knowing who will churn, it’s equally important to understand which variables are critical in predicting churn to help improve our service and product.

Popular machine learning packages such as Scikit-learn offer default calculations of feature importances for model interpretation. However, oftentimes we can’t trust those default calculations. In this article, we are going to use the famous [Titanic data from Kaggle](https://www.kaggle.com/c/titanic/data) and a Random Forest model to illustrate:

* Why you need a **robust** model and **permutation importance** **scores** to properly calculate feature importances.
* Why you need to understand the **features’ correlation** to properly interpret the feature importances.

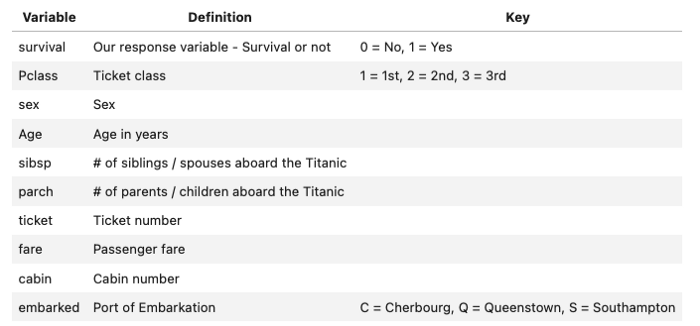
The practice described in this article can also be generalized to other models.

**Best Practice to Calculate Feature Importances**

**The trouble with Default Feature Importance**

We are going to use an example to show the **problem with the default impurity-based feature importances** provided in Scikit-learn for Random Forest. The default feature importance is calculated based on the mean decrease in impurity (or Gini importance), which measures how effective each feature is at reducing uncertainty. See [this great article](https://towardsdatascience.com/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e3) for a more detailed explanation of the math behind the feature importance calculation.

Let’s download the famous Titanic dataset from [Kaggle](https://www.kaggle.com/c/titanic/data). The data set has passenger information for 1309 passengers on Titanic and whether they survived. Here is a brief description of the columns included.



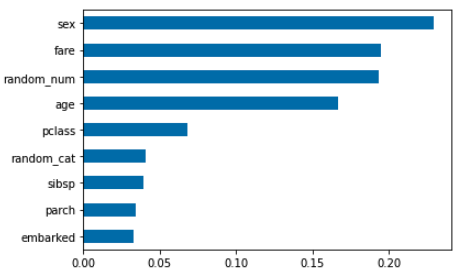
First, we load the data and separate it into a predictor set and a response set. In the predictor set, we add two random variables random\_cat and random\_num. Since they are randomly generated, both of the variables should have a very low feature importance score.

Second, we do some simple cleaning and transformation of the data. This is not the focus of this article.

Third, we build a simple random forest model.

RF train accuracy: 1.000.RF test accuracy: 0.814

The model is slightly overfitting on the training data but still has decent performance on the testing set. Let’s use this model for now to illustrate some pitfalls of the default feature importance calculation. Let’s take a look at the default feature importances.



From the default feature importances, we notice that:

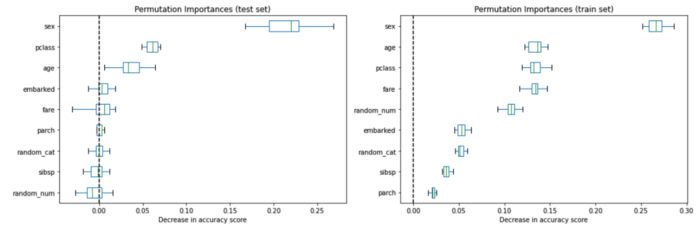
* The random\_num has a higher importance score in comparison with random\_cat, which confirms that **impurity-based importances are biased towards high-cardinality and numerical features**.
* Non-predictive random\_num variable is ranked as one of the most important features, which doesn’t make sense. This reflects the **default feature importances are not accurate when you have an overfitting model**. When a model overfits, it picks up too much noise from the training set to make generalized predictions on the test set. When this happens, the feature importances are not reliable as they are calculated based on the training set. More generally, **it only makes sense to look at the feature importances when you have a model that can decently predict.**

**Permutation Importance to Rescue**

Then how do we appropriately calculate the feature importances? One way is to use permutation importance scores. It’s computed with the following steps:

1. Train a baseline model and record the score (we use accuracy in this example) on the validation set.
2. Re-shuffle values for one feature, use the model to predict again, and calculate scores on the validation set. The feature importance for the feature is the difference between the baseline in 1 and the permutation score in 2.
3. Repeat the process for all features.

Here we leverage the permutation\_importance function added to the Scikit-learn package in 2019. When calling the function, we set the n\_repeats = 20 which means for each variable, we randomly shuffle 20 times and calculate the decrease in accuracy to create the box plot.



We see that sex and pclass show as the most important features and random\_cat and random\_num no longer have high importance scores based on the permutation importances on the test set. The box plot shows the distribution of the decrease in accuracy score with N repeat permutation (N = 20 in our case).

Let’s also compute the permutation importances on the training set. This shows that random\_num and random\_cat get a significantly higher importance ranking than when computed on the test set and the rankings of features look very different from the testing set. As noted before, it’s due to the overfitting of the model.

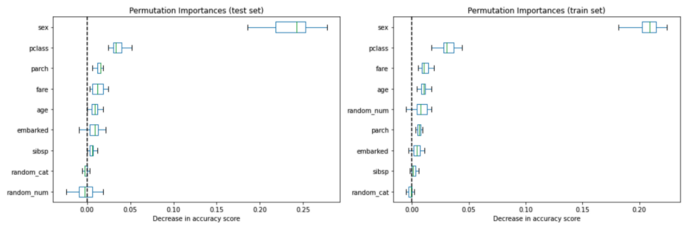
You may wonder why Scikit-learn still includes the default feature importance given it’s not accurate. Breiman and Cutler, the inventors of RFs, indicated that this method of “adding up the Gini decreases for each variable over all trees in the forest gives a fast variable importance that is often very consistent with the permutation importance measure.” So the default is meant to be a proxy for permutation importances. However, as Strobl et al pointed out in [Bias in random forest variable importance measures](https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-8-25) that “the variable importance measures of Breiman’s original Random Forest method … are not reliable in situations where potential predictor variables vary in their scale of measurement or their number of categories.”

**A Robust Model is a Prerequisite for Accurate Importance Scores**

We’ve seen when a mode is overfitting, the feature importances generated from the training and prediction set could be very different. Let’s apply some level of regularization by setting min\_samples\_leaf = 20 instead of 1.

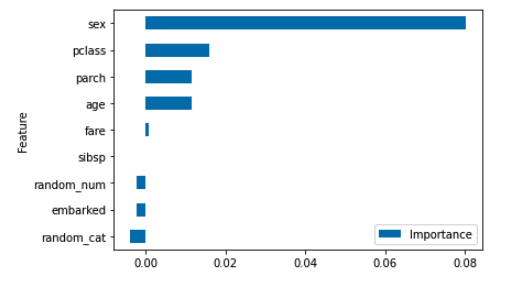
RF train accuracy: 0.810RF test accuracy: 0.832

Now let’s look at feature importances again. After fixing the overfit, the feature importances calculated on the training and the testing set look much similar to each other. This gives us more confidence that a robust model gives accurate model importances.



An alternative way is to calculate **drop-column importance**. It’s the most accurate way to calculate the feature importances. The idea is to calculate the model performance with all predictors and drop a single predictor and see the reduction in the performance. The more important the feature is, the larger the decrease we see in the model performance.

Given the high computation cost of drop-column importance (we need to retrain a model for each variable), we generally prefer permutation importance scores. But it’s an excellent way to validate our permutation importances. The importance values could be different between the two strategies, but the order of the feature importances should be roughly the same.



The rankings of the features are similar to permutation features although the magnitudes are different.

**Best Practice to Interpret Feature Importances**

**The Challenge of Feature Correlation**

After we have a robust model and correctly implement the right strategy to calculate feature importances, we can move forward to the interpretation part.

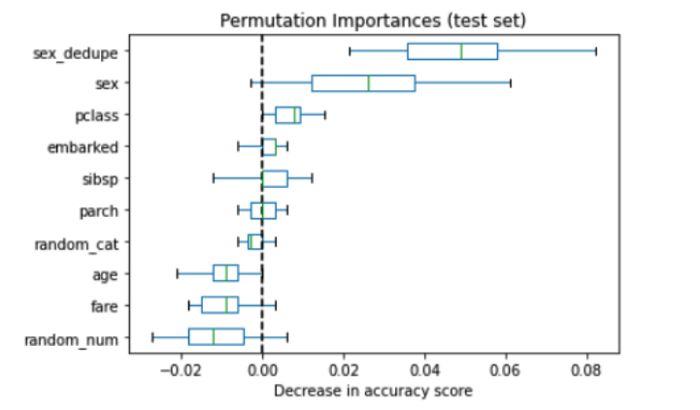
At this stage, correlation is the biggest challenge for us to interpret the feature importances. The assumption we make so far considers each feature individually. If all features are independent and not correlated in any way, it would be easy to interpret. However, if two or more features are collinear, it would affect the feature importance result.

To illustrate this, let’s use an extreme example and duplicate the column sex to retrain the model.

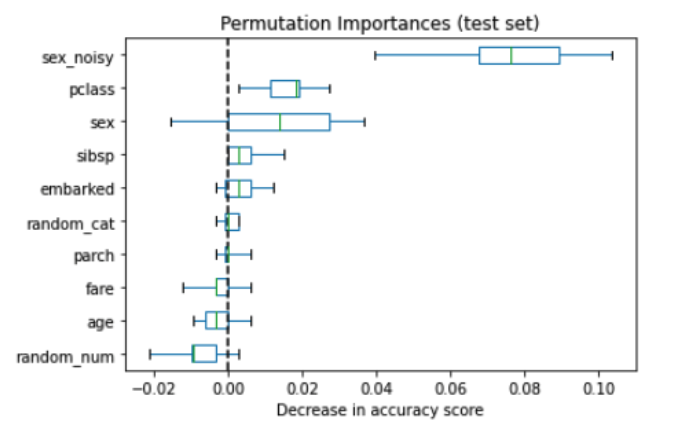
RF train accuracy: 0.794  
RF test accuracy: 0.802

The model performance slightly decreases as we added a feature that didn’t add any information.

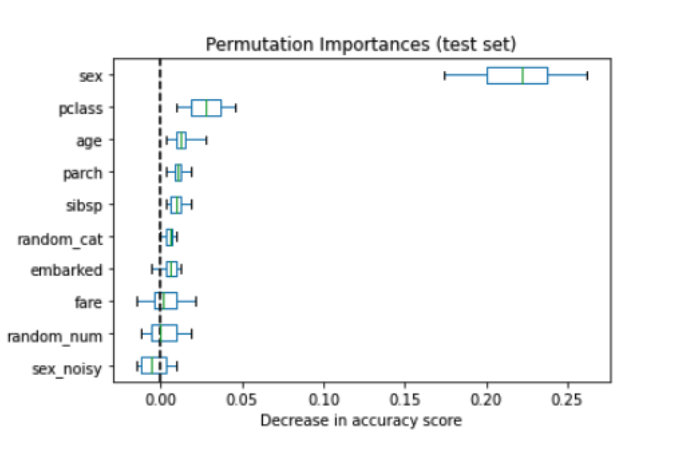
We see now the importance of the sex features are now distributed between two duplicated sex columns. What happens if we add a bit of noise to the duplicated column?



Let’s try adding random noise ranging from 0–1 to the sex.



sex\_noisy is now the most important variable. What happens if we increase the magnitude of noise added? Let’s increase the range of the random variable to 0–3.



Now we can see with more noise added, that sex\_noisy now no longer ranks as the top predictor and sex is back to the top. The conclusion is that permutation importances computed on a random forest model **spread importance across collinear variables**. The amount of sharing appears to be a function of how much noise there is between the two.

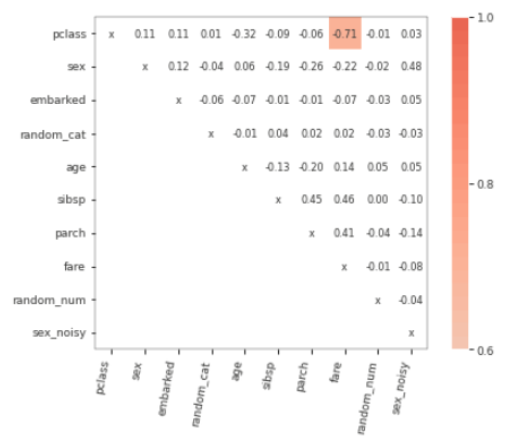
**Dealing with Collinear Features**

Let’s take a look at the correlation between features. We use feature\_corr\_matrix from the rfpimp package, which gives us Spearman’s correlation. The difference between Spearman’s correlation and standard Pearson’s correlation is that Spearman’s correlation first converts two variables to rank values and then runs Pearson correlation on ranked variables. It doesn’t assume a linear relationship between variables.

feature\_corr\_matrix(X\_train)



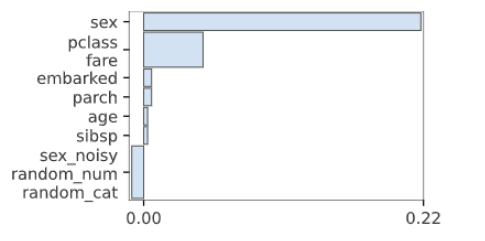
from rfpimp import plot\_corr\_heatmap  
viz = plot\_corr\_heatmap(X\_train, figsize=(7,5))  
viz.view()



pclass is highly correlated with fare, which is not too surprising as class of cabin depends on how much money you pay for it. It happens quite often in business that we use multiple features that are correlated with each other in the prediction model. From the previous example, we see that when two or multiple variables are collinear, the importances computed are shared across collinear variables based on the information-to-noise ratio.

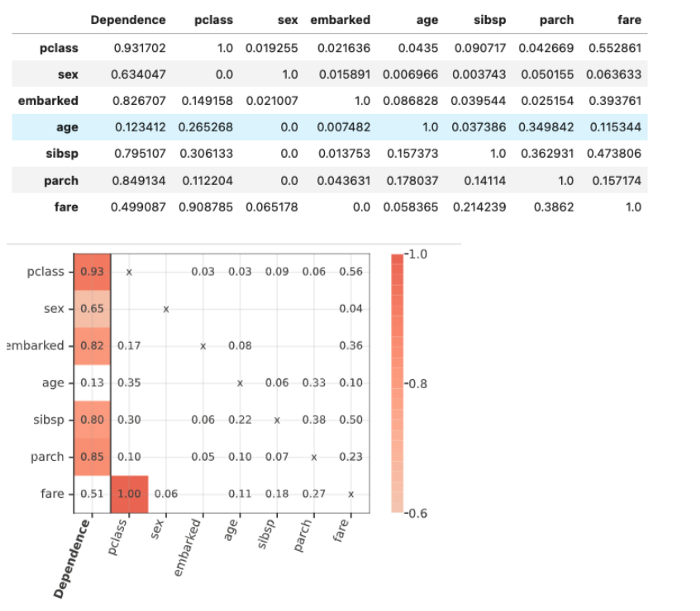
**Strategy 1: Combine Collinear Feature**

One way to tackle this is to combine features that are highly collinear with each other to form a feature family and we can say this feature family together ranks as the X most important. To do that, we will use the rfpimp package that allows us to shuffle two variables at one time.



**Strategy 2: Remove Highly Collinear Variable**

If a feature is dependent on other features, that means the features can be accurately predicted using all other features as independent variables. The higher the model’s R² is, the more dependent feature is, and the more confident we are that removing the variable won’t sacrifice the accuracy.



The first column dependence shows the dependence score. A feature that is completely predictable using other features would have a value close to 1. In this case, we can probably drop one of the pclass and fare without affecting much accuracy.

**At the End**

Once we 1)have a robust model and implement the right strategy to calculate permutation importance and 2)deal with feature correlation, we can start crafting our message to share with stakeholders.

For the common question people ask “Is feature 1 10x more importance than feature 2?”, you may understand at this moment that we have the confidence to make the argument only when all the features are independent or have very low correlation. But in the real world, that’s rarely the case. The recommended strategy is to assign features to the High, Medium, and Low impact tiers, without focusing too much on the exact magnitude. If we need to show the relative comparison between features, try to group collinear features (or drop them) to feature familiar and interpret based on the group to make the argument more accurate.

You can find the code in this article on [my Github](https://github.com/luoyuweidu1/medium-articles/blob/main/Best%20Practice%20for%20Feature%20Importance.ipynb).

**Reference**

[1] [Beware Default Random Forest Importances](https://explained.ai/rf-importance/)

[2][Permutation Importance vs. Random Forest Importance (MDI)](https://scikit-learn.org/stable/auto_examples/inspection/plot_permutation_importance.html)

[3][Feature Importances for Scikit-Learn Machine Learning Models](https://github.com/parrt/random-forest-importances)

[4][The Mathematics of Decision Tree, Random Forest Feature Importance in Scikit-learn and Spark](https://towardsdatascience.com/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e3)

[5][Explaining Feature Importance by example of a Random Forest](https://towardsdatascience.com/explaining-feature-importance-by-example-of-a-random-forest-d9166011959e)

*All images unless otherwise noted are by the author.*