Feature Importance Methods for Black Box Models

William Holt, Michael Wise

Dr. Duy Nguyen

DATA 477 Capstone

**Abstract**

Feature importance for black box models is a very important step in creating robust black box models. There are several methods that can be used in estimating the predictive power of the various variables in a model. The more basic ones of drop-and-relearn and permute-and-relearn may struggle to give an accurate estimate of feature importance as they are highly affected by variable correlation. The more advanced methods such as SHAP, LIME, and MDI may offer much more insight into complex models. Here we will show how these methods differ and their usefulness for a variety of different datasets. We will be testing the methods on random forest models trained on these datasets as well as comparing them to linear or logistic regression models based on the target variable type. We will also briefly cover feature importance for images using LIME using two different segmentation algorithms.

**Table of Contents**

[**1) Introduction to Feature Importance & Selection 3**](#_smbz9fpt6csc)

[**2) The Black Box Problem 4**](#_g5gor1nkmyw5)

[**3) Overview of Chosen Datasets 4**](#_csarknt3d4a1)

[3.1) Heart Disease Dataset 5](#_igf32da3ikob)

[3.2) World Happiness Report, 2005-Present 5](#_7ki91vj718sw)

[3.3) Data Science Job Salaries 5](#_8tg31nkrwpva)

[3.4) MNIST 5](#_gjdv823mh0vy)

[4) Permutation and Drop-and-Relearn Feature Importance 6](#_ky3zaprs6e6c)

[4.1) What is PFI? 6](#_flqj4zm8vwh1)

[4.2) What is Drop-and-Relearn? 7](#_u0knoucfs2fj)

[4.3) Mean Decrease in Impurity (MDI) 7](#_5g8j7u38d5)

[4.4) Implementation 8](#_act6di8gey8v)

[**5) Local Interpretable Model-Agnostic Explanations (LIME) 9**](#_coc21sncllnk)

[5.1) What is LIME? 9](#_dcn3kuly636w)

[5.2) Implementation 11](#_xjb3uqwiknr1)

[**6) SHapley Additive exPlanations (SHAP) 11**](#_yq2lntwdoirk)

[6.1) Shapley Values 11](#_cg57uvavafqv)

[6.2) Monte-Carlo Estimation with SHAP 13](#_nz6l7ggku23t)

[6.3) Implementation 14](#_b42oei6n50k7)

[**7) Example: Data Scientist Salaries 15**](#_3hwlare0he5t)

[**8) Example: Heart Disease 18**](#_hszp5x1i760y)

[**9) Example: World Happiness Report 21**](#_qufjvfq776z6)

[**10) Image Feature Importance 24**](#_hwgcmkhlxzpj)

[**11) Example: MNIST Digits 25**](#_xah7auc2uh7l)

[**Conclusion 27**](#_keots4aagbc3)

# 

# 

# 1) Introduction to Feature Importance & Selection

Machine learning models are used to pivotal in allowing us to make predictions based on some sort of data that is inputted. These predictions allow us to make actionable insights on whatever problem we are facing, but it is foolish to think that *all* of the features or variables in the input data are equally important for making accurate and efficient models. Variable selection is extremely fundamental in building robust models. Reducing the number of features we use has many desirable effects. Primarily, it reduces the computational cost of the model and in many cases can improve the performance of the model. [1] In order to know which variables to keep and which to eliminate, there is a technique called **feature importance.** In general terms, feature importance refers to any technique that calculates a score or threshold for all of the input variables given some machine learning model. Usually we are looking for the features with the highest scores or the lowest scores (depending on the context).

There are several methods for measuring feature importance and shrinking the model to eliminate the least important features. The most common method for looking at variable importance is the p-value for traditional regression models. The p-value is a measure of the chance that the correlation of the output variable to the input variable could occur by chance. These metrics are derived by finding the probability of observing a t-statistic, or the ratio of the departure of the estimated value of a parameter from its hypothesized value to its standard error. If the p-value is less than a pre-specified significance level (typically 0.05), we reject the null hypothesis and conclude that the coefficient is statistically significant. Thus, removing variables with higher p-values is a common way of shrinking regression models. While this method is effective for models that are clearly parametric, this technique isn’t that ideal beyond simple regressions. Some other methods include lasso and ridge regressions which include a penalty for each variable added. Then, if a variable isn’t able to make up for this penalty with the new information that it brings to the model, it is removed. This penalty is usually applied to the R^2 value. We can also combine the logic from these two and perform elastic net regression.

# 2) The Black Box Problem

The above methods work very well, but they only work for traditional regressions. Neural networks, for instance, only produce a series of weighted nodes for the variables to travel through. It is impossible to determine how effective an individual variable is from the weights and the model alone. This brings us to the primary problem we are addressing - the **black box problem**. [2] This problem refers to the lack of transparency and interpretability of machine learning models. We consider a model ‘black box’ when it can make accurate predictions, but often the internal workings of the model are not easily understandable or interpretable. Other types of ‘black box’ models such as decision trees and ensemble models also don’t offer any insight into variable importance. This is one of the disadvantages of using machine learning models over traditional regression models. There are some cases when the data are so complicated that only a machine learning model can be used, though. There are a few techniques to finding the variable importance of these models that have been developed. The ones that we will cover in this paper are permutation variable importance, drop-and-relearn variable importance, LIME, SHAP, conditional permutation importance, and impurity tests for tree-based methods.

# 3) Overview of Chosen Datasets

For our analysis, we wanted to try feature importance techniques on a variety of different data sets and machine learning tasks. We chose a good balance of data sets that deal with both classification and regression with varying levels of variable collinearity, complexity, and data size. Briefly, we will introduce each of them.

## 3.1) Heart Disease Dataset

This dataset is approximately 300 values with 13 predictor variables and 1 target variable. It deals with predicting heart failure based on several measurements and reported metrics such as level of pain. This is a binary classifier dataset.

## 3.2) World Happiness Report, 2005-Present

This is a dataset containing several metrics and a World Health Organization happiness metric for every country from 2005 to the present. This dataset has real world applications for governments trying to make their people happier. By looking at what variables are most important they could focus their efforts on those specific areas. This is a regression dataset.

## 3.3) Data Science Job Salaries

This is a dataset that includes 600 data science related positions and the pay rate for each. This is a useful dataset to help determine what factors impact the pay rate of a data science job the most. It could also help you see if you’re being underpaid or not. This is a regression model with 11 predictor variables and 1 target variable.

## **3.**4**) MNIST**

Dataset consisting of 70k labeled images of handwritten numbers. This will really show how the machine learning variable importance metrics work as this is a dataset that is too complicated for linear regression.

# 4) Permutation and Drop-and-Relearn Feature Importance

## 4.1) What is PFI?

Permutation feature importance (PFI) is a basic variable importance method that involves taking each variable one at a time and permuting all of the values in that variable. Then the model is re-trained and the differences in accuracy are measured. If the model’s accuracy is very similar after the variable is permuted, then that variable is likely not very important and can be removed. If the accuracy drops significantly, then the variable is likely very important to the model and shouldn’t be removed. Scikit-learn has a great explanation of permutation feature importance that can be implemented very easily. The algorithm for computing the importances can also be broken down as follows [3]:

| **PFI Importance Algorithm**  Given fitted predictive model *m*, and tabular dataset *D*:   * Compute the reference score of the model *m* on data *D* (most commonly accuracy for classification or *R2* for regression) * For each feature *j* (column of *D*):   + For each repetition *k* in 1, … , *K:*     - Randomly shuffle column *j* of *D* to generate a randomized copy of data called .     - Compute score *sk,j* of model *m* on corrupted data   + Compute importance *ij* for feature *fj* defined as: |
| --- |

A drawback to this method is that highly correlated values will show up as far less important than they actually are because the information of the one feature is not actually lost when permuting it if it is highly correlated. This can cause models to shrink too much and is not an accurate method unless there is very little variable correlation which is hard to know for certain. It is also very important to know that this depends heavily on the performance of a model. Features that could be found with low importance for a bad model could be very important for a well-performing model. Nevertheless, because this relies solely on the data and the accuracy score we are giving the model, this can be applied to any black box ML implementation.

## 4.2) What is Drop-and-Relearn?

Next, we will mention a technique that is similar to PFI which is called drop-and-relearn. These methods were explored in Lei et al. under a different name they refer to as leave-one-covariate-out methods [4], but the math is identical. This algorithm is very similar to PFI except that instead of permuting a variable, it is removed entirely. The measurement method is the same where the accuracy and other performance metrics are compared between the two models and a decision is then made as to whether or not the variable should be removed. It has similar drawbacks to PFI as well, as highly correlated features will appear to be much worse than they actually are. Removing a highly correlated feature doesn’t actually remove its information, so the model will still appear to be highly accurate when it is removed.

## 4.3) Mean Decrease in Impurity (MDI)

This is a technique that falls under the same umbrella of these other models, except it only works for tree and ensemble models. Without going into intricate detail, this method is still looking at the mean decrease, but rather in accuracy, we are looking at impurity. Specifically, tree structures have nodes that each have their own Gini impurities, which are described by assigned a weight related to the probability of reaching a certain node. This is approximated by the proportion of samples reaching that node and its child nodes if it has any. The node impurity is averaged over all the trees that correlate to each feature class. Using this terminology, the main difference between this and PFI/Drop-and-relearn is that they measure “mean decrease in accuracy” while MDI measures “mean decrease in impurity”. It is the same logic, just for a different metric that applies specifically to trees. This is one of the main metrics used for training trees - the optimal solution is to just take the feature split with the lowest weighted node impurity. We will use this in our analysis as one of our base models is a random forest regressor, which is perfect for this.

## 4.4) Implementation

These methods are easily implemented as they are the most basic. A for-loop looping through each variable and either permuting it or dropping it and then retraining the model is very easy to implement. There is no concrete method for determining whether or not to drop a variable using these methods though. A certain percentage or amount of accuracy drop could be used. Other metrics can also be taken into consideration such as precision and recall. For regression based models, the *R2* value can be looked at. We already outlined the permutation example, so for a hard-code example we will show how to perform drop-and-relearn in a regression context. In Python, the method is defined by our *x* data (features) and *y* data (what we’re trying to predict):

def drop\_learn(x, y, num, cat, model):

fit = drop\_fit(x, y, num, cat, model)

r2 = r2\_score(fit['y\_val'], fit['y\_pred'])

ac\_df = pd.DataFrame(columns = ['Variable Removed', 'R2'])

ac\_df = ac\_df.append({'Variable Removed':'None', 'R2':r2}, ignore\_index=True)

for i in range(0, len(x.columns)):

x\_drop = x.drop(x.columns[i], axis=1)

num\_drop = [j for j in num if j != i]

for j, v in enumerate(num\_drop):

if num\_drop[j] > i:

num\_drop[j] = v-1

cat\_drop = [j for j in cat if j != i]

for j, v in enumerate(cat\_drop):

if cat\_drop[j] > i:

cat\_drop[j] = v-1

fit = drop\_fit(x\_drop, y, num\_drop, cat\_drop, model)

r2\_metric = r2\_score(fit['y\_val'], fit['y\_pred'])

ac\_df = ac\_df.append({'Variable Removed':x.columns[i], 'R2':r2\_metric}, ignore\_index=True)

ac\_df['Difference'] = ac\_df['R2'] - r2

return ac\_df.sort\_values(by=['Difference'], ascending=True)

While the drop\_fit()method is not shown here, it is available in our code and just consists of a small data pipeline that one hot encodes the categorical data and uses a standard scaler on the numerical data, which is then fit on whatever model we give it. And as mentioned above, we use *R2* as our accuracy metric. For all of the following examples, we try it on both linear regression and a random forest regressor. There exist many valid interpretations of these methods, so it is up to the modeler to use his or her best judgment on whether or not to remove a specific variable.

# 5) Local Interpretable Model-Agnostic Explanations (LIME)

## 5.1) What is LIME?

The next technique we introduce is based on local surrogate models. This is called LIME, which was proposed by Ribeiro et al. in 2016 [5]. The goal of LIME is to provide local explanations for individual predictions, as opposed to global explanations for the entire model. LIME achieves this by approximating the complex, non-linear model with a simple, interpretable model that is much easier to understand. To create a local surrogate model, LIME first peturbs the input data around the instance of interest and measures the effect of feature on the model’s prediction. By adding some noise to our original data and feeding them into whatever black box model we are working with, we are able to assign weights to the new data points as some function of their proximity to the original point. The result is a set of weighted features that are used to train a simple model like a linear regression or decision tree on the locally perturbed data.

Once the local surrogate model is trained, it can then be used to explain the prediction as the coefficients actually are able to be examined (unlike the underlying black box model we are using). For example, if the surrogate model is linear regression, we might use one of the similar techniques like p-value importance LIME fits a surrogate model (like linear regression) with said sample weights, creating a connection between the original points and the newly trained model. LIME also provides a measure of the quality of the explanation, called the local fidelity score. The fidelity score measures how well the local surrogate model approximates the original black box model for the instance of interest. A higher fidelity score indicates a more accurate approximation and a more reliable explanation. [6] Mathematically, we can represent the explanation models using the formula



where *x* is the instance, *f* is the original model, and *g* is the surrogate model that minimizes loss *L* (usually mean squared or absolute error). The loss measures how close the explanation is to *f*’s original prediction. There is also an additive model complexity (*g*) that we try to keep as low as possible (maximum is selected by the user). The *x* is called the proximity measure, which defines how large the neighborhood around our instance that we consider for our explanation. Once we get a value for one explanation, we can proceed to do it over and over and get an average weight of importance for each feature.

## 5.2) Implementation

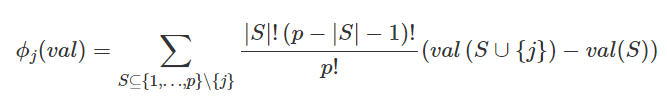
There does exist a LIME Python package [7] which we will use for our implementation, analysis and explanations. All that is required is that the classifier implements some function that takes in raw text or a numpy array of data and outputs probability for each class (which pretty much every standard ML classification model does nowadays). Interestingly enough, LIME actually supports more than just tabular data, as it is also able to process features for textual and image data. We have one example in a later chapter where we wanted to test this out using the MNIST dataset.

In our Python implementation, we choose linear regression as our interpretable surrogate model. We opt to use an exponential smoothing kernel on the normalized data to define the neighborhood of our points. One fallback with LIME that we foresee is due to local behavior being approximated by only linear models in the current implementation (like regression) it will fail to accurately capture variable importance for more complex machine learning models. However, one advantage we expect it to have is its ability to interpret categorical features well (and numerical features too, but this requires categorizing them into bins).

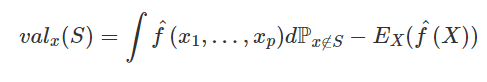
# 6) SHapley Additive exPlanations (SHAP)

## 6.1) Shapley Values

To be honest, SHAP deserves an entire paper to do it justice. To properly understand how it works, you need an understanding of the game theory behind Shapley values as well as the actual feature importance methods and its many variations (KernelSHAP, TreeSHAP). We will try to explain SHAP with a very high-level approach. As another alternative to LIME and PFI, we can use more mathematically driven methods for explaining the predictions of black box models. SHAP uses an approach rooted in game theory for calculating how much each feature’s importance. This requires us to define a Shapley value, which is usually denoted by a value function *val* of players in a set S. The Shapley value of a feature value is its contribution to a “payout” (prediction), weighted and summed over all possible feature value combinations



where S is a subset of the features used in the model, *x* is the vector of feature values of the instance to be explained and *p* the number of features. [8] We also define the prediction for feature values in set S that are marginalized over features that are not included in set S as:



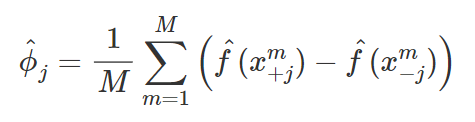
We can apply this logic to a ML interpretation. It’s also interesting to note that because this is rooted in game theory, Shapley values have 4 main properties (axioms) [8] that help provide explanation and sure fairness of “payouts” (equal representation of our predictors).

* *Property 1. (Efficiency)* The feature contributions must add up to the difference of prediction for x and the average.
* *Property 2. (Symmetry)* The contributions of two feature values *j* and *k* should be the same if they contribute equally to all possible coalitions.
* *Property 3. (Dummy)* A feature *j* that does not change the predicted value – regardless of which coalition of feature values it is added to – should have a Shapley value of 0.
* *Property. 4 (Additivity)* For a game with combined payouts val + val+ the respective Shapley values are

The last property is important for tree-based models as additivity guarantees that for a variable, you can calculate the Shapley value for each tree individually, average them, and get the Shapley value for the variable for the random forest.

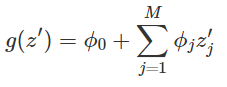
## 6.2) Monte-Carlo Estimation with SHAP

You might notice that the equation for the value function is a nasty integral, which is nearly impossible to calculate analytically. While the theory will give us great measures of feature importance, we rarely can get an exact solution for the Shapley values, especially as the number of predictors (“players”) increases. That would require evaluating the exact Shapley value of all possible sets of our feature space. Strumbelj et al. (2014) [9] provide a way to approximate the value using Monte-Carlo sampling:

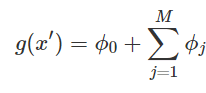


Thus, if we want to actively apply this to a robust model, we will need to approximate the values. represents the prediction for *x*, but we take a random number of feature values and replace them with predictors from a random data point *z.* Inside the sum, it is saying take original observations for features that appear left of *xj* and take random observations for those that appear to the right of *xj*. We are using the order purely as a trick to give us a random mechanism for creating a hybrid that is assembled from two separate instances each *M*th time. Like LIME, it also is an individualized model-agnostic explainer, meaning we can apply this to any black box model and still extract feature importance just from using the input data and the model’s predictions.

When applying it to an actual model, we can specify each explanation as:



where *g* is the explanation model and *z’* ∈ {0,1}M is the coalition vector of all of our features. *M* here is the maximum amount of features (coalition size) where the Shapley values are represented by for the *j*th feature. The model simplifies when we assume that *x’* is the vector of all 1’s, i.e. instances where all of the feature values are present. This eliminates the vector giving us:

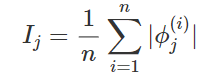


This is how the SHAP algorithm approximates its values. While we excluded a lot of the more tedious mathematics behind the algorithm, it’s important to show how it’s implemented in code and the benefits of using it over other methods.

## 6.3) Implementation

Coding the SHAP algorithm from scratch in Python along with effective approximation is possible, yet very time consuming. Luckily, there is a SHAP Python library [10] that allows for easy implementation on our chosen data. We have shown that while the game theoretical approach is very desirable as we are given exact metrics, calculating said values is computationally expensive, which is why this library uses a kernel approach for approximation. The SHAP library also offers plotting tools that we can use to visualize our results.

For our purposes, we assume for feature importance that we want the largest absolute Shapley values for each of our variables. After all, we are looking for the global importance of the features in relation to our entire black box model. The values are simply calculated by:



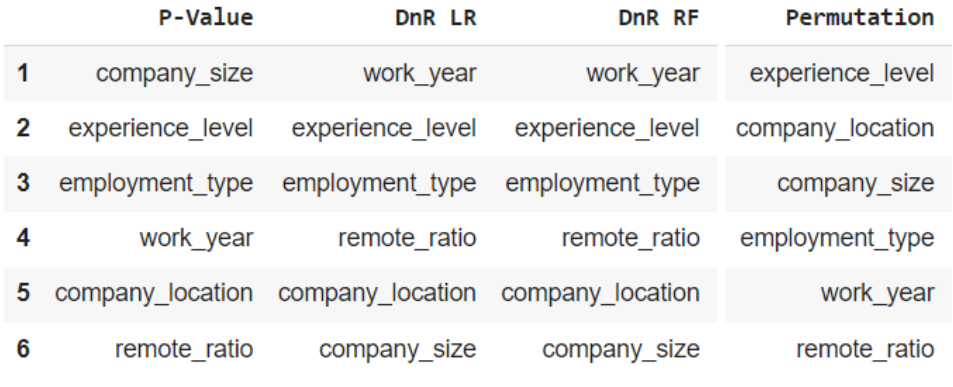
where the Shapley values *j* are approximated using the methodology used above in combination of something called kernel/tree approximation (which we will not get into here).

The only thing left once the features are approximated is to sort the features by decreasing importance, which we will subsequently do with every model to do an overarching comparison later on. The main advantages of SHAP are the solid theoretical foundation and the quick implementation for tree-based models. Some disadvantages is that it can still be computationally slow (mainly KernelSHAP), but it largely depends on the type of model and method of approximation you are using. The other drawback is that Shapley values can be misinterpreted.

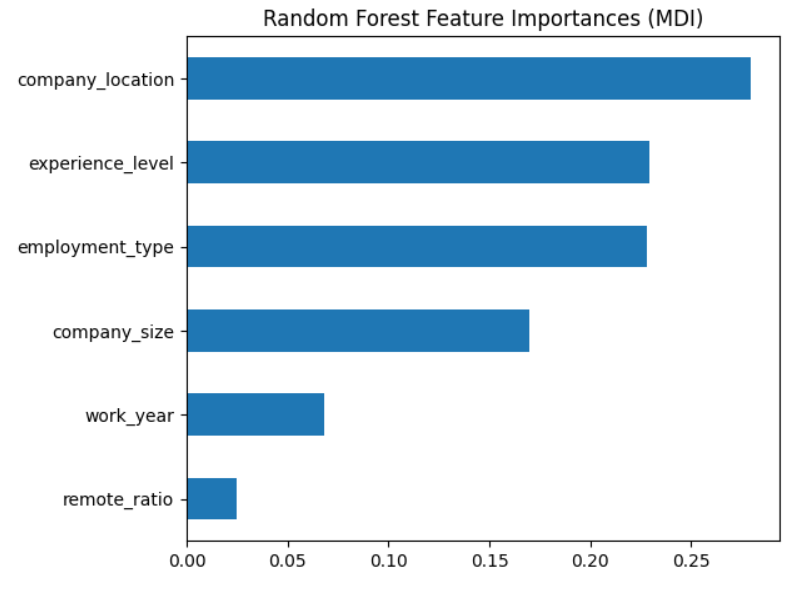
The values are the contribution a feature value to the difference between the actual prediction and the mean prediction is the estimated, and *not* the difference in predicted value after removing a feature from training. Next, we will look at all these methods and try them out on various datasets and machine learning problems, accounting for both classification and regression.

# 7) Example: Data Scientist Salaries

The data science salaries dataset is a collection of salaries for data scientists from 2020 to 2022 aggregated by ai-jobs.net. It consists of primarily categorical data such as work location, company size, and experience level. The dataset was paired down to be only english speaking countries in order to make the dataset easier to work with. This dataset will be interesting to work with because it is difficult to determine variable correlation when there are only categorical variables. The prevalence of categorical variables may affect how well the more basic methods of drop-and-relearn and permutation importance work compared to the more advanced methods. The results from the more basic methods are as follows.

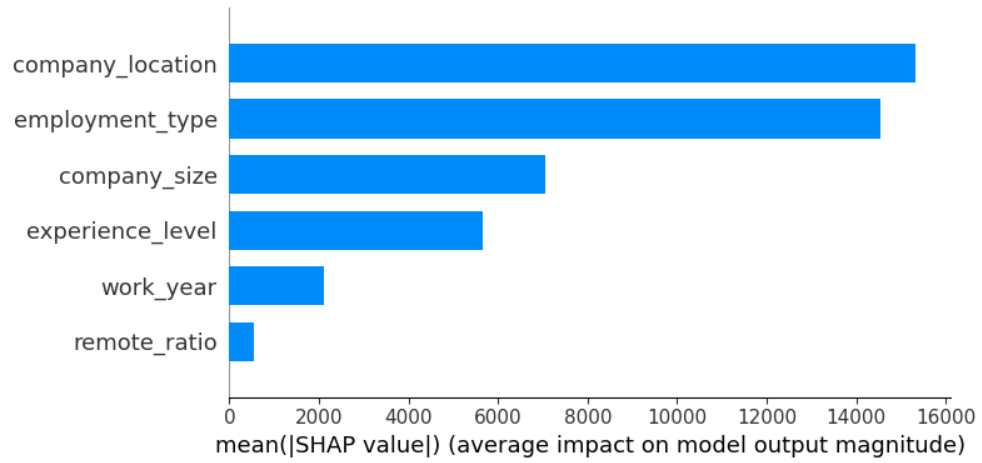


The methods seem to disagree on the importance for most of the features. They agree that experience level is a driving predictive value and that remote ration is relatively unimportant, but the agreement stops there. For the p-value rankings, company size is the most important factor, but for both DnR methods it is the least important factor. For permutation importance it is in the middle. This shows that these methods have lots of variation and using just one to see the feature importance of your model will not give you the full picture. Let’s move on to mean decrease in impurity for the random forest model.

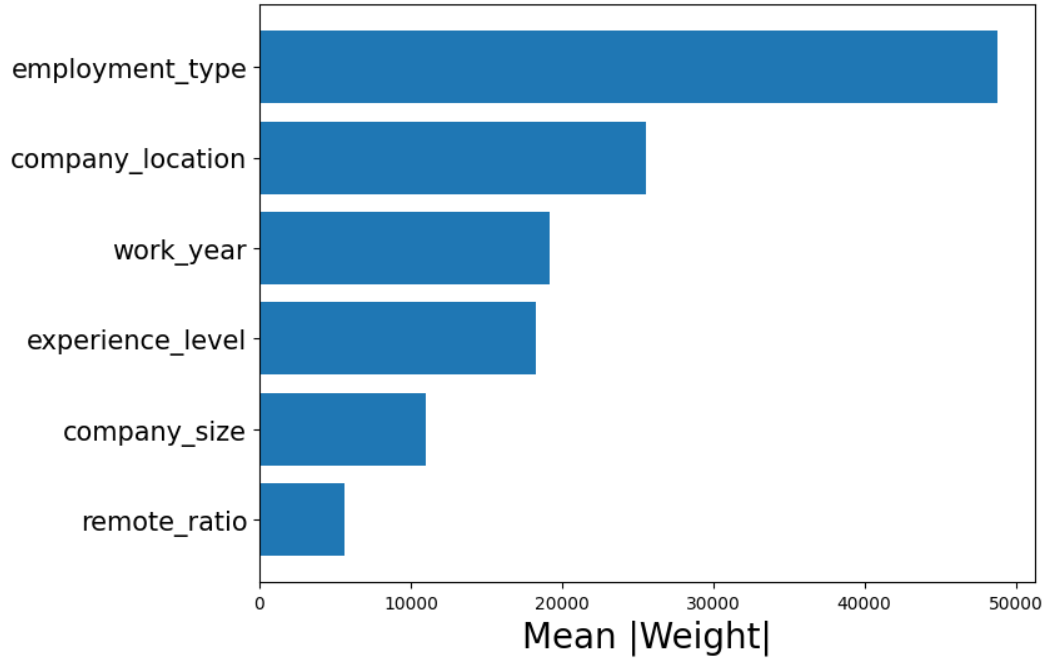


MDI appears to be more inline with the permutation importance rankings. With company location, experience level, and employment type being at the top, and work year and remote ration being relatively unimportant.

Next, let’s look at the SHAP implementation on our random forest model. Note that we have to one hot encode our data, but SHAP will take Shapley values for each subset of the one hot encoding. Thus, before we plot our results, we have to take the empirical average of all of the Shapley values under a certain category (i.e. all of the Shapley values for each class of location, employment type, company size, and so on). Plotting the mean SHAP values we get:



SHAP tells us generally similar results to MDI, with work\_year and remote\_ratio still having the least amount of importance. Interestingly, it devalues experience level, which is a little surprising. Let’s look at LIME and see what it gives us when we aggregate all of the weights together:



Employment type takes the cake when we run LIME on it. This is an example of where each model has a different frontrunner for feature importance. This is very interesting and something to keep note of, especially given that we expect a decent level of correlation between our variables. Overall, we would say that employment type, the experience level, and the company location all have significant effects on the model, despite each model ranking the 3 a little differently.

# 8) Example: Heart Disease

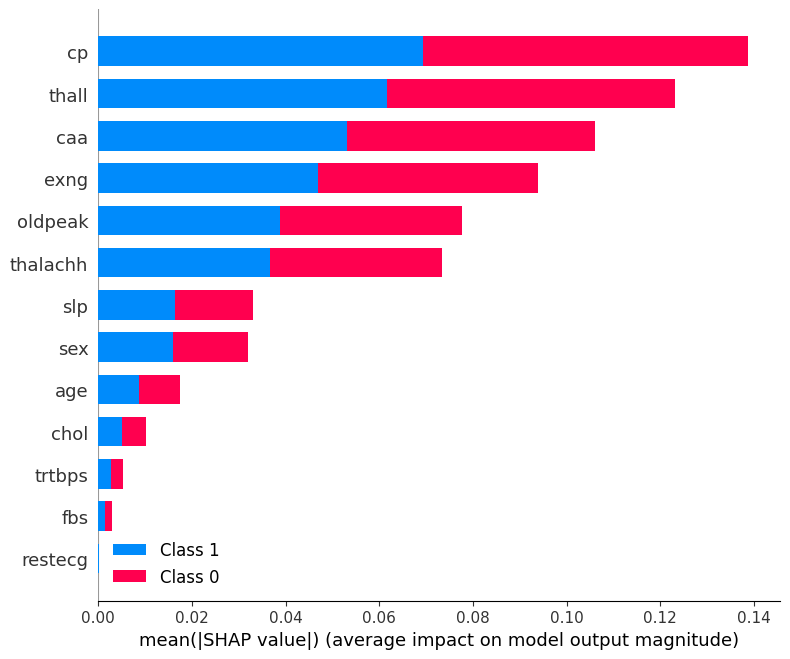
The heart disease dataset consists of a few measurements and some other info about the patient. The target variable of heart disease is binary, and several of the predictor variables are categorical. This means that they need to be one-hot encoded in order to train a logistic regression using this data. Because of this, the resulting p-values of the regression will be for every one-hot encoded variable instead of the non-encoded variable. To make comparisons between this and the other feature importance methods mentioned above, the average ranking of the one-hot encoded variables was taken to produce an approximation for the importance of the variable as a whole.

The data isn’t very highly correlated, so the more basic methods of drop-and-relearn and permutation importance shouldn’t be too far off of the more sophisticated methods. The results of the logistic regression, drop-and-relearn for both logistic regression and random forest, and permutation importance for random forest are as follows.



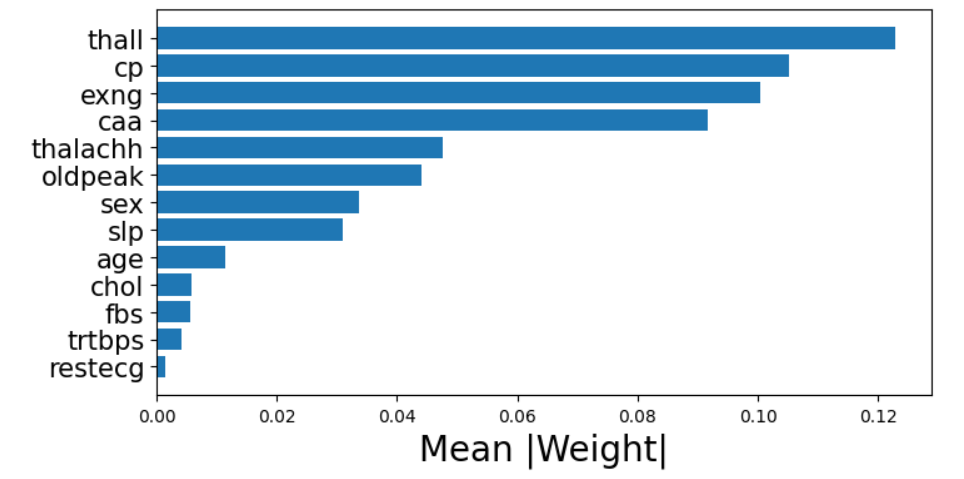
The results are pretty consistent between all of these methods except for a few exceptions, thall is at the bottom of the p-value and DnR feature importance lists, but near the top of the permutation importance list. Inversely, sex is at the bottom for permutation importance, while near the top for the other three. This is interesting, because DnR and permutation importance are very similar methods, so it is surprising that they are so different from each other.

The next method that we will be looking at is SHAP. The results are as follows.



As can be seen above, SHAP more closely follows permutation importance with thall towards the top and sex towards the bottom. However, SHAP places more importance on exon which is more similar to the logistic regression p-value ranking.

The next method to compare is LIME. These results are found in a similar way to SHAP, so they should be relatively similar. The results are below.



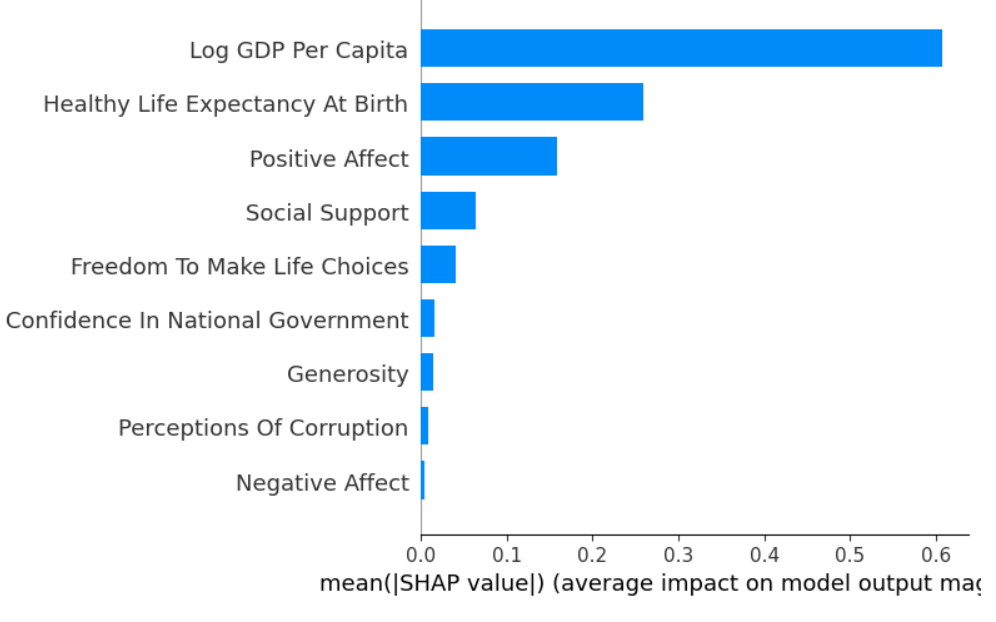
The results for LIME are very similar to SHAP with variables only differing by a single place at most for the rankings. They both seem to suggest that whether a person has the blood disease thalassemia (thall), their chest pain level (cp), whether or not exercising induces angina (exng), and the number of blood vessels colored by fluoroscopy (caa) are the most important factors in predicting heart disease for our model.

# 9) Example: World Happiness Report

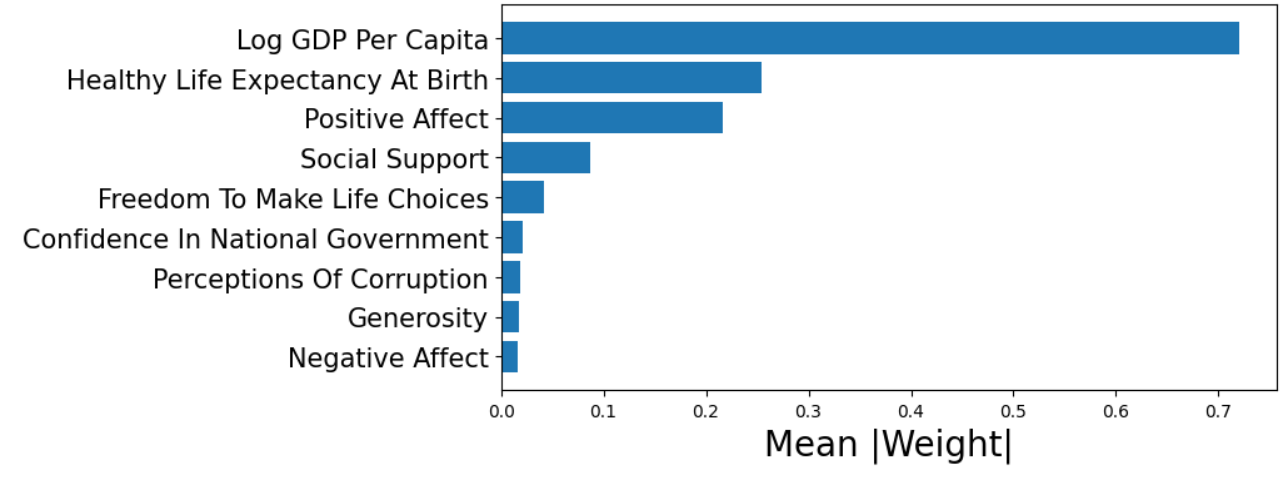
The world happiness report was conducted by the Gallup World Poll from 2005 to 2020. It has various features to try and predict how happy a nation’s population is. Every variable is continuous, so there will be no need to one-hot encode. Also, we will be using a linear regression in place of the logistic regression from earlier. This means that we will be able to directly compare the p-value rankings of the variables without having to make any assumptions. Additionally, the DnR feature importance algorithm will use R2 instead of accuracy to measure model performance. Some of the variables have quite high correlation, so this should make the more basic methods of DnR and permutation importance perform worse compared to the more advanced methods of SHAP, LIME and MDI. The results of the more basic methods of p-value ranking, DnR, and permutation importance are below.



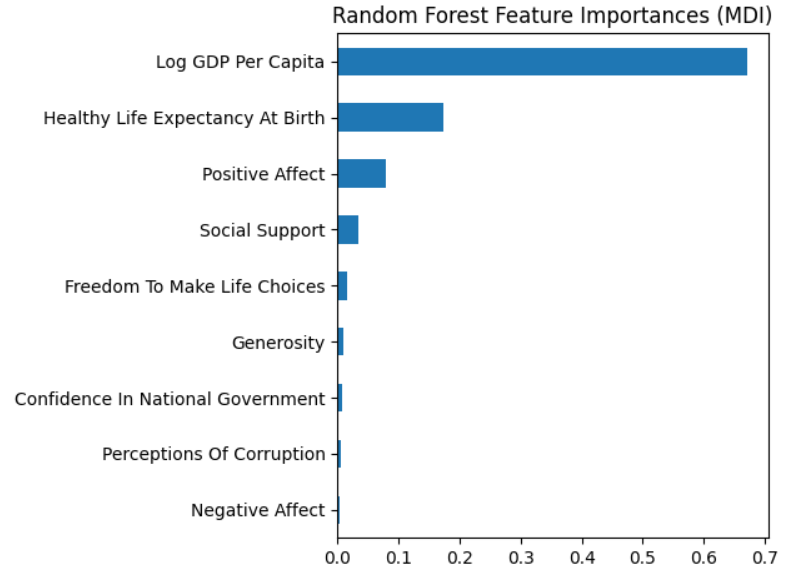
The results for this are very interesting because the p-value ranking and the two DnR rankings are exactly the same, but Permutation importance seems to be somewhat different. All models have GDP per capita at the top, but they differ mostly on positive affect and confidence in national government. This means that GDP per capita is a very strong predictive variable for this model. Let’s see how the rankings for SHAP differ.



The results are exactly the same as permutation importance. As we can see from the SHAP results graph, GDP per capita is by far the strongest predictor. That must be why it is at the top of every feature importance method ranking so far. The variables at the bottom appear to have a very low impact on the model. Next, we’ll take a look at the mean decrease in impurity results. Now let’s compare the LIME results.



Here we can see a very similar result to SHAP. GDP per capita appears to be an even stronger predictor according to LIME. So far, this model appears to be driven by just a few big predictor variables, while the other variables appear to be almost useless. Finally, let's look at the MDI results to finalize our conclusion.



Again, GDP per capita is far and away the strongest predictor variable. From this we can conclude that for this model, the log GDP per capita, healthy life expectancy at birth, positive affect, and social support are the strongest predictive variables for this model. This model is unique in that one variable is so much more powerful than the others. This makes the rest of the variables seem useless by comparison. In the future it might be useful to try the rankings again after a shrinkage method such as lasso or ridge has been applied to the model. Would GDP per capita look more powerful or less without the less powerful predictors in the model?

# 10) Image Feature Importance

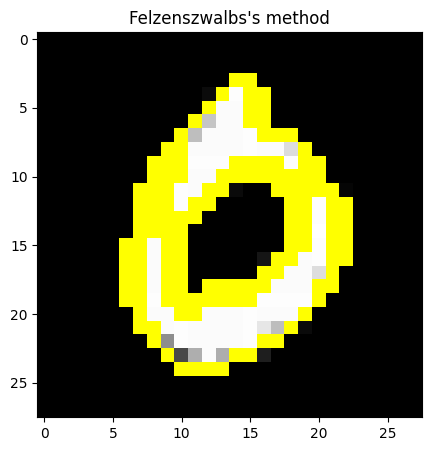
Image feature importance is much more difficult than regular numerical or categorical models. This is because the variables are pixels. They represent nothing by themselves. Pixels only form an image when there are lots of them working together. This means that the previous methods won’t work without significant changes to how they work.

The most basic method for feature importance is drop-and-relearn. This can be applied at pixel level to images, but there isn’t really any important information that can be gathered by doing this. One could perhaps see which pixels are used the most by the images, but again, this is relatively useless information as there are potentially millions of pixels per image. If we instead look at images as a collection of subimages rather than a collection of pixels, we can then use drop-and-relearn on these segments and find what part of the image the model is actually using to classify the image. Similarly, we can use LIME and SHAP to identify which segments of the images are most important.

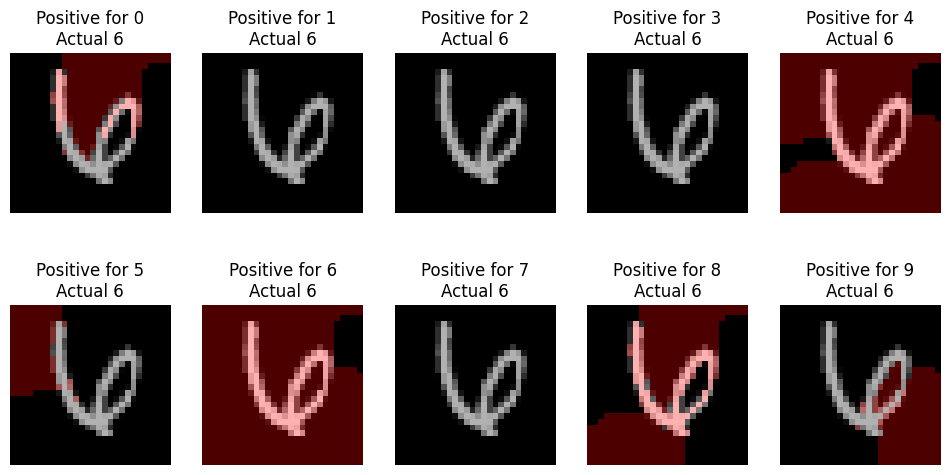
There are a few different types of segmentation for images. For the purpose of this essay, we will only use quickshift and Felzenszwalbs's method. These methods work by looking for clusters of pixels that are separate from the others. They can be tuned to find the right amount of segments to fit your model, so they are very flexible. The segments are then run through the feature importance algorithms and one can look at which segments helped or hurt your model’s prediction. This is particularly useful in determining what part of the image your model was using to misclassify an image. This information can help you tune your model to get better results.

# 11) Example: MNIST Digits

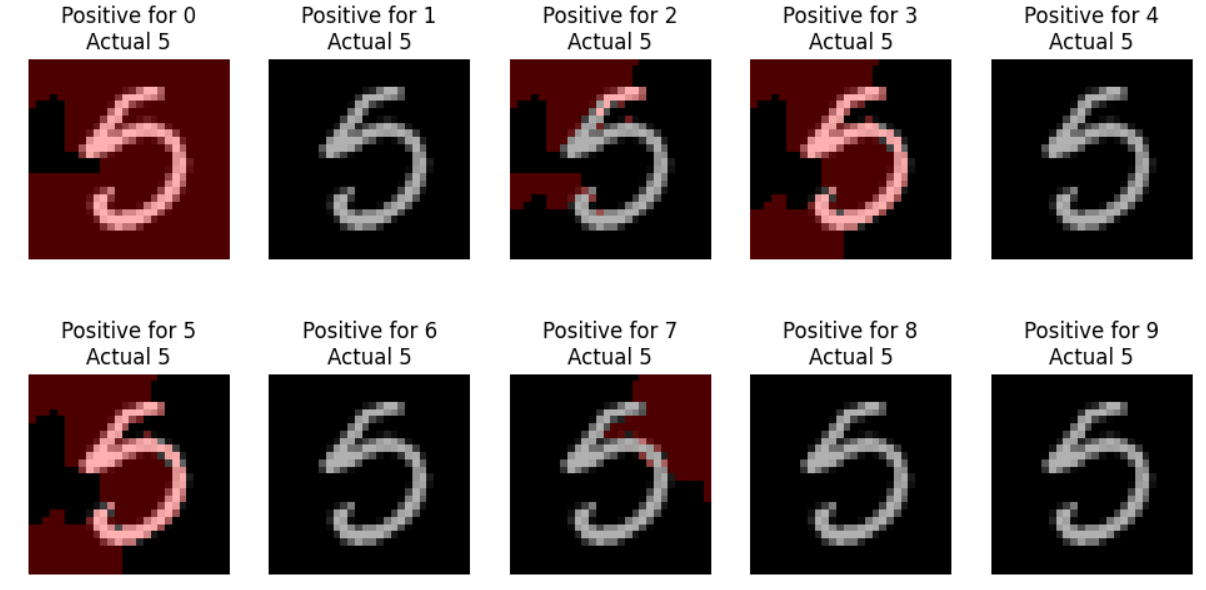
The MNIST (Modified National Institute of Standards and Technology) digits dataset is a collection of labeled numbers. The images are relatively small, as they are only 28x28 pixels. This makes it a computationally light image classifier training dataset. It will also make it easier to see how LIME works to determine the effectiveness of the various segments. First we’ll look at what the Felzenszwalbs's method for segmentation looks like. Below is the number 0 segmented using this method.



As you can see it is segmented into four parts: the upper and lower halves of the 0, the middle, and the outside space. These segments can then be permuted or dropped and can be treated as features for feature importance. Next, let’s look at a LIME feature importance that uses the quickshift segmentation method for when the number 6 is correctly classified.



Here, we can see that the image with the most red is the positive for 6 image. This means that the model is most confident about this guess. The positive for 4 image is also very red. This means that the model thinks there is a relatively high likelihood that this image is of a 4. Where this really comes into play is when one is trying to tune a model to perform classification better. Usually, it is nearly impossible to understand why a model misclassified some object. LIME offers a way to see what the model was looking at when it made its incorrect guess.



Here we see that the model incorrectly guessed this number to be a 0 when it is actually a 5. The model’s next best guesses were for 5 and 3. This can be useful in determining how to tune a model to perform better. More training data can be added to help the model learn that this shape is also a 5 or hyperparameters can be tuned to make the model learn better. LIME offers a way to see inside the blackbox a little to help improve image classification models.

# Conclusion

Feature importance for black box models is a very important step in creating robust black box models. There are several methods that can be used in estimating the predictive power of the various variables in a model. The more basic ones of drop-and-relearn and permute-and-relearn struggle to give an accurate estimate of feature importance, but can be useful to get a basic understanding of the data. The more advanced methods such as SHAP, LIME, and MDI offer much greater insight into the workings of models. It is interesting how permutation importance seems to follow more closely to the more advanced methods than drop-and-relearn does. Both work in relatively similar ways, but drop-and-relearn is far worse. This was a surprising result for us because we were expecting the 3 most basic methods to be similarly bad at estimating feature importance. This means that permutation importance is the best method for feature importance when you want a basic look at the model and don’t have the computation power or time to implement the more advanced methods.

In the future, our work could include using these methods to shrink black box models variables in order to create the most robust model possible. We could also take a look at using these methods for other types of models besides random forests such as neural networks, XGboost, and support vector machines. There is also much more to image feature importance that wasn’t covered in this paper due to it being out of scope, but there are many more types of image segmentation and feature importance algorithms for images. Feature importance is essential in the modern world which is dominated by complex black box models.

**References**

[1]<https://towardsdatascience.com/understanding-feature-importance-and-how-to-implement-it-in-python-ff0287b20285>

[2] <https://umdearborn.edu/news/ais-mysterious-black-box-problem-explained>

[3] <https://scikit-learn.org/stable/modules/permutation_importance.html>

[4] <https://arxiv.org/abs/1604.04173>

[5] <https://arxiv.org/abs/1602.04938>

[6] <https://christophm.github.io/interpretable-ml-book/lime.html#fn50>

[7] <https://github.com/marcotcr/lime>

[8] <https://christophm.github.io/interpretable-ml-book/shapley.html>

[9] Štrumbelj, Erik, and Igor Kononenko. “Explaining prediction models and individual predictions with feature contributions.” Knowledge and information systems 41.3 (2014): 647-665.

[10] <https://shap.readthedocs.io/en/latest/index.html>