Machine Learning Explainability Introduction via eli5

Extract insight from your Machine Learning model

Cornellius Yudha Wijaya Jun 29, 2020 · 6 min read ★

Photo by h heyerlein on Unsplash

explain why your model is working and what kind of insight your model gives. By insight, I am not referring to the model accuracy or any metric but the machine learning model itself. This is what we called Machine Learning Explainability.

The most straightforward example of Machine Learning Explainability is the Linear Regression Model with the Ordinary Least Square estimation method. Let me give you an example by using a dataset. Note that I violate some of the Ordinary Least Square assumptions, but my point is not about creating the best model; I just want to have a model that could give an insight.

```
#Importing the necessary package
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
from statsmodels.api import OLS

#Load the dataset and preparing the data
mpg = sns.load_dataset('mpg')
mpg.drop(['origin', 'name'], axis =1, inplace = True)
mpg.dropna(inplace = True)

#Ordinary Least Square Linear Regression model Training
sm_lm = OLS(mpg['mpg'], sm.add_constant(mpg.drop('mpg', axis = 1)))
result = sm_lm.fit()

result.summary()
```

OLS Regression Results

Dep. Variable:	mpg	**R-squared:**	0.809
Model:	OLS	**Adj. R-squared:**	0.806
Method:	Least Squares	**F-statistic:**	272.2
Date:	Sun, 28 Jun 2020	**Prob (F-statistic):**	3.79e-135
Time:	15:47:28	**Log-Likelihood:**	-1036.5
No. Observations:	392	**AIC:**	2087.
Df Residuals:	385	**BIC:**	2115.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]

displacement	0.0077	0.007	1.044	0.297	-0.007	0.022
horsepower	-0.0004	0.014	-0.028	0.977	-0.028	0.027
weight	-0.0068	0.001	-10.141	0.000	-0.008	-0.005
acceleration	0.0853	0.102	0.836	0.404	-0.115	0.286
model_year	0.7534	0.053	14.318	0.000	0.650	0.857

Omnibus:	37.865	**Durbin-Watson:**	1.232
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	60.248
Skew:	0.630	**Prob(JB):**	8.26e-14
Kurtosis:	4.449	**Cond. No.**	8.53e+04

I would not explain the model in detail, but the Linear model assumed there is **linearity** between the **Independent variables** (Features to predict) and the **Dependent variable** (what you want to predict). The equation is stated below.

$$y = c + m_1 x_1 + m_2 x_2 + \ldots + m_n x_n$$

What is important here is that every independent variable(x) is multiplied by the coefficient(m). It means that the coefficient tells the relationship between the independent variable with the dependent variable. From the result above, we can see that the coefficient (coef) of the 'model_year' variable is 0.7534. It means that every increase of 'model_year' by one, the Dependent variable 'mpg' value would increase by 0.7534.

The Linear Regression Model with their coefficient is an example of Machine Learning explainability. The model itself is used to explain what happens with our data, and extraction of insight is possible. However, not all model is viable to do this.

Machine Learning Explainability of Black-Box Model

Why? A forest consists of a large number of deep trees, where each tree is trained on bagged data using a random selection of features. To gaining a full understanding by examining each tree would close to impossible.

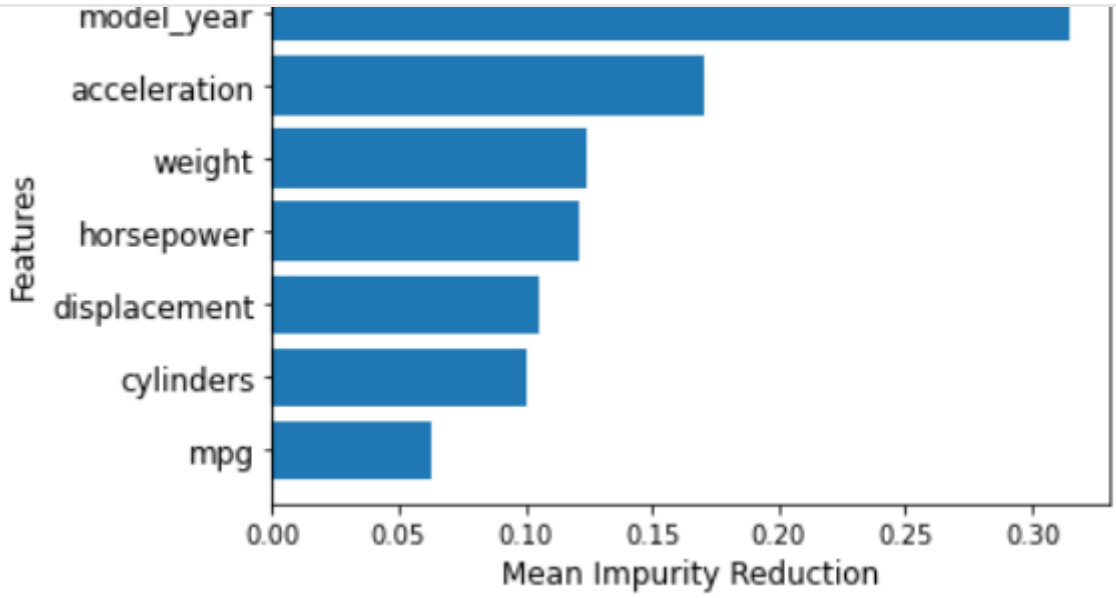
For example, the famous XGBoost Classifier from the xgboost package is nearly a black-box model that utilises a random forest process. This model is considered as a black box model because we did not know what happens in the model learning process. Let's try it using the same dataset as an example.

```
#Preparing the model and the dataset

from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
mpg = sns.load_dataset('mpg')
mpg.drop('name', axis =1 , inplace = True)

#Data splitting for xgboost
X_train, X_test, y_train, y_test =
train_test_split(mpg.drop('origin', axis = 1), mpg['origin'],
test_size = 0.2, random_state = 121)

#Model Training
xgb_clf = XGBClassifier()
xgb_clf.fit(X_train, y_train)
```

Just like that, we have the model, but did we get any insight from the data? Or could we know the relationship between the dependent to the independent?.

Well, you could argue that the classifier owns a feature importance method which is a tree-model specific to measure how important the feature. To be precise, it measures the feature contribution to the mean impurity reduction of the model.

```
tree_feature =  pd.Series(xgb_clf.feature_importances_,
X_train.columns).sort_values(ascending = True)

plt.barh(X_train.columns, tree_feature)
plt.xlabel('Mean Impurity Reduction', fontsize = 12)
plt.ylabel('Features', fontsize = 12)
plt.yticks(fontsize = 12)
plt.title('Feature Importances', fontsize = 20)
```



To a certain extent, this is a Machine Learning explainability example. Although, you need to remember that xgboost relies on the bootstrapping process for creating the model. Which means, how important the feature is could happen because of the randomised process. Moreover, the contribution only tells how high the feature could reduce the overall impurity (Overall is the mean from all the produced trees).

Permutation Importance via eli5

There is another way to getting an insight from the tree-based model by permuting (changing the position) values of each feature one by one and checking how it changes the model performance. This is what we called the **Permutation Importance** method. We could try applying this method to our xgboost classifier using the *eli5* package. First, we need to install the package by using the following code.

```
#installing eli5

pip install eli5
#or
conda install -c conda-forge eli5
```

```
#Importing the module
from eli5 import show_weights
from eli5.sklearn import PermutationImportance

#Permutation Importance
perm = PermutationImportance(xgb_clf, scoring = 'accuracy'
,random_state=101).fit(X_test, y_test)
show_weights(perm, feature_names = list(X_test.columns))
```

Weight	Feature
0.3797 ± 0.0577	displacement
0.0785 ± 0.0405	weight
0.0329 ± 0.0441	horsepower
0.0127 ± 0.0160	cylinders
0.0101 ± 0.0372	acceleration
-0.0051 ± 0.0124	model_year
-0.0152 ± 0.0405	mpg

The idea behind permutation importance is how the scoring (accuracy, precision, recall, etc.) shift with the feature existence or no. In the above result, we can see that displacement has the highest score with 0.3797. It means, when we permute the displacement feature, it will change the accuracy of the model as big as 0.3797. The value after the plus-minus sign is the uncertainty value. The permutation Importance method is inherently a random process; that is why we have the uncertainty value.

The higher the position, the more critical the features are affecting the scoring. Some feature in the bottom place is showing a minus value, which is interesting because it means that the feature increasing the scoring when we permute the feature. This happens because by chance the feature permutation actually improves the score.

We have known about both approaches by measuring the impurity reduction and permutation importance. They are useful but crude and static in the sense that they give little insight into understanding individual decisions on actual data. This is why we would use the eli5 weight feature importance calculation based on the tree decision path.

```
show_weights(xgb_clf, importance_type = 'gain')
```

Weight	Feature
0.3008	displacement
0.1730	cylinders
0.1370	horsepower
0.1336	mpg
0.0934	weight
0.0869	acceleration
0.0752	model_year

We can see that the 'displacement' feature is the most important feature, but we have not yet understood how we get the weight. For that reason, let's see how the classifier tries to predict for individual data.

```
#Taking an example of test data

from eli5 import show_prediction
show_prediction(xgb_clf, X_test.iloc[1], show_feature_values=True)
```

y=europe (probability **0.227**, score **0.028**) top features

Contribution	Feature	Value
+0.347	acceleration	16.000
+0.280	mpg	22.000
+0.104	displacement	122.000
+0.052	model_year	72.000
-0.026	weight	2395.000
-0.150	horsepower	86.000
-0.580	<BIAS>	1.000

y=japan (probability **0.024**, score **-2.206**) top features

Contribution	Feature	Value
+0.478	weight	2395.000
-0.003	acceleration	16.000
-0.052	cylinders	4.000
-0.299	mpg	22.000
-0.392	<BIAS>	1.000
-0.945	horsepower	86.000
-0.993	displacement	122.000

y=usa (probability **0.749**, score **1.222**) top features

Contribution	Feature	Value
+0.864	<BIAS>	1.000
+0.631	horsepower	86.000
+0.248	weight	2395.000
+0.111	cylinders	4.000
+0.019	acceleration	16.000
-0.068	model_year	72.000
-0.122	mpg	22.000
-0.461	displacement	122.000

The table above shows us how our classifier predicts our data based on the data given. Every class have their probability and how each feature contributes to the probability and the score (Score calculation is based on the decision path). The classifier also introduce `<BIAS>` feature which is expected average score output by the model, based on the distribution of the training set. If you want to know more about it, you could check it out here.

the percentage of each feature contributed to the final prediction across all trees (If you sum the weight it would be close to 1).

With this package, we capable to measure how important the feature is not just based on the feature performance scoring but how each feature itself contribute to the decision process.

Conclusion

With eli5, we capable of turning the black-box classifier into a more interpretable model. There are still many methods we could use for the Machine Learning Explainability purposes which you could check in the eli5 homepage.

If you enjoy my content and want to get more in-depth knowledge regarding data or just daily life as a Data Scientist, please consider subscribing to my newsletter here.