Feature Selection and Importance

March 9, 2022

0.1 Executive Summary

Feature importance and selection is a critical part of inferring the results from a trained machine learning model. Some basic machine learning models give this for free - like linear regression. However, when we employ non-linear models then we often lose interpretability as many of them are practically black boxes. We can still work with trained models and employ a range of statistical and often clever tricks to assess which features are important for a given model.

This report surveys a variety of methods, starting from interrogating the data itself to implementing two proxies for feature importance - drop column and permutation importance. The attached notebook contains all the supporting code and visualizations used in this report.

0.1.1 Why feature importance matters!

A good starting point for this report is to first understand why feature selection and importance matters so much in machine learning. In my opinion, it boils down to these key reasons -

- The curse of dimensionality. A lot of features essentially results in a more sparse feature set and therefore many machine learning models dont converge to the optimal solution. Therefore, an optimal set of features leads to better results and improved model performance.
- Feature importance gives necessary context for many applications. For example, it could indicate what factors influence progression of a disease or what factors help in increasing the selling prices of houses.
- Modern machine learning pipelines are highly automated and run sophisticated data ingestion activities in a production environment. Data engineering teams ensure that the integrity of this pipeline is maintained and spurious input data is minimized as much as possible. Therefore, having lesser features needed to generate inferences or predictions from a machine learning model results in a simpler pipeline and therefore less maintenance overhead.

The biggest catch to be aware of is that feature importance is model dependent. A linear regression might not find a particular feature important but a non linear model like SVM could. So, we should always assess feature importance as what a stable and well trained model thinks is useful in predicition.

0.1.2 Dataset

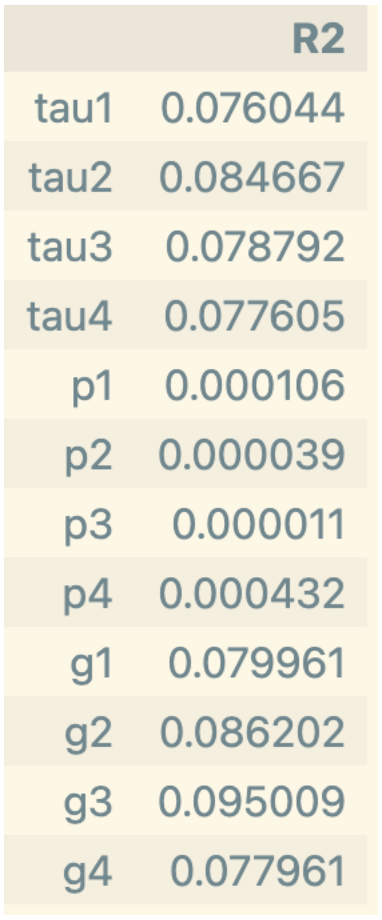
We are addressing a multivariate regression dataset for this report. The dataset was sourced from UCI ML repository with the link provided below. https://archive.ics.uci.edu/ml/datasets/Electrical+Grid+Stability+Simulated+Data+

The dataset contains twelve features corresponding to electricity production, nominal demand and price elasticity. The target is to predict grid stability for different values of the twelve predictors. All the columns are numeric in nature. The dataset has 100,000 observations recorded with no missing values.

0.1.3 Inferring importance directly from data without any machine learing model!

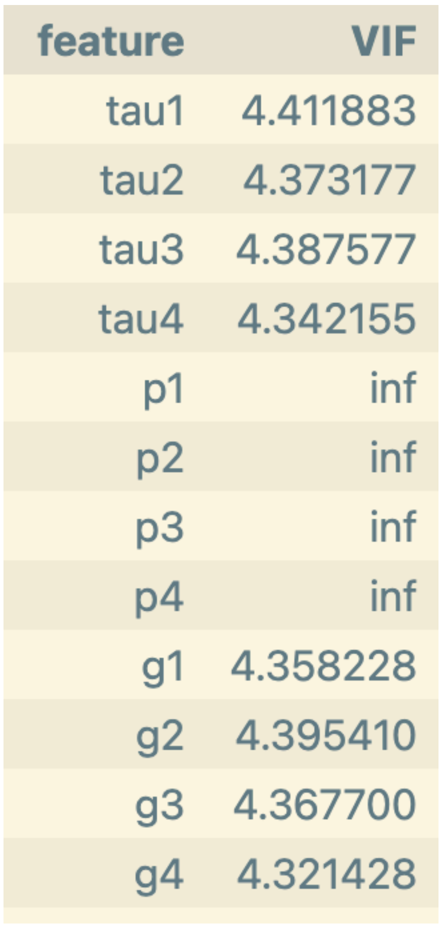
1. Principal Component Analysis One common techinque for dimension reduction is the principal components analysis. The principal components of a collection of points in a real coordinate space are a sequence of p unit vectors, where the i-th vector is the direction of a line that best fits the data while being orthogonal to the first i-1 vectors. Here, a best-fitting line is defined as one that minimizes the average squared distance from the points to the line. These directions constitute an orthonormal basis in which different individual dimensions of the data are linearly uncorrelated. Principal component analysis (PCA) is the process of computing the principal components and using them to perform a change of basis on the data, sometimes using only the first few principal components and ignoring the rest.

2. Correlation of each feature to the target We can look at the R2 score of each individual feature's regression with the label as another proxy for relative feature importance. When we perform the regression of each of the eight features we notice that four features have low R2 scores. One major drawback of this approach is it ignores interaction between features and co-relation in real world data. For example, two features could have a positive interaction effect as presence of one improves the overall regression results but looking at each one of them in isolation gives a limited view. Additionally, having correlated features in the data produces spurious results as the dependent columns will display high R2 scores if the independent features have high predictive power. So in effect, they are getting carried by the key features.



	R2
tau1	0.076044
tau2	0.084667
tau3	0.078792
tau4	0.077605
p1	0.000106
p2	0.000039
p3	0.000011
p4	0.000432
g1	0.079961
g2	0.086202
g3	0.095009
g4	0.077961

3. Checking for co-dependence One could also consider a simple correlation matrix to inspect the presence of strong correlation among features. This would work if a feature was correlated to another single feature. This of course fails in case of multicollinearity. In these cases, inspecting the so-called Variance Inflation Factor gives a good idea about which features contribute to multicollinearity. The idea is very simple - we regress each feature against the other features and compute the so called VIF factor, $\frac{1}{1-R^2}$. Any feature with a VIF factor larger than 10 is typically considered as contributing to multicollinearity. Running VIF analysis on our data indicates that four features -p1,p2,p3,p4 - are extremely correlated and are causing issues in the data structure.



feature	VIF
tau1	4.411883
tau2	4.373177
tau3	4.387577
tau4	4.342155
p1	inf
p2	inf
p3	inf
p4	inf
g1	4.358228
g2	4.395410
g3	4.367700
g4	4.321428

0.2 Linear models

Linear models provide an easy to understand interpretation of our predictor variables. If our feature columns are scaled and standardized, the regression co-efficients provide valuable information. If a variable has a higher influence on the response then its co-efficient's absolute value would be larger than the others. To highlight this, we perform a regression on the scaled input data for

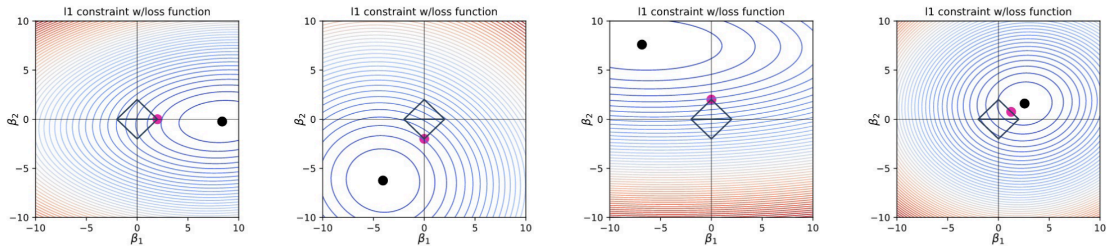
L1 regularization, or lasso regularization, is an method for constraining the regression co-efficients so that they dont overfit and helps the model fit diverse data in a more generalized fashion. This is achieved by adding a the co-efficient weights to the loss function. Therefore, the training process would prefer lower co-efficents weights to minimize the loss.

$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j$

to

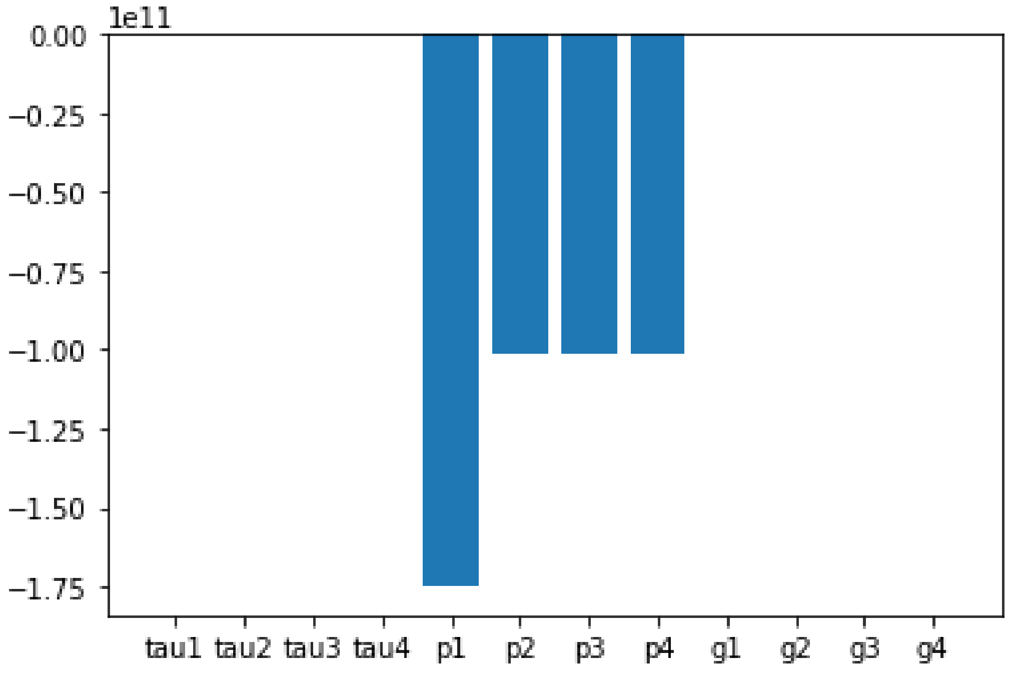
$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j + \lambda |\hat{\beta}_j|$

The λ is the regularization parameter and decides how punitive we want to be when shrinking the co-efficients. As a result of this regularization, some co-efficients are dropped to zero. To understand why this occurs, the visualization below simplifies a regression cost function to two variables

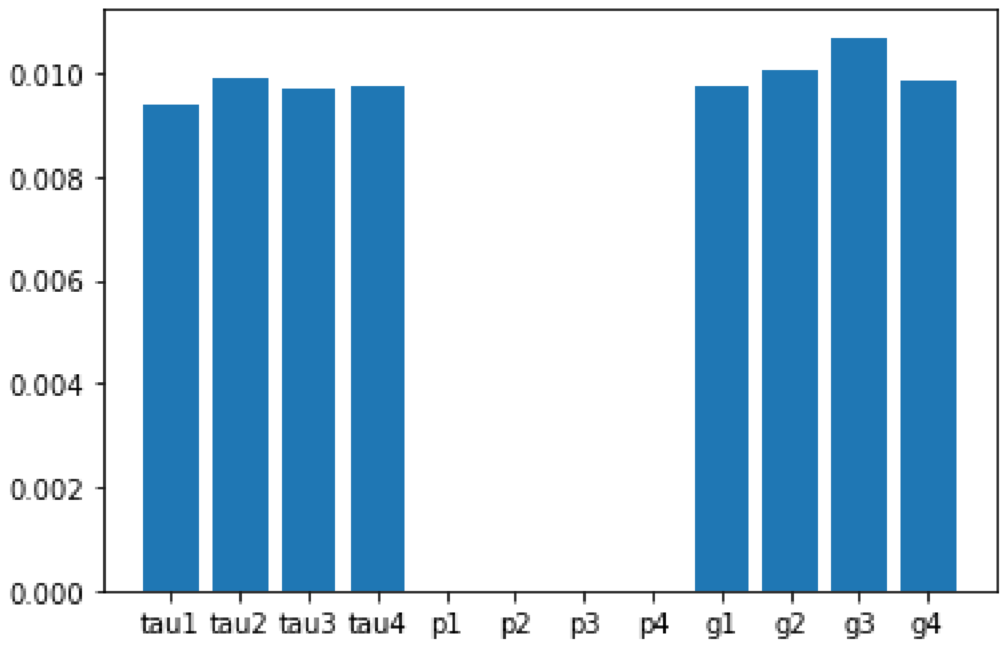


The contours in the plot above indicate different values of training loss depending of combinations of β_1 and β_2. The diamond region is the boundary condition representing the absolute penalty introduced in the L1 regression loss function. We are interested in the loss contour that "hits" this diamond penalty region - as that would be the lowest possible loss value that satisfies the loss function with regularization. This also implies that a contour could meet with our constraint region at the vertices more often. Therefore, the L1 regularization algorithm can set some regression co-efficents to zero. One way of interpreting this behaviour is those variables are not significant enough for L1 regression.

If we fit our scaled dataset to an OLS model, we notice some of the co-efficients are very large and it is impractical to undertstand their utility.



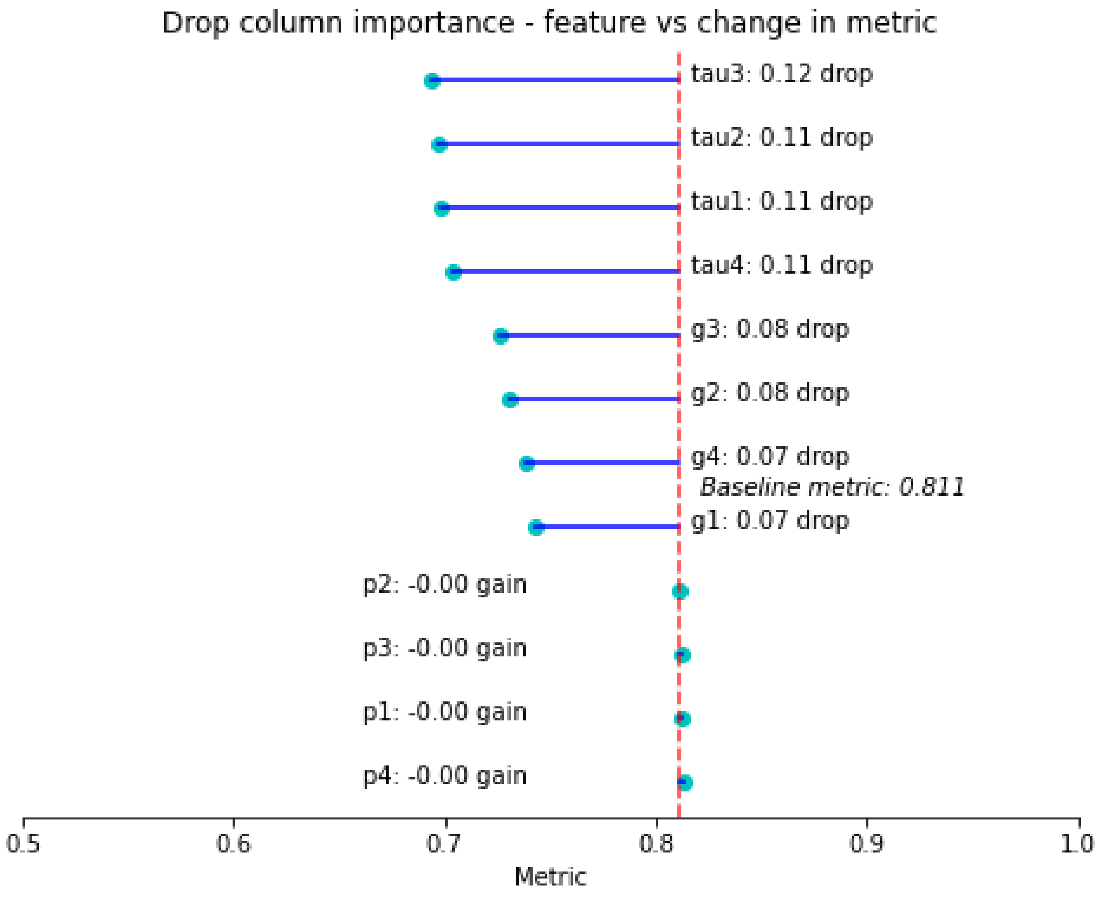
Applying L1 regularization however produces a more stable set of co-efficients and we notice that four features - p1, p2, p3 and p4 - have been set to zero. We also notice that the validation set adjsuted score R2 has dropped marginally to 0.66 from the unregularized score of 0.68. Therefore, L1 regularization gives us a more compact set of features with reduced bias and improved variance.



While all of this sounds great in theory, we have to consider the limitations of linear models. Due to their setup and construction, they are not able to extract all signals from the training data. A more sophisticated non linear model would better utilize the dropped features and improve upon the metric of interest low R2 scores. Therefore, using L1 regularization as a proxy of feature importance is not ideal thought it does give us some information on which features are more useful than others.

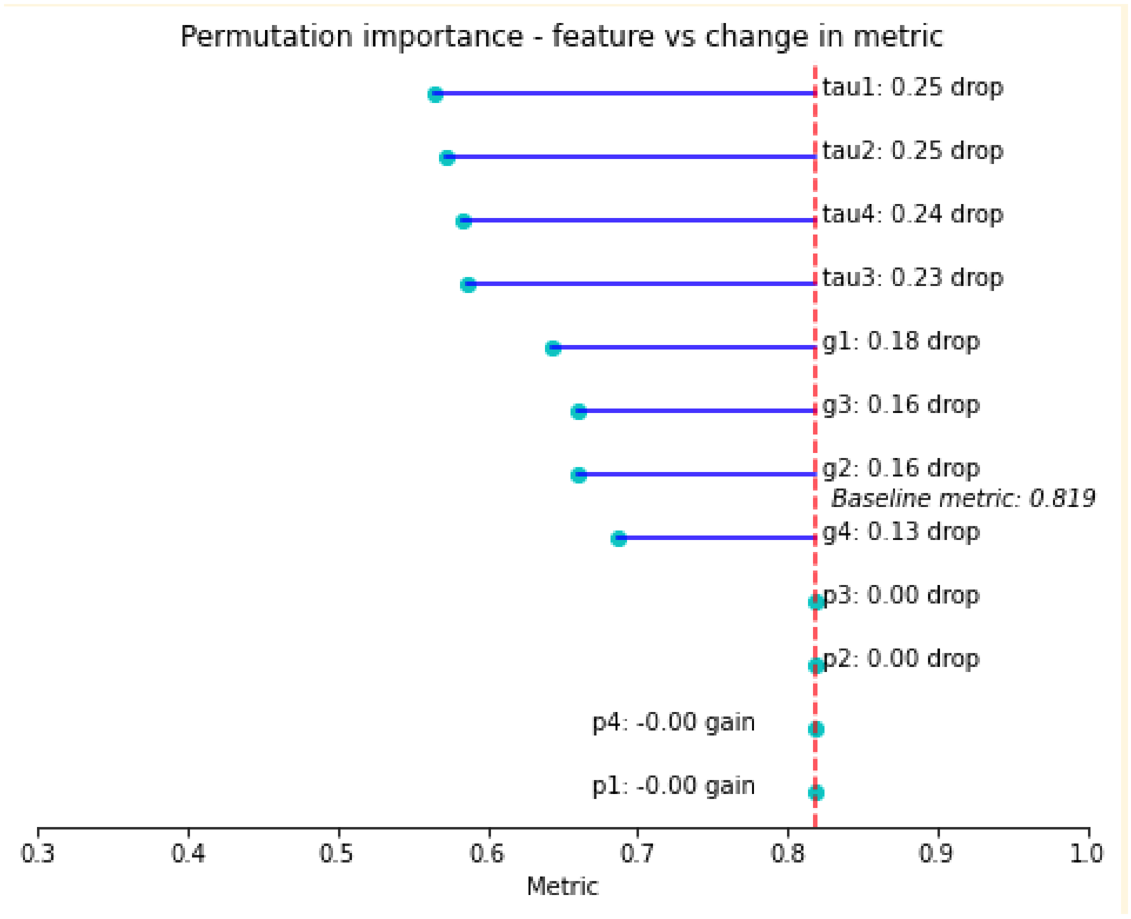
0.2.1 Drop column importance

This method is a very direct way of estimating feature importance - The core idea being that the important features case a much bigger impact to the performance of the model. We train a model on our data and compute a baseline estimate with all the features present. Then we delete each column and train a new model on it and compute its metric of interest with the validation dataset. Once complete, we can plot the results to visualize how the loss of a feature affects the model's performance. When we apply drop column to our dataset, we observe the following



0.2.2 Permutation importance

This approach is similar to drop column but instead of removing a feature entirely, we scramble its contents randomly. This would break the relationship of that feature to the target variable and hence make it act as random noise. We train the model on the whole data and work out a baseline metric level on the validation set. Then we progressively scramble each column of the validation set and work out the new metric. We collect all the results and plot it below for our dataset. The baseline R2 score for a Random Forest Regressor was 0.81 and we show the change in the metric when we scramble each column. We find the features related to price elasticity dont contribute much to the model - they are essentially random noise for our model. However, the features related to power production are very import as randominzing each of them produces a material impact in the model's performance



0.2.3 Comparision between drop column and permutation importance

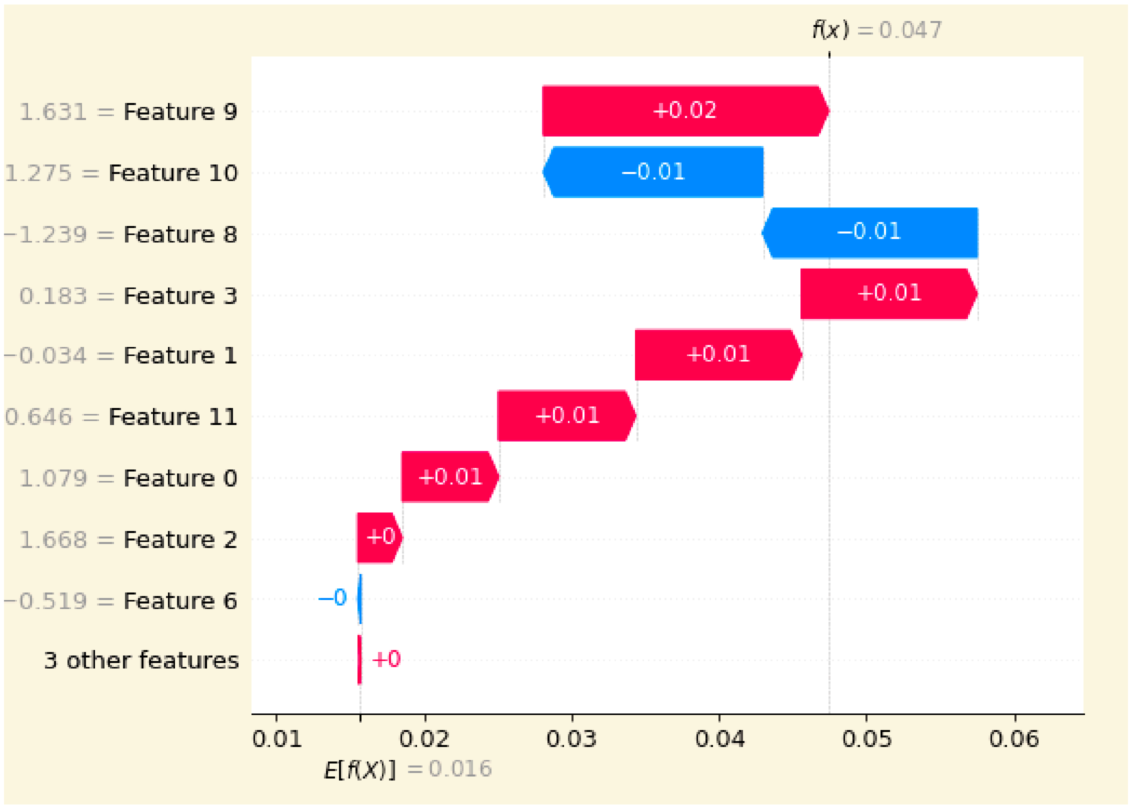
We notice some very interesting trends when comparing the outputs of drop column and permutation importance. The common theme is that both found four features to be of very low interest - p1, p2, p3, p4. Our earlier tests with L1 regularized regression also arrived at the same conclusion. So this is additional confirmation that a more "powerful" non linear model like Random Forest didnt find these features meaningful in prediction. Interestingly, both methods dont necessarily find the same features to be the most important. Permutation importance algorithm found "tau1" to have the most impact on randomization but drop column ranked it as the third most important feature.

It would be helpful to highlight the differences among the two methods and how they deal with our nemesis - codependent features!

- Drop column importance is costly to run. It needs to train the whole dataset - sans one column - everytime. However, the representation of data is more robust. Permutation importance is much faster in practice as we reuse the same model multiple times. However, it could be argued that some of the records might be nonsensical after scrambling a column, like pregnant males etc.
- Drop column tends to show 0 or low importance for co-dependent features. Ideally we would want to understand which one of the co-dependent feature to retain. Permutation importance spreads the importance of co-dependent features equally.

- In short, collinearity is a big problem and needs to be addressed through careful study of the data with subject matter experts. We encode dependent columns to create new features, like BMI instead of height and weight etc.

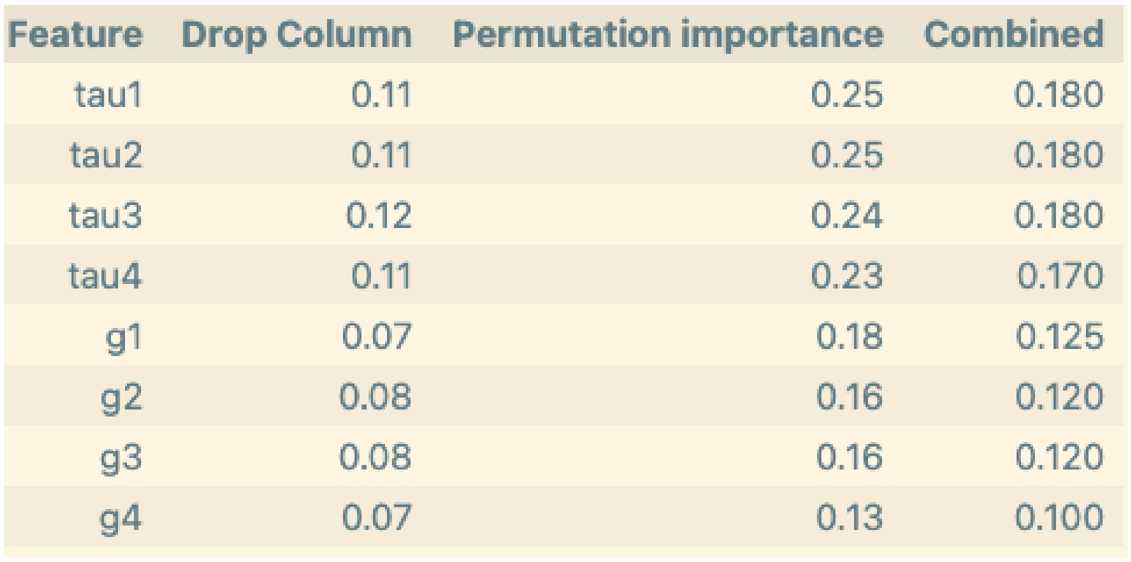
Comparing our results with SHAP indicates that we are on the right track here. We trained an XGBoost model on the same dataset and ran SHAP algorithm on the trained model. While the approach is different from drop column or permutaiton importance, it found features p1, p2, p3 and p4 to be less important in maximizing the expecation of the response.



0.3 Automatic feature selection

With the implementation of two algorithms to quantify feature importance - drop column and permutation importance - we can come up with a basic algorithm to select k important features from a dataset. We achieve this by following this algorithm

1. Normalize the data to convert the numeric features to the same scales
2. Perform drop column importance and only retain columns which harm the baseline on removal
3. Perform permutation importance and retain columns which harm the baseline
4. Average the "drop in metric" for each feature from step 3 and step 4.
5. Rank the features in an ascending manner based on the average score.
6. Select min(k, number of features) features from step 5



Feature	Drop Column	Permutation importance	Combined
tau1	0.11	0.25	0.180
tau2	0.11	0.25	0.180
tau3	0.12	0.24	0.180
tau4	0.11	0.23	0.170
g1	0.07	0.18	0.125
g2	0.08	0.16	0.120
g3	0.08	0.16	0.120
g4	0.07	0.13	0.100

0.4 Conclusion

Surveying literature and implementing code for this project helped my understanding of feature importance. The algorithmns and methods highlighted in this report help a lot in assessing which features are redundant and could be dropped entirely from our pipeline. But as I found out, there is no clear definition of feature importance. We can quantify the effect a feature has on our target but that is dependent on the model and how it utilizes the features.

The most important part in the ML pipeline is to understand the data first and ensure that co-dependent features are minimized at the start. We could ask a subject matter expert to inspect the data and indicate potential problems. We could validate that again using PCA or our top k algorithm to determine the important features.