One of the best challenges in Machine Learning tends to let the model speak themself. It not also is important to develop a strong solution with great predicting power, but also in a lot of business applications is interesting to know how the model provides these results: which variables are engaged the most, the presence of correlations, the possible causation relationships and so on.

These needs made the Tree-based model a good weapon in this field. They are scalable and permits to compute variable explanation very easy. Every software provides this option and each of us has at least once tried to compute the variable importance report with Random Forest or similar. With Neural Net this kind of benefit is considered taboo. Neural Network is often seen as a black box, from which it is very difficult to extract useful information for another purpose like feature explanations.

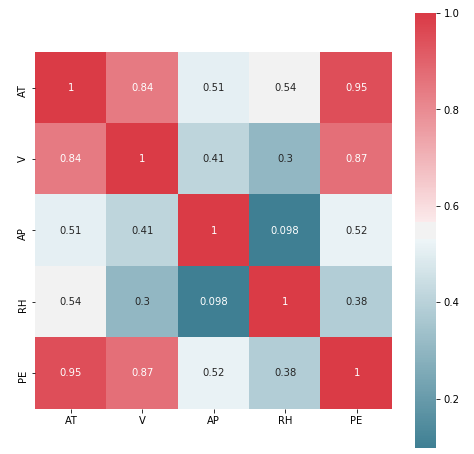
In this post, I try to provide an elegant and clever solution, that with few lines of codes, permits you to squeeze your Machine Learning Model and extract as much information as possible, in order to **provide feature importance, individuate the significant correlations and try to explain causation**.

**THE DATASET**

Given a real dataset, we try to investigate which factors influence the final prediction performances. To achieve this aim we took data from UCI Machine Learning Repository. The privileged dataset was the [Combined Cycle Power Plant](https://archive.ics.uci.edu/ml/datasets/Combined+Cycle+Power+Plant) Dataset, where were collected *6 years of data when the power plant was set to work with full load. Features consist of hourly average variables: Ambient Temperature (AT), Ambient Pressure (AP), Relative Humidity (RH) and Exhaust Vacuum (V) to predict the net hourly electrical energy output (PE) of the plant.*

The variables engaged are related by Pearson correlation linkages as shown in the matrix below.

Image for post



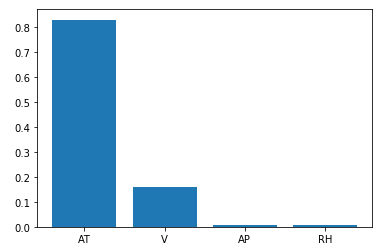
Correlation Matrix

**GRADIENT BOOSTING FEATURE IMPORTANCE**

We start building a simple Tree-based model in order to provide energy output *(PE)* predictions and compute the standard feature importance estimations. This final step permits us to say more about the variable relationships than a standard correlation index. These numbers summarized the reduction in impurity index over all trees when a particular feature is pointed during internal space partition (in training phase). Sklearn applies normalization in order to provide output summable to one. It is also a free result, obtainable indirectly after training.

gb = GradientBoostingRegressor(n\_estimators=100)  
gb.fit(X\_train, y\_train.values.ravel())plt.bar(range(X\_train.shape[1]), gb.feature\_importances\_)  
plt.xticks(range(X\_train.shape[1]), ['AT','V','AP','RH'])

Image for post



GradientBoosting Features Importance

This result is easily interpretable and seems to replicate the initial assumption made computing correlations with our target variable (last row of correlation matrix): higher the value, higher is the impact of this particular feature predicting our target.

Despite the goods results we achieved with our Gradient Boosting we don’t want to completely depend by this kind of approach… We want to generalize the process of computing feature importance, let us free to develop another kind of Machine Learning model with the same flexibility and explainability power; making also a step further: provide evidence of the presence of significant casualty relationship among variables.

**PERMUTATION IMPORTANCE**

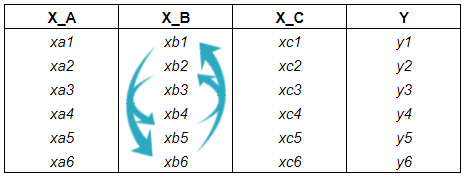
The models identified for our experiment are doubtless Neural Networks for their reputation to be a black box algorithm. In order to demystify this stereotype, we’ll focus on Permutation Importance. Its easy implementation, combined with its tangible understanding and adaptability, making it a consistent candidate to answer the question: *What features have the biggest impact on predictions?*

Permutation importance **is calculated after a model has been fitted.** So we have only to squeeze it and get what we want. This method works on a simple principle: *If I randomly shuffle a single feature in the data, leaving the target and all others in place, how would that affect the final prediction performances*?

From this random reordering of variables I expect to obtain:

* Less accurate predictions, since the resulting data no longer corresponds to anything observed in the real world;
* Worst performances, from the shuffle of the most important variables. This is because we are corrupting the natural structure of data. If we, with our shuffle, break a strong relationship we’ll compromise what our model has learned during training, resulting in higher errors (**high error = high importance**).

Image for post



Permutation Importance at work

Practically speaking this is what’s happened in our real scenario…

We chose an adequate Neural Net structure to model the *hourly electrical energy output*(*EP*). Remember to scale also the target variable in a lower range: I classically subtracted mean and divided for standard deviation, this helps the train.

inp = Input(shape=(scaled\_train.shape[1],))  
x = Dense(128, activation='relu')(inp)  
x = Dense(32, activation='relu')(x)  
out = Dense(1)(x)model = Model(inp, out)  
model.compile(optimizer='adam', loss='mse')model.fit(scaled\_train, (y\_train - y\_train.mean())/y\_train.std() , epochs=100, batch\_size=128 ,verbose=2)

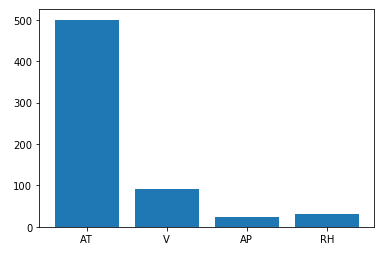
At the prediction stage, the Gradient Boostingand the Neural Net achieve the same performance in terms of Mean Absolute Error, respectively 2.92 and 2.90 (remember to reverse predictions).

At this point, we ended with training and let’s start to randomly sample.

We compute the shuffle of every feature on validation data (4 times in total = 4 explicative variables) and provide error estimations at each step; remember to return the data to the original order at every step. Then I plot the MAE we achieved at every shuffle stage as percentage variation from the original MAE (around 2,90)

plt.bar(range(X\_train.shape[1]), (final\_score - MAE)/MAE\*100)  
plt.xticks(range(X\_train.shape[1]), ['AT','V','AP','RH'])

Image for post



Permutation Importance as percentage variation of MAE

The graph above replicates the RF feature importance report and confirms our initial assumption: the *Ambient Temperature (AT)* is the most important and correlated feature to predict *electrical energy output (PE)*. Despite *Exhaust Vacuum (V)*and*AT*showed a similar and high correlation relationship with *PE*(respectively 0.87 and 0.95), they have a different impact at the prediction stage. This phenomenon is a soft example of how not always a high correlation (in Pearson term) is synonymous of high explainability power.

**CAUSATION RELATIONSHIPS**

Prove correlation, in order to avoid [spurious relationships](https://www.tylervigen.com/spurious-correlations), is always an insidious operation. At the same time, it is difficult to show evidence of casualty behaviors. In literature, there are a lot of methods to prove causality. One of the most important is the [Granger Causality Test](https://en.wikipedia.org/wiki/Granger_causality). This technique is widely applied in time series domain for determining whether one-time series is useful in forecasting another: i.e. demonstrate (according to an F-test on lagged values) that it adds explanatory power to the regression.

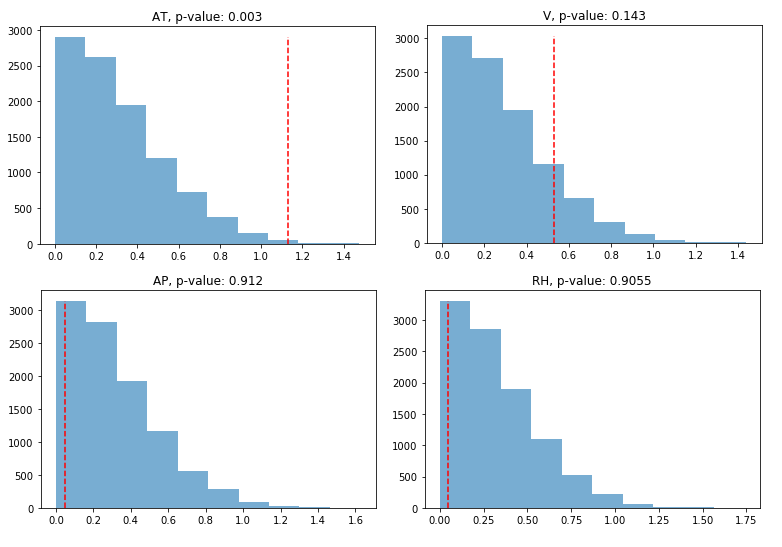
Indirectly this is what we have already done computing Permutation Importance. Shuffling every variable and looking for performance variations, we are proving how much explicative power has this feature to predict the desired target.

In order to prove causation, what we have to do now is to demonstrate that the data shuffle provides significative evidence in performance variation. We operate on the final predictions, achieved without and with shuffle, and verify if there is a difference in mean among the two prediction population. It means that the mean predictions with shuffle might as well be observed by any random subgroup of predictions. So that’s exactly what we’ll do for every feature: we’ll merge prediction with and without permutation, we’ll randomly sample a group of predictions and calculate the difference between their mean value and the mean values of the prediction without shuffle.

np.random.seed(33)  
id\_ = 0 #feature index  
merge\_pred = np.hstack([shuff\_pred[id\_], real\_pred])  
observed\_diff = abs(shuff\_pred[id\_].mean() - merge\_pred.mean())  
extreme\_values = []  
sample\_d = []for \_ in range(10000):  
 sample\_mean = np.random.choice(merge\_pred,  
 size=shuff\_pred[id\_].shape[0]).mean()  
 sample\_diff = abs(sample\_mean - merge\_pred.mean())  
 sample\_d.append(sample\_diff)  
 extreme\_values.append(sample\_diff >= observed\_diff)  
   
np.sum(extreme\_values)/10000 #p-value

In order to have all under control, it’s a good choice to visualize the results of our simulations. We plot the distribution of the simulated mean differences (blue bar) and mark the real observed difference (red line). We can see that for *AT*there is evidence for a difference in mean with the prediction made without shuffle (low p-value: below 0.1). The other variables don’t bring a significant improvement in the mean.

Image for post



Simulation distributions and relative p-values

Correlation doesn’t always imply causation! With this in mind, **we proved causation in terms of the ability of a selected feature to add explicative power**. We’ve recreated, with our knowledge of statistician and programmer, a way to prove this concept making use of our previous findings made with permutation importance, adding information about the relationships of our variables.

**SUMMARY**

In this post, I’ve introduced Permutation Importance, an easy and clever technique to compute feature importance. It’s useful with every kind of model (I use Neural Net only as a personal choice) and in every problem (an analog procedure is applicable **in a classification task: remember to choose an adequate loss measure** when computing permutation importance, like cross-entropy, avoiding the ambiguous accuracy). We’ve also used the permutations to present a method that proves casualty among variables hacking the p-value!