Explainable AI - Lecture 7

LOFO, SAGE and XAI metrics

norwegian open ai lab NTNU

Feature importance

Assume a model f assigning credit score y based on a set of features x_d

I give you a ranking of the input features, the first one being most and the last one being least important.

This feature importance ranking could for example be:

[Income, debt, experience, age, sex]

This is a **model explanation** in terms of **feature importances**. An XAI method that produces such a list does **feature importance attribution**.

Feature importance

Assume a model f assigning credit score y based on a set of features x_d

I give you a ranking of the input features, the first one being most and the last one being least important.

This feature importance ranking could for example be:

[Income, debt, experience, age, sex]

This is a model explanation in terms of feature importances.

What does is mean that a feature is important?

Don't try to give the "correct answer", just give your intuition.

A feature is important means that...

Let´s go through the methods we have studied so far.

Feature importance according to...

Let´s go through the methods we have studied so far.

Counterfactuals:

Local surrogate models (LIME):

SHAP:

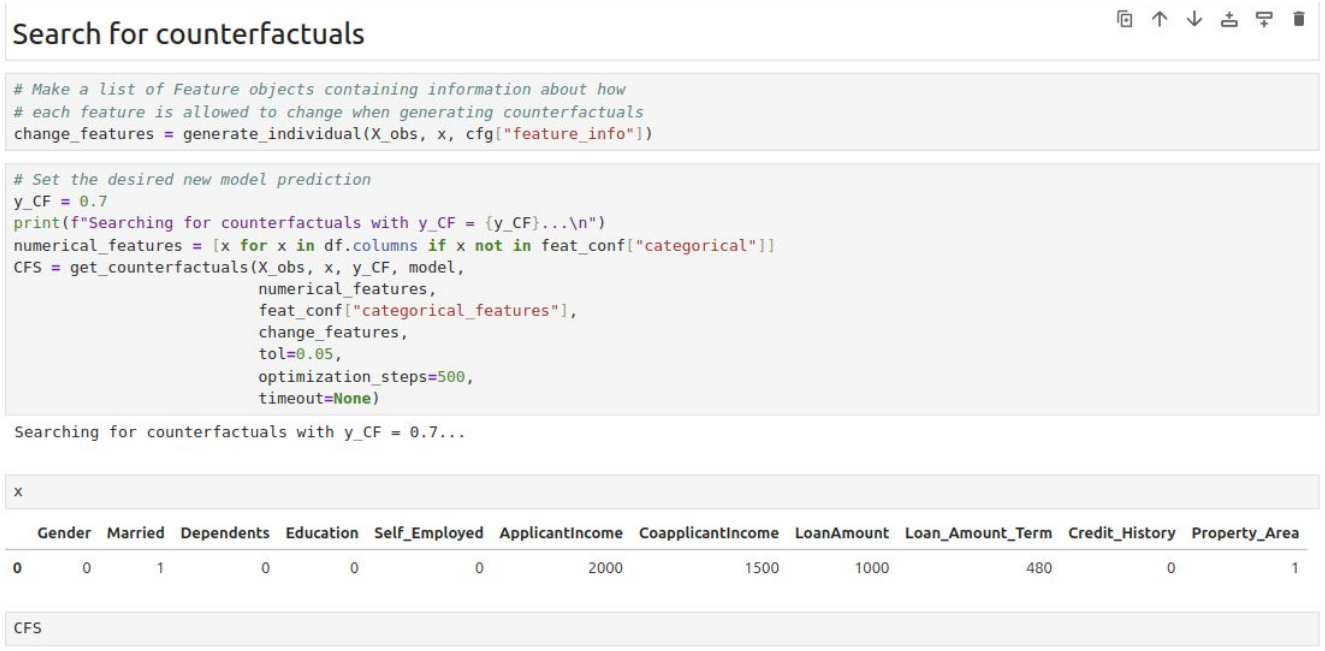
PDP:

ALE:

Feature importance according to...

Counterfactuals tell us...

The importance is...?



Search for counterfactuals

```
# Make a list of Feature objects containing information about how
# each feature is allowed to change when generating counterfactuals
change_features = generate_individual(X_obs, x, cfg["feature_info"])
```

```
# Set the desired new model prediction
y_CF = 0.7
print(f"Searching for counterfactuals with y_CF = {y_CF}...\n")
numerical_features = [x for x in df.columns if x not in feat_conf["categorical"]]
CFS = get_counterfactuals(X_obs, x, y_CF, model,
                          numerical_features,
                          feat_conf["categorical_features"],
                          change_features,
                          tol=0.05,
                          optimization_steps=500,
                          timeout=None)
```

```
Searching for counterfactuals with y_CF = 0.7...
```

```
x
```

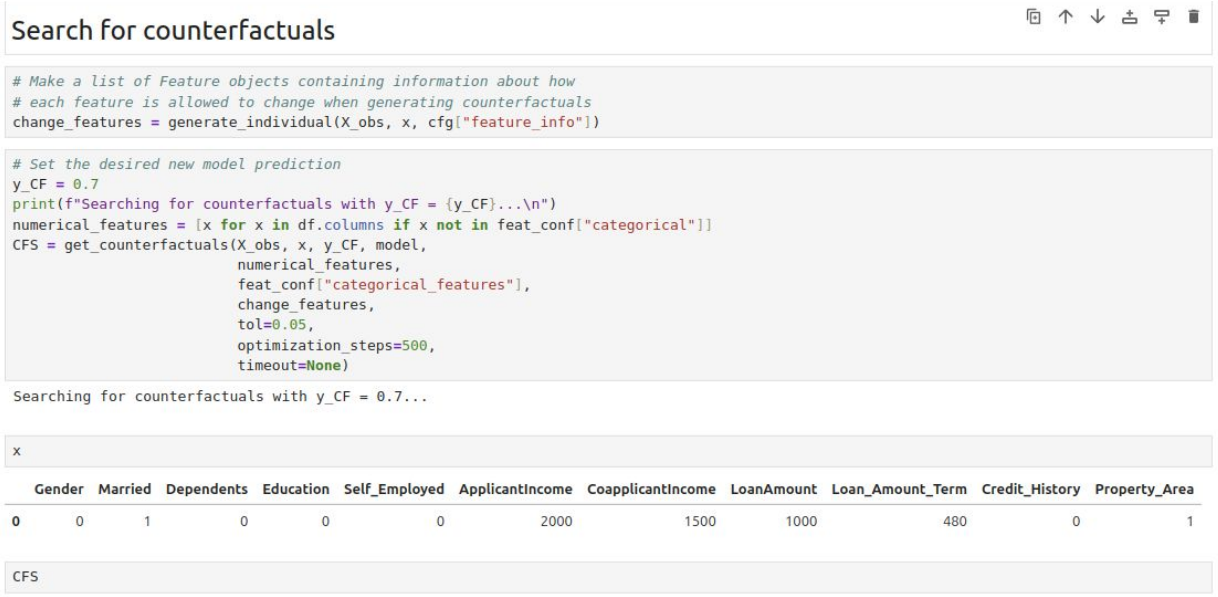
	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
0	0	1	0	0	0	2000	1500	1000	480	0	1

```
CFS
```

Feature importance according to...

Counterfactuals tell us how much a feature must change to alter the model's decision.

The importance is inversely proportional to the required change. If the prediction changes much for a small feature change, the feature is important.



Search for counterfactuals

```
# Make a list of Feature objects containing information about how
# each feature is allowed to change when generating counterfactuals
change_features = generate_individual(X_obs, x, cfg["feature_info"])
```

```
# Set the desired new model prediction
y_CF = 0.7
print(f"Searching for counterfactuals with y_CF = {y_CF}...\n")
numerical_features = [x for x in df.columns if x not in feat_conf["categorical"]]
CFS = get_counterfactuals(X_obs, x, y_CF, model,
                          numerical_features,
                          feat_conf["categorical_features"],
                          change_features,
                          tol=0.05,
                          optimization_steps=500,
                          timeout=None)
```

```
Searching for counterfactuals with y_CF = 0.7...
```

```
x
```

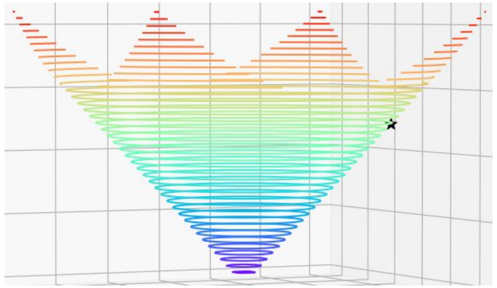
	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
0	0	1	0	0	0	2000	1500	1000	480	0	1

```
CFS
```

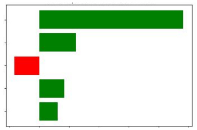
Feature importance according to...

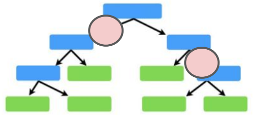
Local surrogate models...

The local surrogate weight represents...?



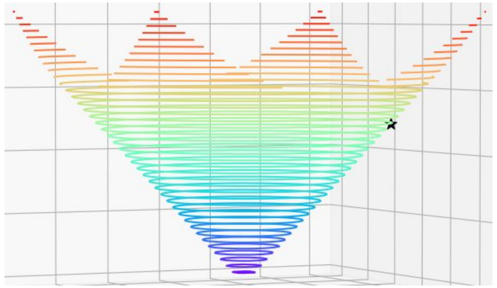
$$y = \beta_0 + \beta_1 x$$



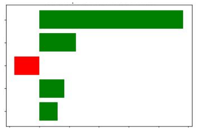


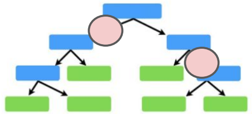
Feature importance according to...

Local surrogate models approximate the model in a neighborhood. The local surrogate weight represents how much the prediction changes when perturbing the corresponding feature locally.



$$y = \beta_0 + \beta_1 x$$

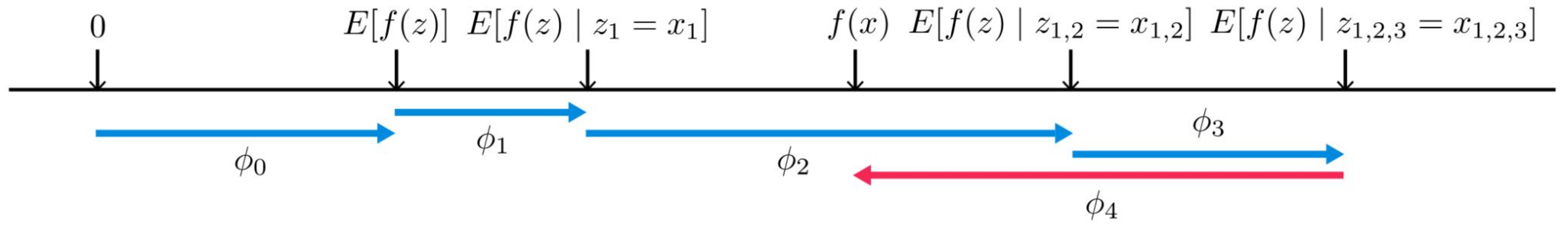




Feature importance according to...

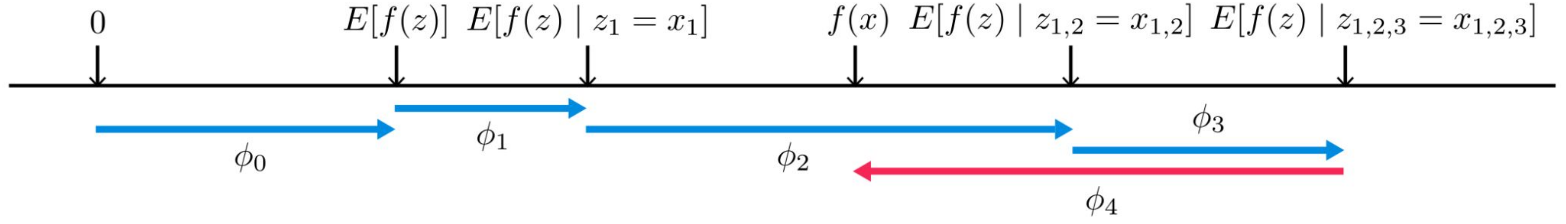
*A feature's **SHAP** value represents...?*

A high SHAP value indicates...?



Feature importance according to...

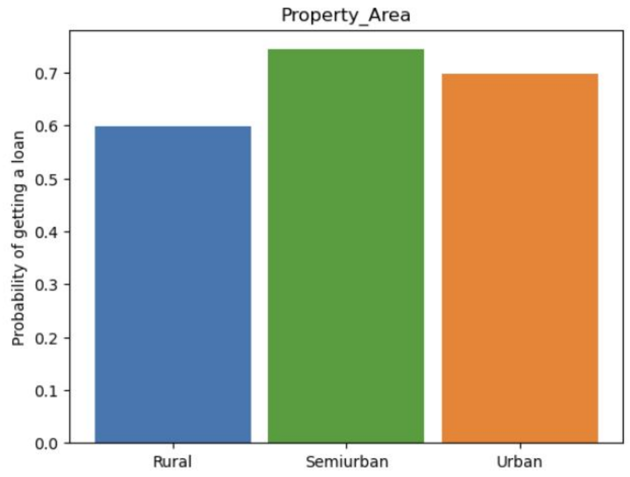
A feature's **SHAP** value represents the feature's contribution to the model's prediction wrt the baseline. A high SHAP value indicates a large contribution to the model prediction.



Feature importance according to...

*A feature's **PDP** shows...?*

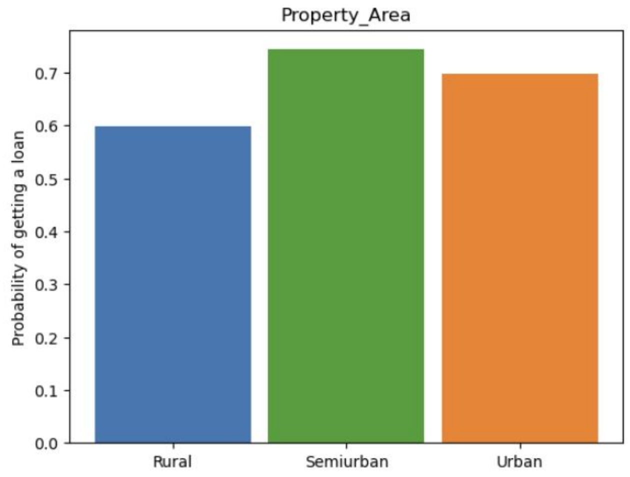
The partial dependence value indicates...?



$$PD_S(\mathbf{x}_S) = E_{X_C}\left[f(\mathbf{x}_S, X_C)\right] \approx \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_S, \mathbf{x}_C^{(i)})$$

Feature importance according to...

A feature's **PDP** shows the average impact of the feature on the model prediction, when sampling the other feature values freely. The partial dependence value indicates how much the average model prediction changes as a single feature varies.

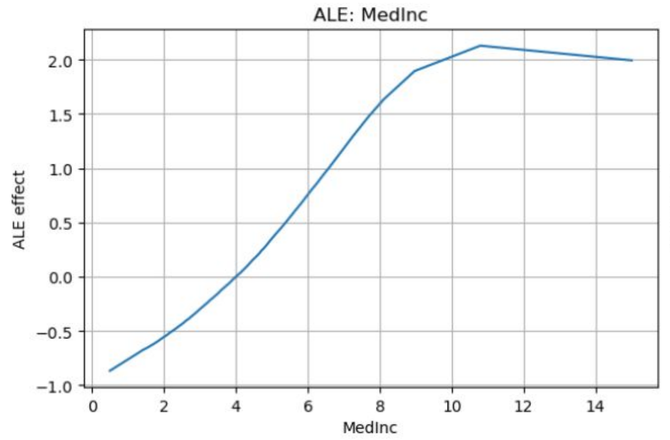


$$PD_S(\mathbf{x}_S) = E_{X_C}\left[f(\mathbf{x}_S, X_C)\right] \approx \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_S, \mathbf{x}_C^{(i)})$$

Feature importance according to...

A feature's ***ALE*** *curve indicates...?*

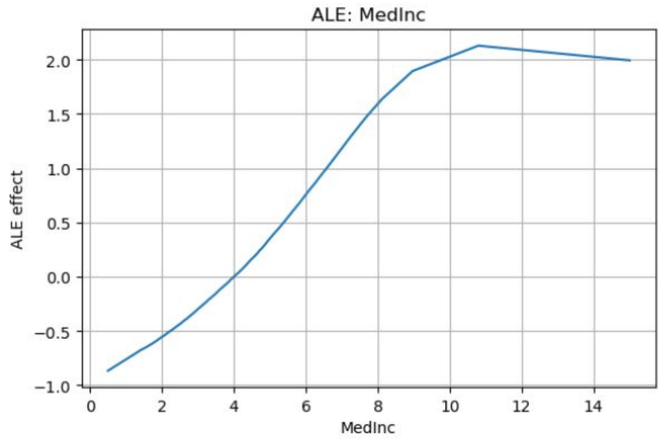
The accumulated local effect shows...?



$$\mathrm{LE}_S(x) = \sum_{i:x_S^{(i)} \in N_S(k)} \left[f(x_S = z_{k,S}, \mathbf{x}_{\setminus j}^{(i)}) - f(x_S = z_{k-1,S}, \mathbf{x}_{\setminus j}^{(i)}) \right]$$

Feature importance according to...

A feature's **ALE** curve indicates how much the model prediction changes with that feature in its empirical [remaining features' values are conditional] domain. The accumulated local effect shows the effect of the feature's value on the average model prediction.



$$\mathrm{LE}_S(x) = \sum_{i:x_S^{(i)} \in N_S(k)} \left[f(x_S = z_{k,S}, \mathbf{x}_{\setminus j}^{(i)}) - f(x_S = z_{k-1,S}, \mathbf{x}_{\setminus j}^{(i)}) \right]$$

Feature importance according to...

Counterfactuals tell us how much a feature must change to alter the model's decision. The importance is inversely proportional to the required change. If the prediction changes much for a small change, the feature is important.

Local surrogate models approximate the model in a neighborhood. The local surrogate weight represents how much the prediction changes when perturbing the corresponding feature locally.

A feature's **SHAP** value represents the feature's average marginal contribution to the model's prediction across all coalitions of features. A high SHAP value indicates a large contribution to the model prediction.

A feature's **PDP** shows the average impact of the feature on the model prediction, when sampling the other feature values freely. The partial dependence value indicates how much the average model prediction changes as a single feature varies.

A feature's **ALE** curve indicates how much the model prediction changes with that feature in its empirical [remaining features' values are conditional] domain. The accumulate local effect shows the model prediction's sensitivity to a feature accumulated across its range.

Feature importance

Do any of these tell us how much **predictive power** a feature provides to the model?

Feature importance

Do any of these tell us how much **predictive power** a feature provides to the model?

Would you use any of these methods for feature selection? Which one?

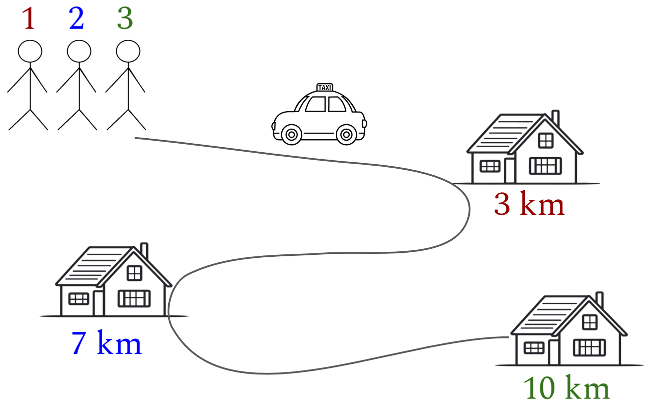
Warmup: LOFO

Which feature is most useful to the model?

Instead of perturbing features of a trained model and observe how the predictions change, we can take a step back and ask how the different features contribute to the model's predictive performance.

Recall the taxi example.

In a *feature selection* scenario, which player/feature is needed to gain the maximum payoff?

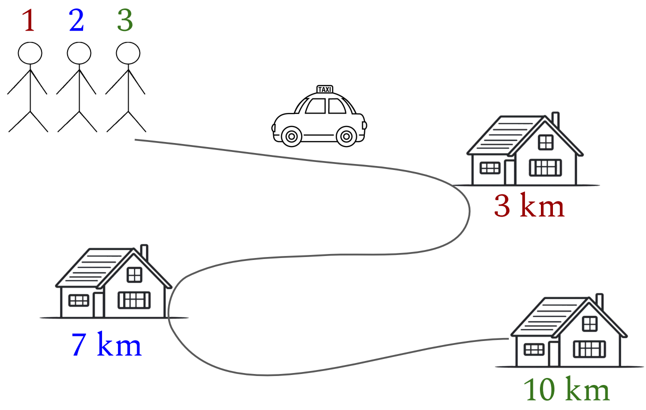


Which feature is most useful to the model?

Instead of perturbing features of a trained model and observe how the predictions change, we can take a step back and ask how the different features contribute to the model's predictive performance.

Recall the taxi example.

In a *feature selection* scenario, only player/feature 3 is needed to gain the maximum payoff.



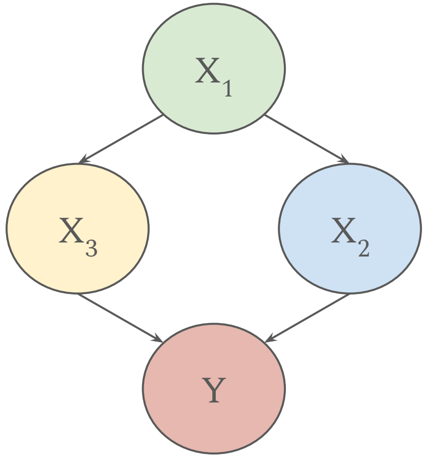
Which feature is most useful to the model?

Instead of perturbing features of a trained model and observe how the predictions change, we can take a step back and ask how the different features contribute to the model's predictive performance.

A similar situation occurs in the case of *redundant features*.

A redundant feature is one that provides the same or similar information as another feature. Consider a data generating process like on the right:

Again, from a *feature selection* perspective, which features are important?

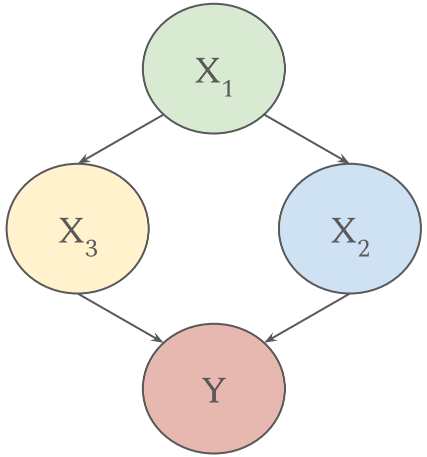

$$\begin{tikzcd} & X_1 \arrow[dl] \arrow[dr] & \\ X_3 \arrow[dr] & & X_2 \arrow[dl] \\ & Y & \end{tikzcd}$$

Which feature is most useful to the model?

Instead of perturbing features of a trained model and observe how the predictions change, we can take a step back and ask how the different features contribute to the model's predictive performance.

A similar situation occurs in the case of *redundant features*.

A redundant feature is one that provides the same or similar information as another feature. Consider a data generating process like on the right:



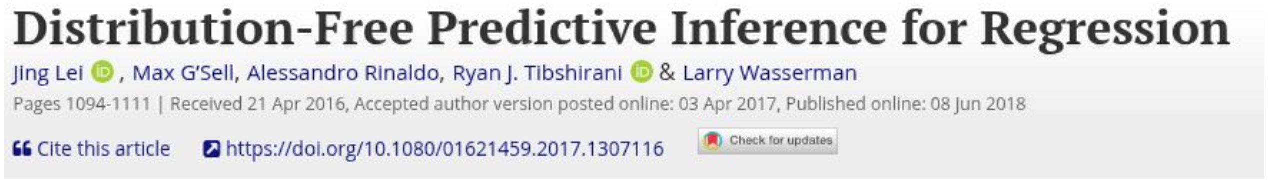
From a *feature selection* perspective, the features X_2 and X_3 are redundant.

Still, a perturbative approach might tell us that both features are important, as they might both have an effect on the prediction when perturbed.

Which feature is most useful to the model?

In 2018, Lei et al proposed “a model-free notion of variable importance, called leave-one-covariate-out or LOCO inference”.

This gave rise to the method more commonly known as Leave One Feature Out (LOFO), a method for obtaining feature importance by *leaving features out when training the model.*


Distribution-Free Predictive Inference for Regression

Jing Lei, Max G'Sell, Alessandro Rinaldo, Ryan J. Tibshirani & Larry Wasserman

Pages 1094-1111 | Received 21 Apr 2016, Accepted author version posted online: 03 Apr 2017, Published online: 08 Jun 2018

Cite this article https://doi.org/10.1080/01621459.2017.1307116 Check for updates

Leave One Feature Out (LOFO)

Basically: Retrain the model once per feature (each time leaving one feature out) and evaluate performance on the test set.

If removing a feature reduces predictive performance, the feature was important; if performance remains unchanged, the feature is not important.

Leave One Feature Out (LOFO)

Basically: Retrain the model once per feature (each time leaving one feature out) and evaluate performance on the test set.

If removing a feature reduces predictive performance, the feature was important; if performance remains unchanged, the feature is not important.

The LOFO importance for each left-out feature is calculated by comparing the missing-feature-performance to the performance of the full model.

Leave One Feature Out (LOFO)

Basically: Retrain the model once per feature (each time leaving one feature out) and evaluate performance on the test set.

If removing a feature reduces predictive performance, the feature was important; if performance remains unchanged, the feature is not important.

The LOFO importance for each left-out feature is calculated by comparing the missing-feature-performance to the performance of the full model.

Can the LOFO importance can be negative? What would this mean?

Leave One Feature Out (LOFO)

Basically: Retrain the model once per feature (each time leaving one feature out) and evaluate performance on the test set.

If removing a feature reduces predictive performance, the feature was important; if performance remains unchanged, the feature is not important.

The LOFO importance for each left-out feature is calculated by comparing the missing-feature-performance to the performance of the full model.

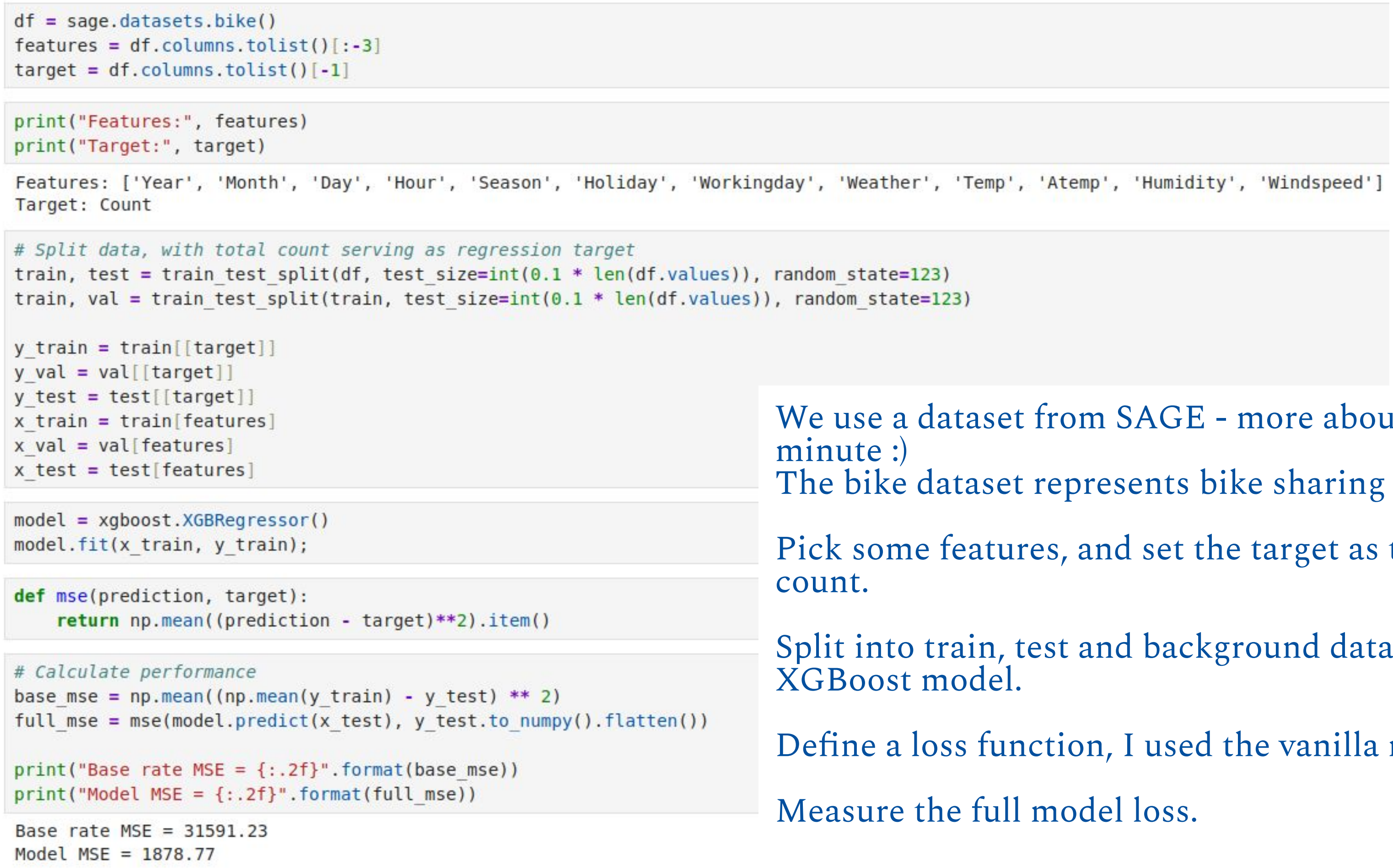
The LOFO importance can be negative if dropping a feature improves the model.

Since the model is retrained with the feature missing, we don't have to worry about sampling the absent feature or estimating marginal/conditional probabilities.

Pseudocode - LOFO

```
Input: Trained model $f$, list of features, training data $X_{train}$, $y_{train}$ and test data $X_{test}$, $y_{test}$, loss function $\ell$.

Step 1: measure the loss of the full model, $\ell_f$

Step 2: for each feature $j \in \{1,2, …, d\}$:

      Remove feature j from the datasets, creating $X_{train, -j}$ and $X_{test, -j}$

      Train a new model $f_{-j}$ on $(X_{train, -j}, y_{train})$

      Measure and keep the loss of new model, $\ell_{-j}$

Step 3: Calculate the LOFO per feature as $LOFO_j = \ell_{-j} / \ell_f$ or $LOFO_j = \ell_{-j} - \ell_f$

Step 4: sort and visualise the features by descending $LOFO_j$
```

Code :)



```
df = sage.datasets.bike()
features = df.columns.tolist()[:-3]
target = df.columns.tolist()[-1]
```

```
print("Features:", features)
print("Target:", target)
```

```
Features: ['Year', 'Month', 'Day', 'Hour', 'Season', 'Holiday', 'Workingday', 'Weather', 'Temp', 'Atemp', 'Humidity', 'Windspeed']
Target: Count
```

```
# Split data, with total count serving as regression target
train, test = train_test_split(df, test_size=int(0.1 * len(df.values)), random_state=123)
train, val = train_test_split(train, test_size=int(0.1 * len(df.values)), random_state=123)

y_train = train[[target]]
y_val = val[[target]]
y_test = test[[target]]
x_train = train[features]
x_val = val[features]
x_test = test[features]
```

```
model = xgboost.XGBRegressor()
model.fit(x_train, y_train);
```

```
def mse(prediction, target):
    return np.mean((prediction - target)**2).item()
```

```
# Calculate performance
base_mse = np.mean((np.mean(y_train) - y_test) ** 2)
full_mse = mse(model.predict(x_test), y_test.to_numpy().flatten())

print("Base rate MSE = {:.2f}".format(base_mse))
print("Model MSE = {:.2f}".format(full_mse))
```

```
Base rate MSE = 31591.23
Model MSE = 1878.77
```

We use a dataset from SAGE - more about that library in a minute :)
The bike dataset represents bike sharing demand.

Pick some features, and set the target as the bike demand count.

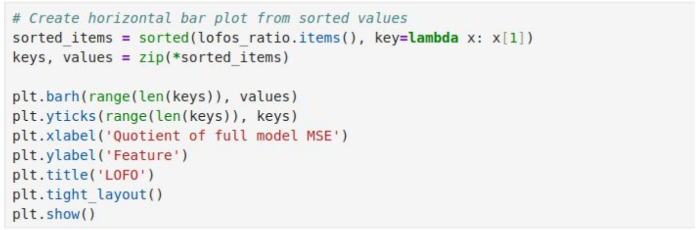
Split into train, test and background data, and train a simple XGBoost model.

Define a loss function, I used the vanilla mean squared error.

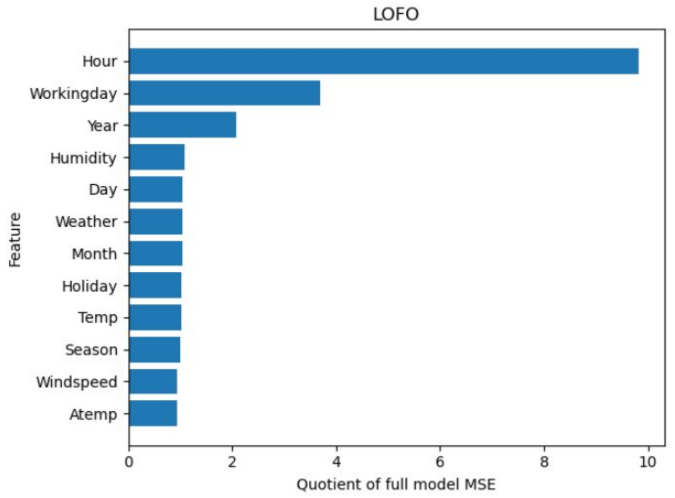
Measure the full model loss.

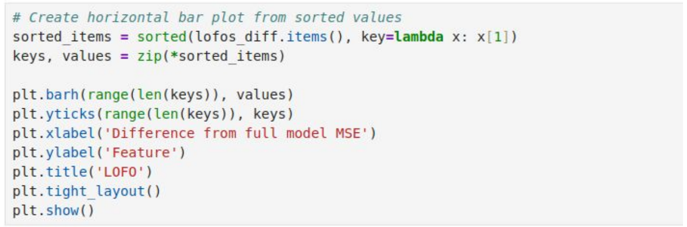
Homework, part 1:

Write the lofo function, using the provided notebook. You should be able to recreate either of these plots:

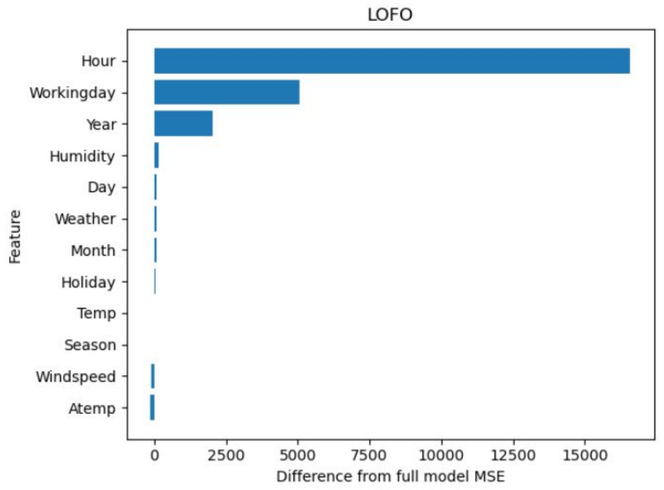


```python
# Create horizontal bar plot from sorted values
sorted_items = sorted(lofos_ratio.items(), key=lambda x: x[1])
keys, values = zip(*sorted_items)

plt.barh(range(len(keys)), values)
plt.yticks(range(len(keys)), keys)
plt.xlabel('Quotient of full model MSE')
plt.ylabel('Feature')
plt.title('LOFO')
plt.tight_layout()
plt.show()
```



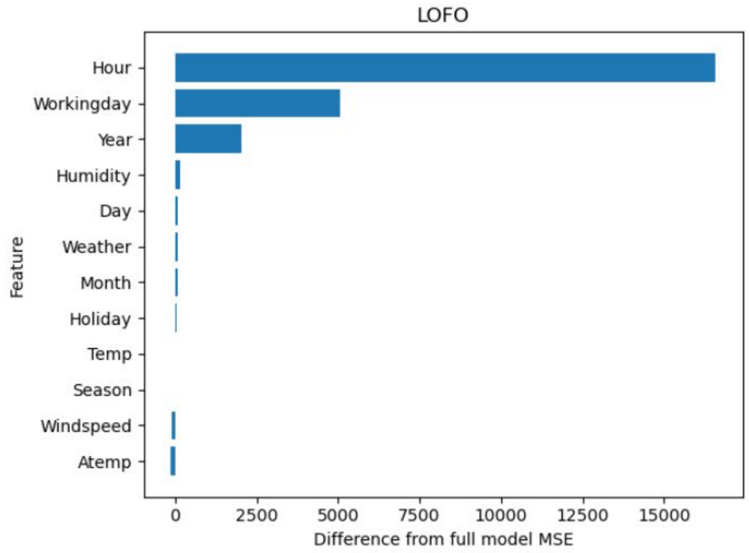


```python
# Create horizontal bar plot from sorted values
sorted_items = sorted(lofos_diff.items(), key=lambda x: x[1])
keys, values = zip(*sorted_items)

plt.barh(range(len(keys)), values)
plt.yticks(range(len(keys)), keys)
plt.xlabel('Difference from full model MSE')
plt.ylabel('Feature')
plt.title('LOFO')
plt.tight_layout()
plt.show()
```



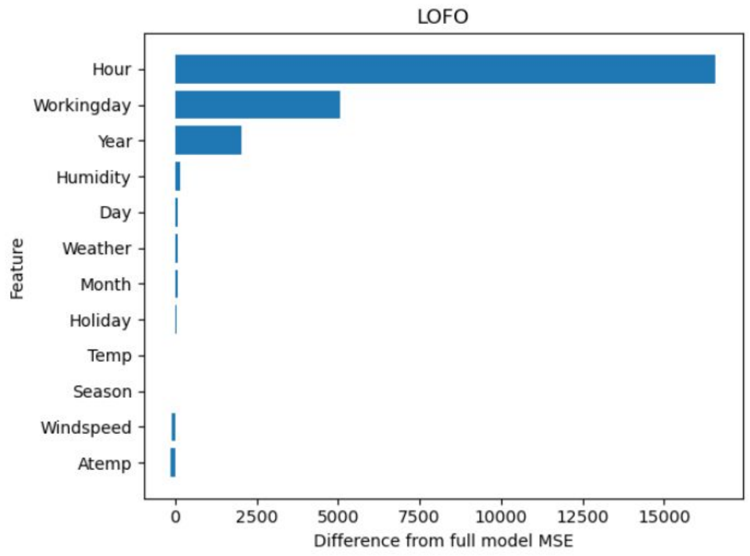
LOFO

How often do we need to retrain the model for N features?



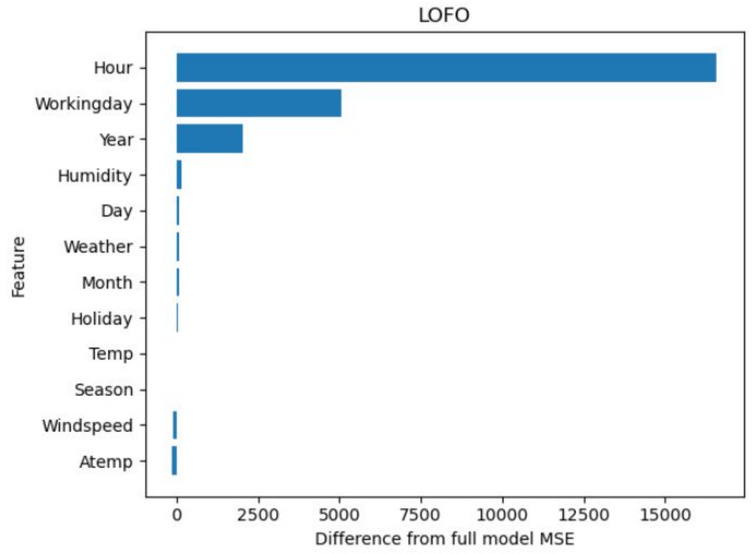
LOFO

We retrain the model once per feature, so N+1 times, if we include the original training of the full model.



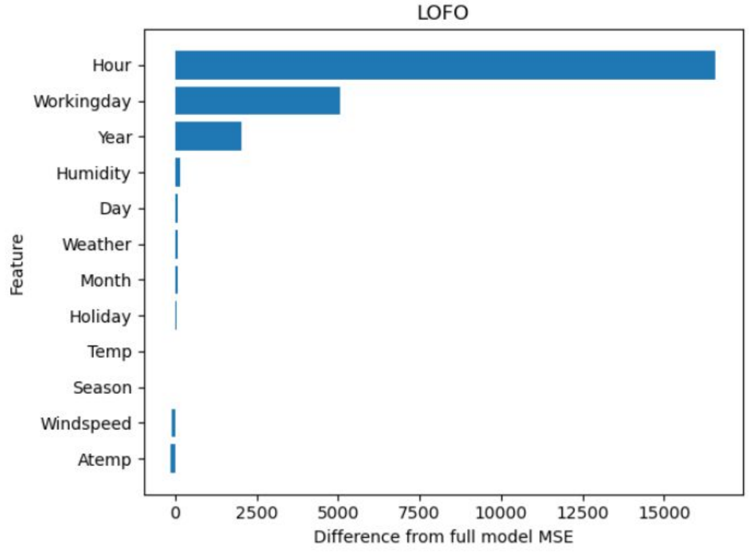
LOFO

And what does a feature's LOFO importance tell us?



LOFO

It literally tells us how the presence of that feature affects the model's predictive performance (loss or something else we choose to measure).



Leave One Feature Out (LOFO)

Where do we place this in the taxonomy?

Post-hoc?

Model-agnostic or model-specific?

Local or global?

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		

Leave One Feature Out (LOFO)

Where do we place this in the taxonomy?

Post-hoc: *We got a model and want to explain its behaviour. However, we need access to retrain the model!*

Model-agnostic or model-specific?

Local or global?

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		

Leave One Feature Out (LOFO)

Where do we place this in the taxonomy?

Post-hoc: *We got a model and want to explain its behaviour. However, we need access to retrain the model!*

Model-agnostic: *The feature importances do not depend on the model structure.*

Local or global?

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		

Leave One Feature Out (LOFO)

Where do we place this in the taxonomy?

Post-hoc: *We got a model and want to explain its behaviour. However, we need access to retrain the model!*

Model-agnostic: *The feature importances do not depend on the model structure.*

Global: *Retraining the model and studying the overall loss tells us about its global performance.*

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		

Leave One Feature Out (LOFO)

Where do we place this in the taxonomy?

Post-hoc: *We got a model and want to explain its behaviour. However, we need access to retrain the model!*

Model-agnostic: *The feature importances do not depend on the model structure.*

Global: *Retraining the model and studying the overall loss tells us about its global performance.*

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global	LOFO	

Benefits

What do you like?

Benefits

No package required: Implementing LOFO is fairly easy (and you'll do it yourself :))

Feature selection: a feature with a LOFO importance of zero can be removed without affecting the performance. *However, the LOFO importance can depend on the feature split, so you should do repeated Monte Carlo sampling to find out whether the importance varies.*

Explaining the data: In addition to being useful for feature selection, since LOFO provides insight into the data features' predictive power, it is useful for understanding the data in addition to the actual model.

Combined effects: LOFO can be extended to leave two or several features out, increasing the complexity of the calculation.

On manifold: LOFO doesn't create data points, like LIME, and doesn't model excluded features, like SHAP, and therefore doesn't run the risk of using unrealistic data points.

Limitations

What's not so great?

Limitations

Computational complexity: LOFO can be costly; it requires training the model N times for N features. If we want feature interaction effects, we have to retrain the model again per feature combination.

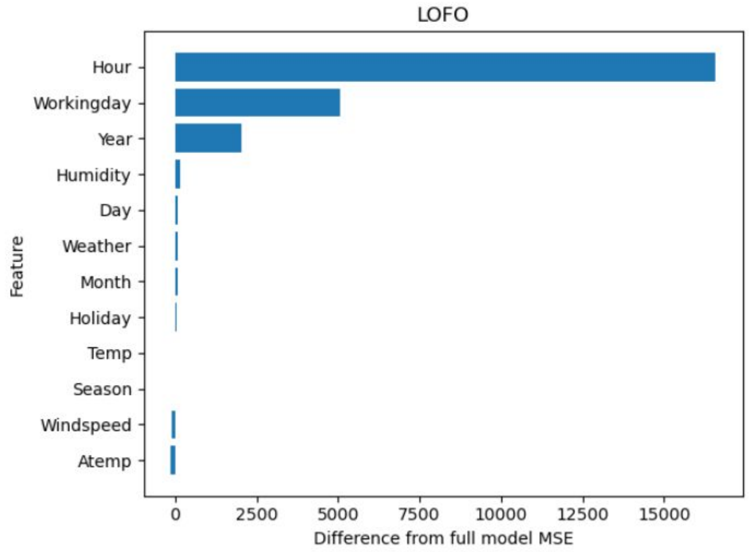
Interpretation: LOFO tells us which feature is important for the model's performance, but does not necessarily tell us how important that feature is for the full model. Therefore, LOFO is likely not useful e.g. for auditing a model. Since the feature importance is based on training new models, *the interpretation is not only based on the full model, but all models trained on the feature subsets.*

Hyperparameter optimization: When training on a subset of features, other hyperparameter settings than those for the full model may be preferable. If the hyperparameters were optimised for the full model, it is unclear whether such optimisation should also be done for the subsets of features. This is not only costly, but implies that we would end up with different models per subset.

Leave One Feature Out (LOFO)

What do you think - could we do better?

Did we solve the problem of redundant features?



Leave One Feature Out (LOFO)

Testing one feature at a time, LOFO doesn't capture features interactions.

If several features carry overlapping information (redundancy), LOFO tends to underestimate both, since removing one hardly changes performance.

In order to address these challenges, we could systematically remove features to get the effect of all features in all possible subsets, on the model's loss.

Have we encountered such a way of systematically removing features before? :)

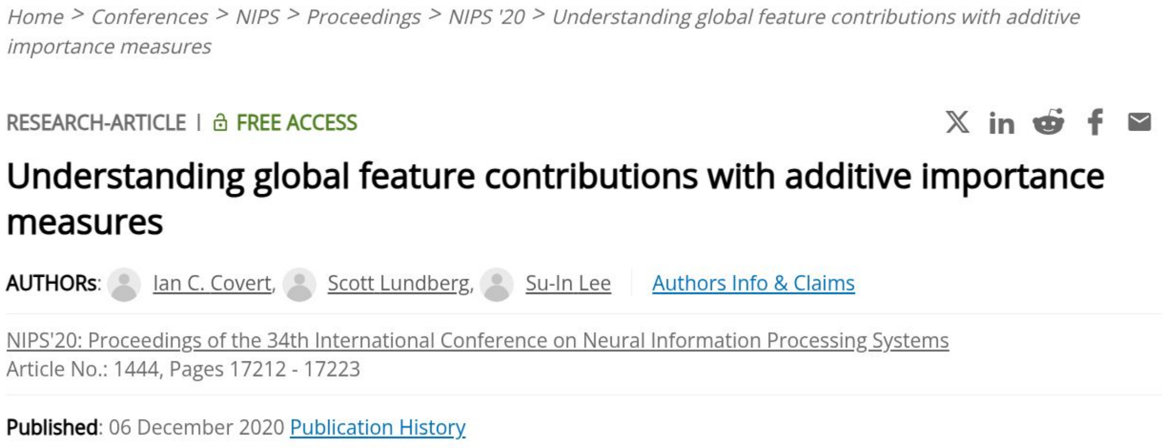
SAGE

Feature importance as predictive power

A 2020 paper by Covert, Lundberg & Lee (remember the last two guys?) suggests a Shapley value based approach similar to SHAP, called SAGE (Shapley additive global importance).

You can check out the git repo https://github.com/iancovert/sage for loads of coding examples.

SAGE answers the question "How much predictive power does a feature provide to the model?"



Home > Conferences > NIPS > Proceedings > NIPS '20 > Understanding global feature contributions with additive importance measures

RESEARCH-ARTICLE | FREE ACCESS

Understanding global feature contributions with additive importance measures

AUTHORs: Ian C. Covert, Scott Lundberg, Su-In Lee | Authors Info & Claims

NIPS'20: Proceedings of the 34th International Conference on Neural Information Processing Systems
Article No.: 1444, Pages 17212 - 17223

Published: 06 December 2020 Publication History

Feature importance as predictive power

Our full model is f, and we have data with features $(x_1, x_2, \ldots, x_d)$.

A subset of features is denoted $x_S \equiv \{x_i | \ i \in S\}$

The restricted model f_S is the model evaluated in the "absence" of features

$$f_S(x_S) = E\left[f(X)|X_S = x_S\right]$$

We can sample the missing features, in the set S, using the marginal or the conditional distribution $p(X_{S}|X_S = x_S)$.

(You may remember this as the main headache from SHAP)

Feature importance as predictive power

The **risk** of a model f under a loss function ℓ is defined as the expected loss over the true data distribution.

For the restricted model f_S, the population risk is

$$E\left[\ell(f_S(X_S), Y)\right]$$

meaning the loss of the model when given access only to the values of the features in S.

Feature importance as predictive power

The **risk** of a model f under a loss function ℓ is defined as the expected loss over the true data distribution.

For the restricted model f_S, the population risk is

$$E\left[\ell(f_S(X_S), Y)\right]$$

meaning the loss of the model when given access only to the values of the features in S.

Defining predictive power as the *reduction in risk* compared to the mean prediction, we define

$$v_f(S) = \underbrace{E\left[\ell(f_\emptyset(X_\emptyset), Y)\right]}_{\text{Expected loss using the mean prediction}} - \underbrace{E\left[\ell(f_S(X_S), Y)\right]}_{\text{Expected loss using the prediction of the model with access to the features in } S}$$

Feature importance as predictive power

The **risk** of a model f under a loss function ℓ is defined as the expected loss over the true data distribution.

For the restricted model f_S, the population risk is

$$E\left[\ell(f_S(X_S), Y)\right]$$

meaning the loss of the model when given access only to the values of the features in S.

Defining predictive power as the *reduction in risk* compared to the mean prediction, we define

$$v_f(S) = E\left[\ell(f_\emptyset(X_\emptyset), Y)\right] - E\left[\ell(f_S(X_S), Y)\right]$$

As before, the set function v is a characteristic function, and maps a set into a single real number.

Feature importance as predictive power

Defining predictive power as the *reduction in risk* compared to the mean prediction, we define

$$v_f(S) = \underbrace{E\left[\ell(f_\emptyset(X_\emptyset), Y)\right]}_{\textit{Expected loss using the mean prediction}} - \underbrace{E\left[\ell(f_S(X_S), Y)\right]}_{\textit{Expected loss using the prediction of the model with access to the features in } S}$$

This function quantifies the amount of predictive power f gets from the features in S.

We generally expect that including more features in S increases the predictive power, i.e. makes v_f larger.

(the "empty model without features" is approximated using the mean model prediction, and the loss taken between that and the target. However, do we need to estimate it?)

Feature importance as predictive power

SAGE is the Shapley decomposition of a model using as characteristic function

$$v_f(S) = E\left[\ell(f_\emptyset(X_\emptyset), Y)\right] - E\left[\ell(f_S(X_S), Y)\right]$$

Thus, a feature's SAGE value represents its total contribution to the model's predictive power.

Put differently, it measures how much the feature's presence lowers the expected loss (risk) when added to the model's available features.

What does SAGE explain? How is this different from what SHAP explains?

Feature importance as predictive power

SAGE is the Shapley decomposition of a model using as characteristic function

$$v_f(S) = E\left[\ell(f_\emptyset(X_\emptyset), Y)\right] - E\left[\ell(f_S(X_S), Y)\right]$$

Thus, a feature's SAGE value represents its total contribution to the model's predictive power.

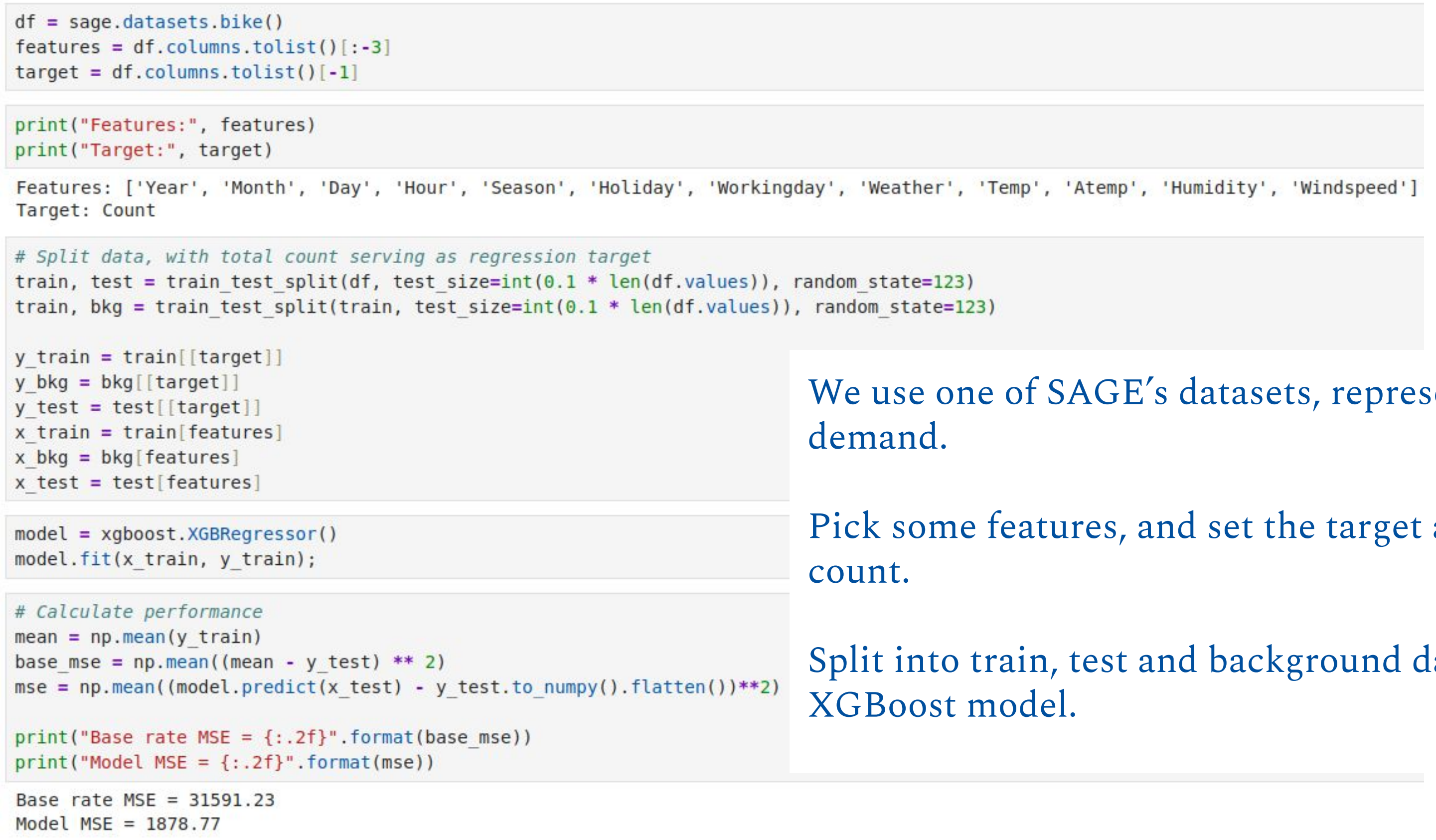
Put differently, it measures how much the feature's presence lowers the expected loss (risk) when added to the model's available features.

SAGE explains the model's performance, not its predictions (which SHAP does).

Code :)

norwegian open ai lab NTNU

SAGE in python (pip install sage-importance)



```
df = sage.datasets.bike()
features = df.columns.tolist()[:-3]
target = df.columns.tolist()[-1]
```

```
print("Features:", features)
print("Target:", target)
```

```
Features: ['Year', 'Month', 'Day', 'Hour', 'Season', 'Holiday', 'Workingday', 'Weather', 'Temp', 'Atemp', 'Humidity', 'Windspeed']
Target: Count
```

```
# Split data, with total count serving as regression target
train, test = train_test_split(df, test_size=int(0.1 * len(df.values)), random_state=123)
train, bkg = train_test_split(train, test_size=int(0.1 * len(df.values)), random_state=123)

y_train = train[[target]]
y_bkg = bkg[[target]]
y_test = test[[target]]
x_train = train[features]
x_bkg = bkg[features]
x_test = test[features]
```

```
model = xgboost.XGBRegressor()
model.fit(x_train, y_train);
```

```
# Calculate performance
mean = np.mean(y_train)
base_mse = np.mean((mean - y_test) ** 2)
mse = np.mean((model.predict(x_test) - y_test.to_numpy().flatten())**2)

print("Base rate MSE = {:.2f}".format(base_mse))
print("Model MSE = {:.2f}".format(mse))
```

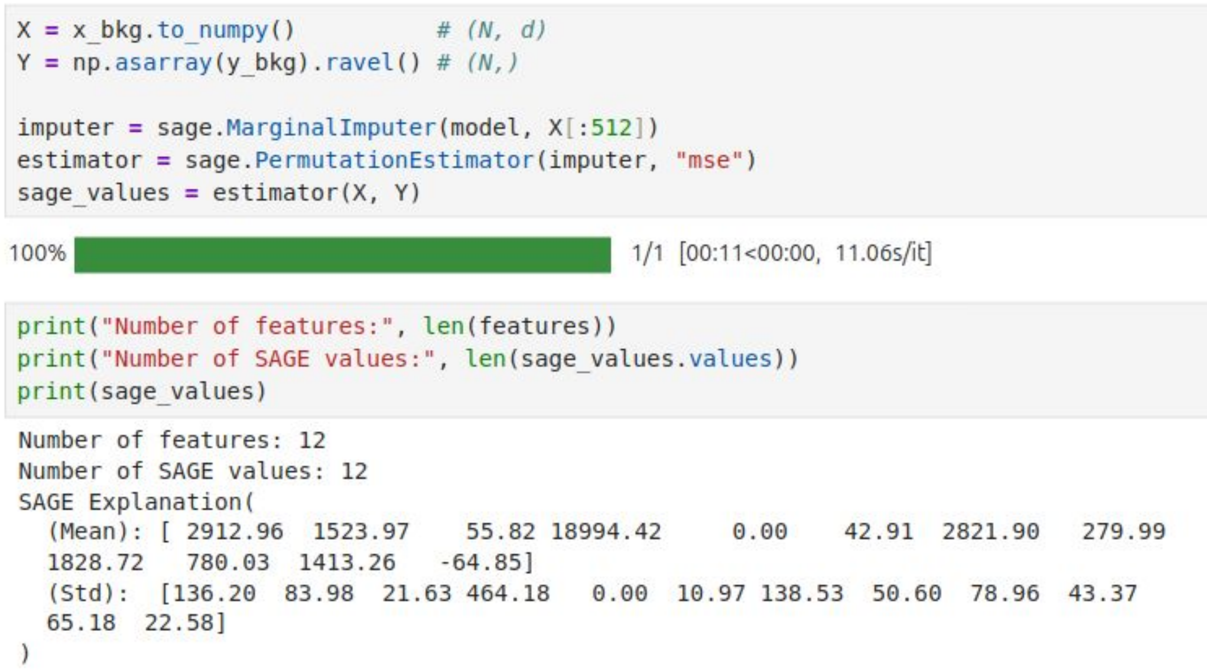
```
Base rate MSE = 31591.23
Model MSE = 1878.77
```

We use one of SAGE's datasets, representing bike sharing demand.

Pick some features, and set the target as the bike demand count.

Split into train, test and background data, and train a simple XGBoost model.

SAGE in python



```
X = x_bkg.to_numpy()         # (N, d)
Y = np.asarray(y_bkg).ravel() # (N,)

imputer = sage.MarginalImputer(model, X[:512])
estimator = sage.PermutationEstimator(imputer, "mse")
sage_values = estimator(X, Y)
```

```
100% 1/1 [00:11<00:00, 11.06s/it]
```

```
print("Number of features:", len(features))
print("Number of SAGE values:", len(sage_values.values))
print(sage_values)
```

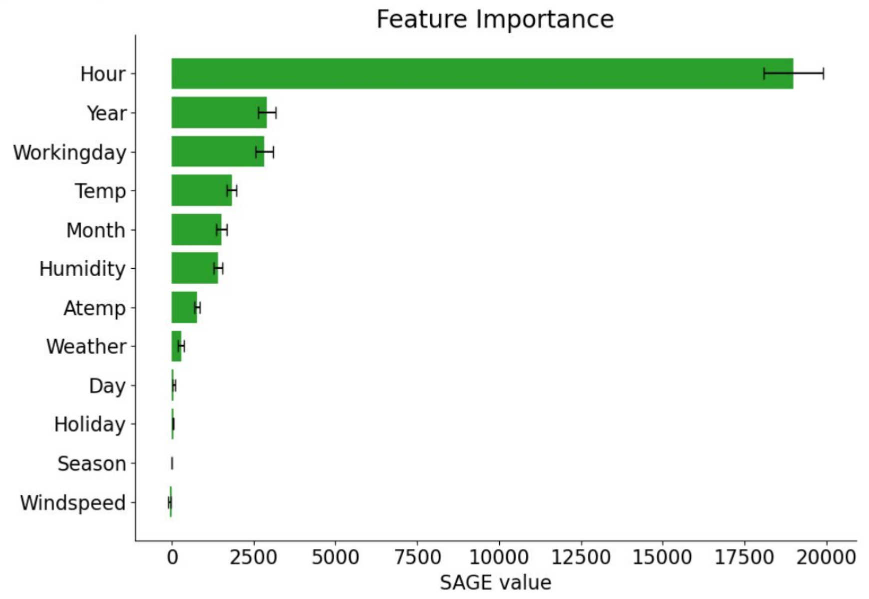
```
Number of features: 12
Number of SAGE values: 12
SAGE Explanation(
  (Mean): [ 2912.96  1523.97    55.82 18994.42     0.00    42.91  2821.90   279.99
  1828.72   780.03  1413.26   -64.85]
  (Std):  [136.20   83.98  21.63 464.18   0.00  10.97 138.53  50.60  78.96  43.37
  65.18  22.58]
)
```

The library can be a bit quarrelsome, but as long as your model accepts numpy arrays as input, and you convert properly, it usually runs.

The computation takes a while, before it returns a mean (approximated) SAGE value per feature, and one (approximated) standard deviation per feature.

SAGE in python

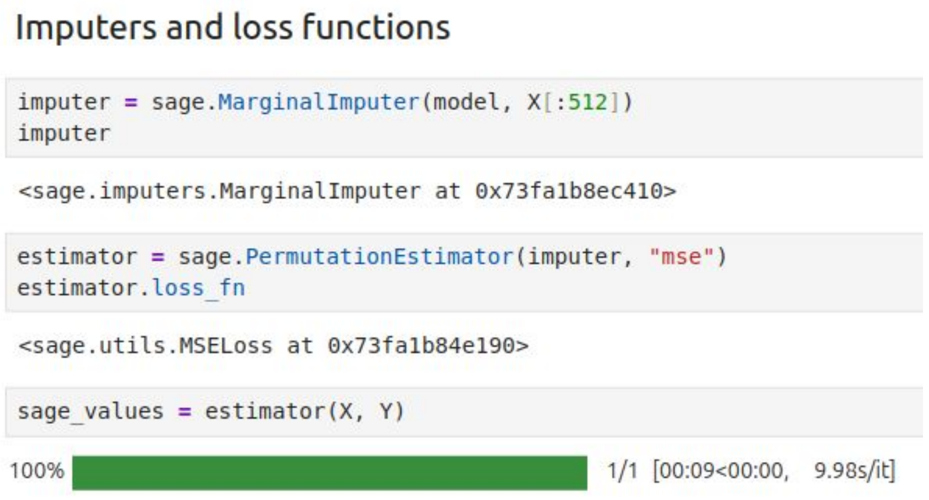
```
sage_values.plot(features)
```



The SAGE values are most easily studied as a bar plot, where the mean value represents the feature importance (contribution to loss reduction) and the standard deviation is indicated by black bars.

Based on this plot, what is the single most important feature for having a model with loss prediction loss?

The approximations...



Imputers and loss functions

```
imputer = sage.MarginalImputer(model, X[:512])
imputer
```

```
<sage.imputers.MarginalImputer at 0x73fa1b8ec410>
```

```
estimator = sage.PermutationEstimator(imputer, "mse")
estimator.loss_fn
```

```
<sage.utils.MSELoss at 0x73fa1b84e190>
```

```
sage_values = estimator(X, Y)
```

```
100% ████████████ 1/1 [00:09<00:00,  9.98s/it]
```

```
imputer = sage.MarginalImputer(model, X[:512])
```

“Use this model and these background data points to approximate what the model would output if some features were missing, assuming the missing features are independent of the observed ones.”

```
estimator = sage.PermuationsEstimator(imputer, mse)
```

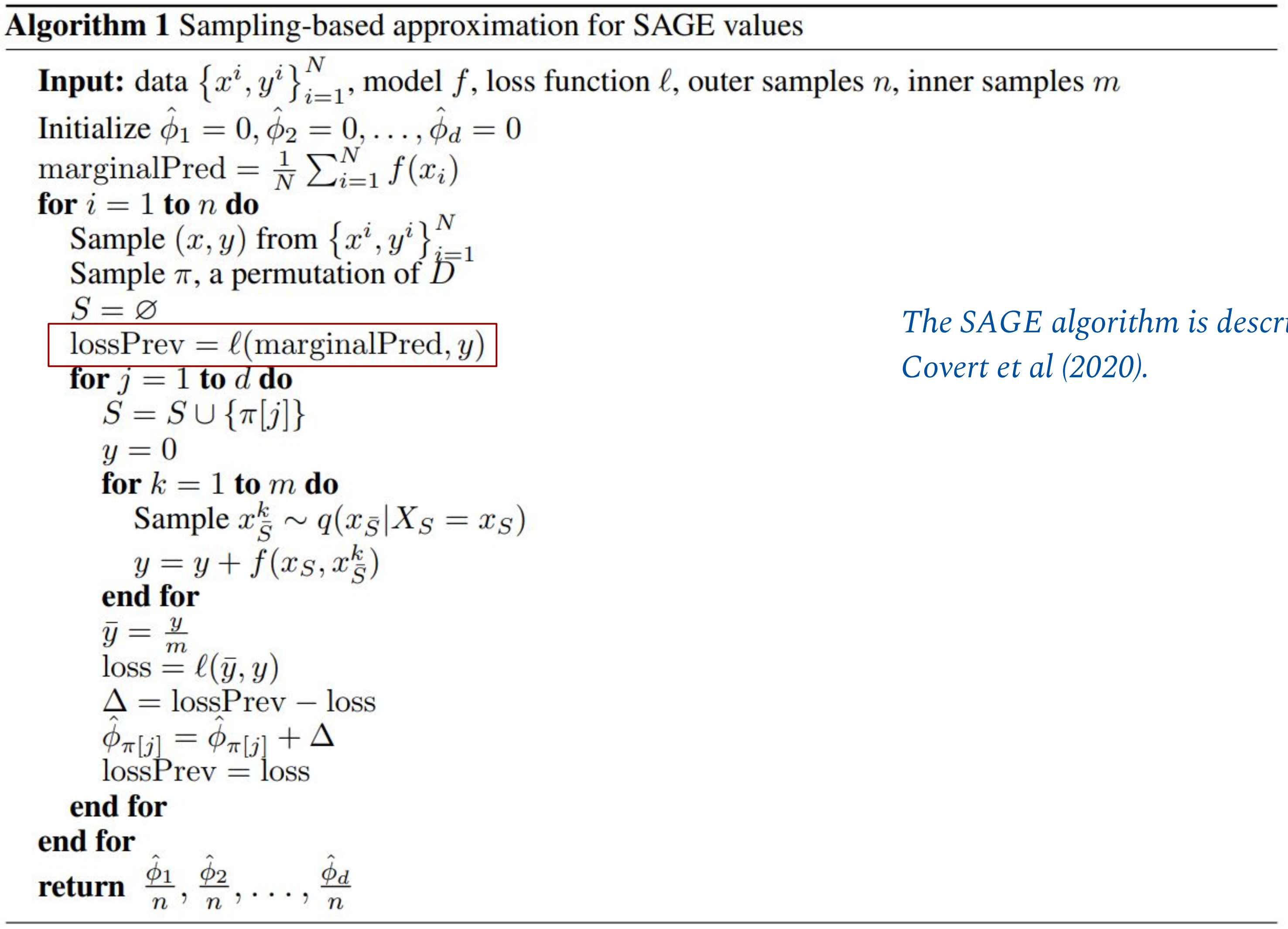
“Approximate the SAGE values by sampling random feature orderings and measuring the average reduction in mean squared error loss as each feature is added, estimating each feature’s expected contribution.”

The approximations...

The SAGE calculation involves the same challenges as SHAP, and they are solved in a similar manner in the python library.

Keep in mind, for SAGE:

- Feature absence is simulated by sampling (marginal distribution for the basic implementation).
- Not all coalitions are considered; they are sampled to approximate the contribution per feature.
- Estimation of the SAGE values is monitored for changes, and stopped once convergence is detected.



Algorithm 1 Sampling-based approximation for SAGE values

```
Input: data $\{x^i, y^i\}_{i=1}^N$, model $f$, loss function $\ell$, outer samples $n$, inner samples $m$
Initialize $\hat{\phi}_1 = 0, \hat{\phi}_2 = 0, \dots, \hat{\phi}_d = 0$
marginalPred $= \frac{1}{N} \sum_{i=1}^N f(x_i)$
for $i = 1$ to $n$ do
    Sample $(x, y)$ from $\{x^i, y^i\}_{i=1}^N$
    Sample $\pi$, a permutation of $D$
    $S = \varnothing$
    lossPrev $= \ell(\text{marginalPred}, y)$
    for $j = 1$ to $d$ do
        $S = S \cup \{\pi[j]\}$
        $y = 0$
        for $k = 1$ to $m$ do
            Sample $x_{\bar{S}}^k \sim q(x_{\bar{S}} | X_S = x_S)$
            $y = y + f(x_S, x_{\bar{S}}^k)$
        end for
        $\bar{y} = \frac{y}{m}$
        loss $= \ell(\bar{y}, y)$
        $\Delta =$ lossPrev $-$ loss
        $\hat{\phi}_{\pi[j]} = \hat{\phi}_{\pi[j]} + \Delta$
        lossPrev $=$ loss
    end for
end for
return $\frac{\hat{\phi}_1}{n}, \frac{\hat{\phi}_2}{n}, \dots, \frac{\hat{\phi}_d}{n}$
```

The SAGE algorithm is described in Appendix D of Covert et al (2020).

Feature importance as predictive power

SAGE satisfies the theoretical properties of the Shapley decomposition.

SAGE must be calculated across data points (just like SHAP). Due to the linearity property,
(given two games and the combined game, the Shapley values from each game combine linearly to the Shapley value of the combined game)
SAGE values are the expectation of per-instance SHAP values applied to the model loss.

Each SAGE value tells you how much, on average across all instances, a feature reduces the model's loss, i.e. how valuable it is for predictive performance.

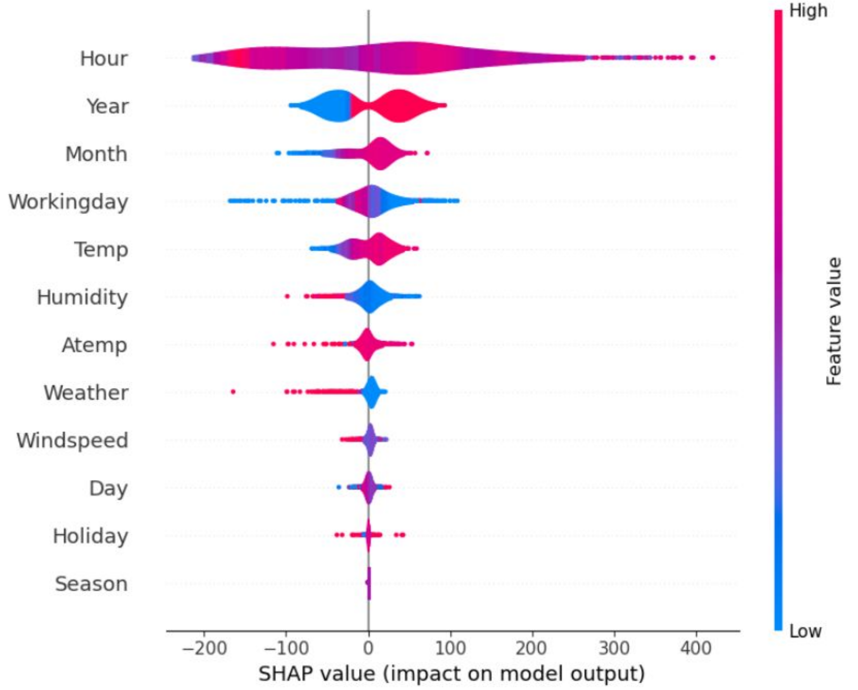
Comparison to SHAP

norwegian open ai lab NTNU

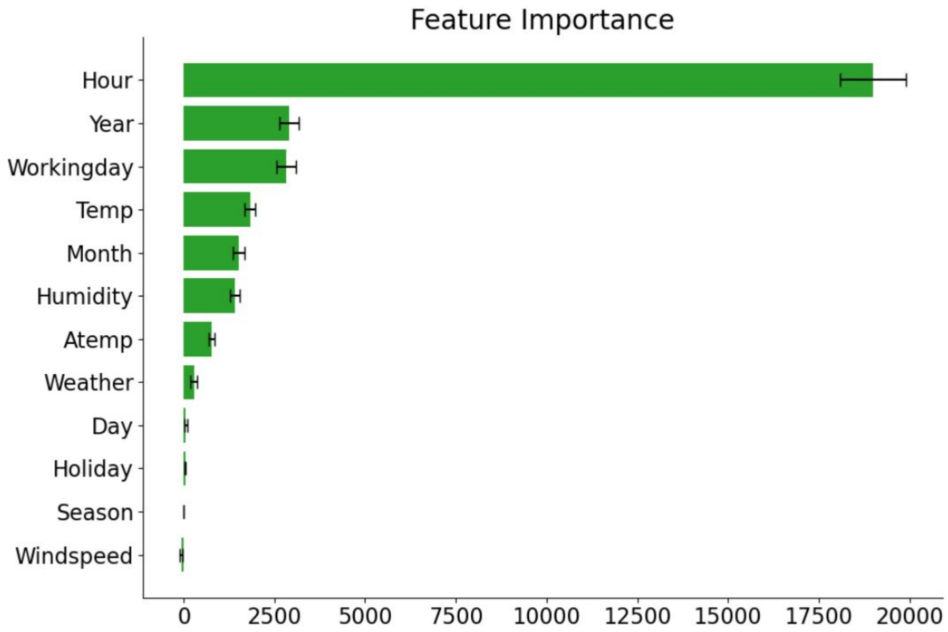
What does SHAP tell us?
What's the most important feature (and what does that mean)?
What does SAGE tell us?

```
explainer = shap.TreeExplainer(model, data=x_bkg)
shap_values = explainer.shap_values(x_test)
```

```
shap.summary_plot(shap_values, x_test, plot_type="violin")
```



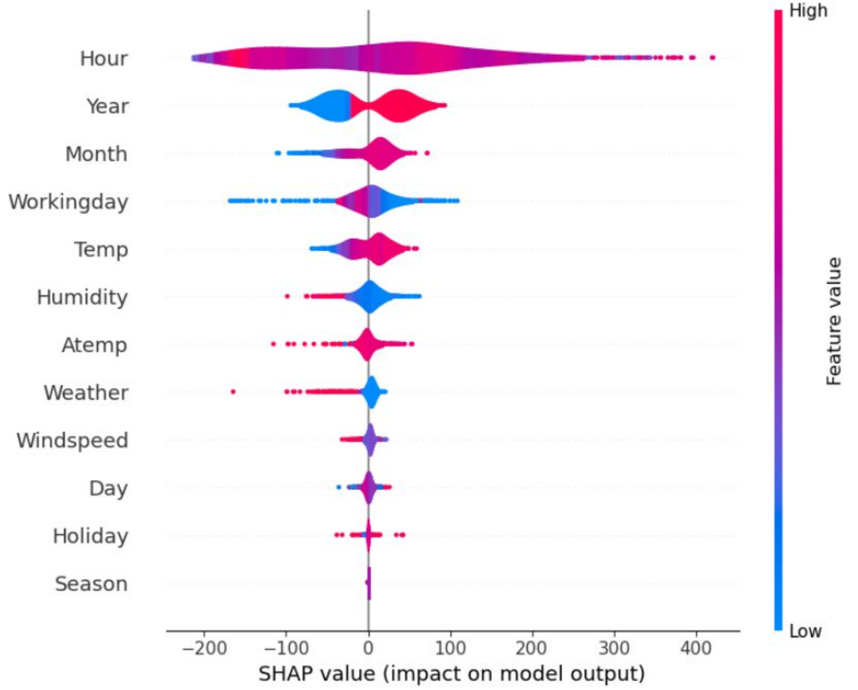
```
imputer = sage.MarginalImputer(model, X[:512])
estimator = sage.PermutationEstimator(imputer, "mse")
sage_values = estimator(X, Y)
sage_values.plot(features)
```



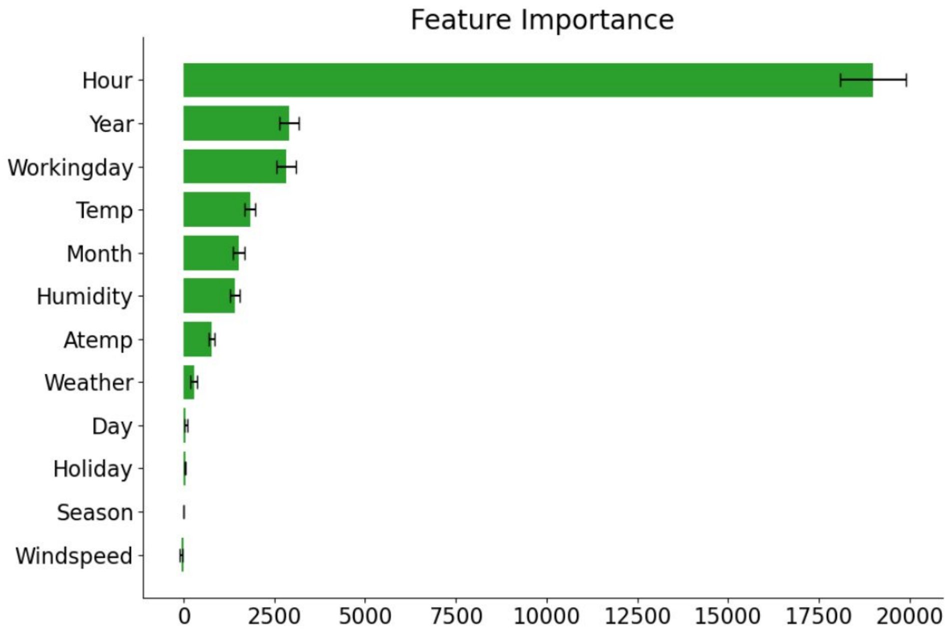
SHAP tells us *for each datapoint* how much its feature values drives the prediction away from the mean. SAGE tells us how much the presence of each feature decreases the model's loss on average.

```
explainer = shap.TreeExplainer(model, data=x_bkg)
shap_values = explainer.shap_values(x_test)
```

```
shap.summary_plot(shap_values, x_test, plot_type="violin")
```

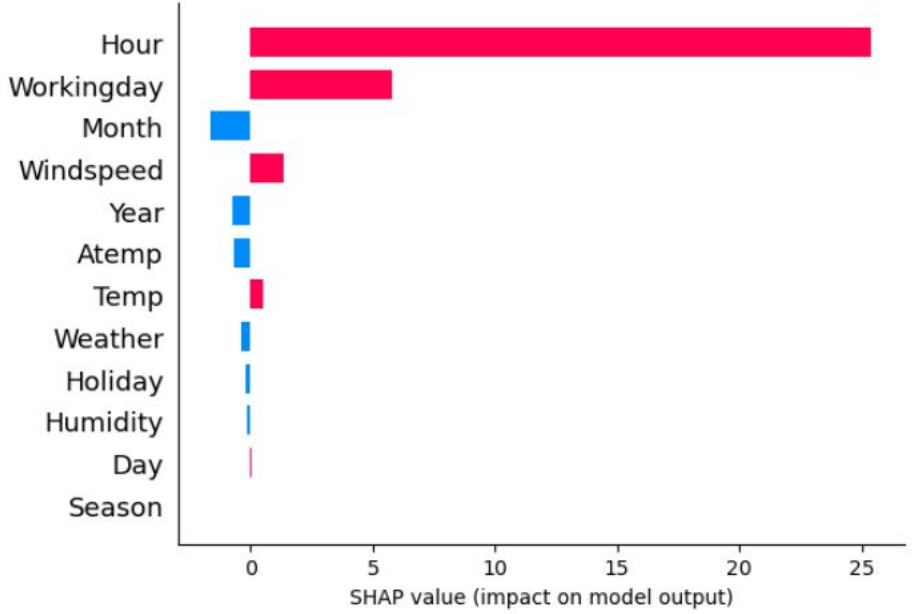


```
imputer = sage.MarginalImputer(model, X[:512])
estimator = sage.PermutationEstimator(imputer, "mse")
sage_values = estimator(X, Y)
sage_values.plot(features)
```

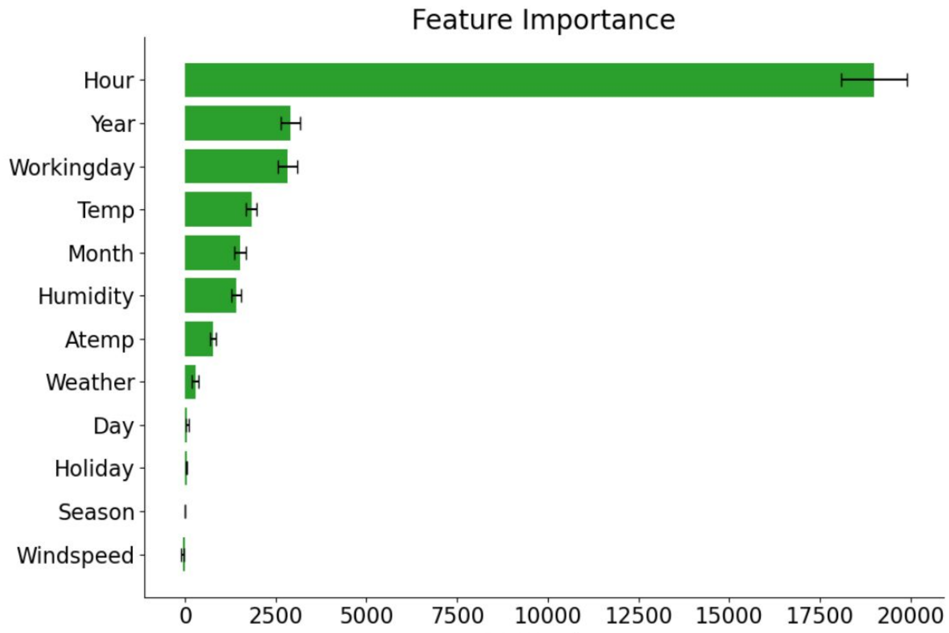


This plot shows the mean SHAP values.
Can we compare these to the SAGE values?

```
shap_mean = np.mean(shap_values, axis=0)
shap.bar_plot(shap_mean, feature_names=features, max_display=12)
```

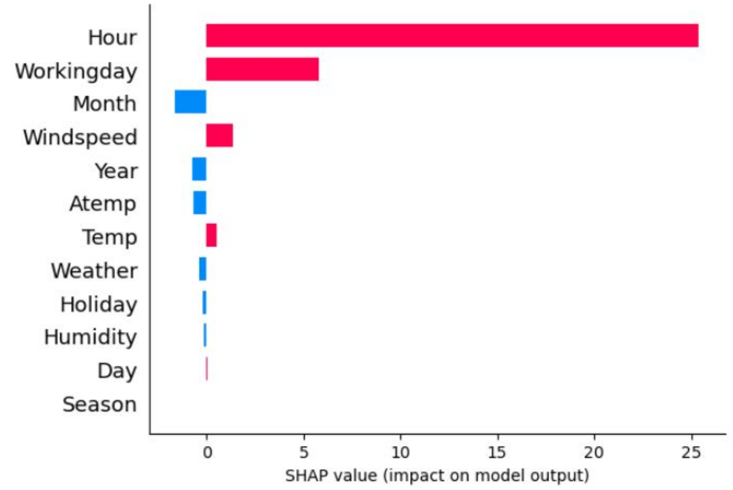


```
imputer = sage.MarginalImputer(model, X[:512])
estimator = sage.PermutationEstimator(imputer, "mse")
sage_values = estimator(X, Y)
sage_values.plot(features)
```



Can we compare mean SHAP values to the SAGE values?
Hint: Look at 'Year'.

```
shap_mean = np.mean(shap_values, axis=0)
shap.bar_plot(shap_mean, feature_names=features, max_display=12)
```

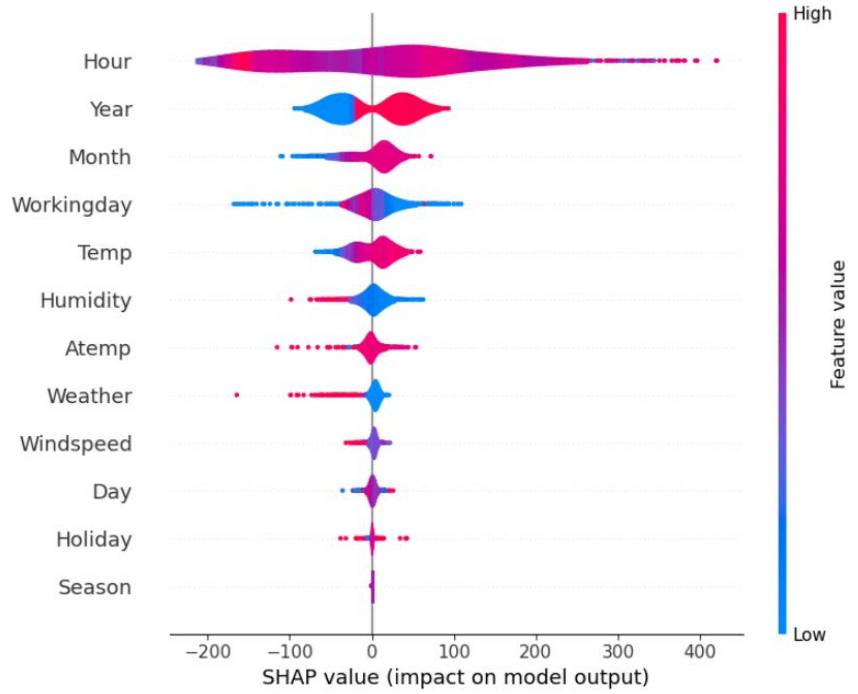


```
df["Year"]
```

```
0        2011
1        2011
2        2011
3        2011
4        2011
         ...
10881    2012
10882    2012
10883    2012
10884    2012
10885    2012
Name: Year, Length: 10886, dtype: int32
```

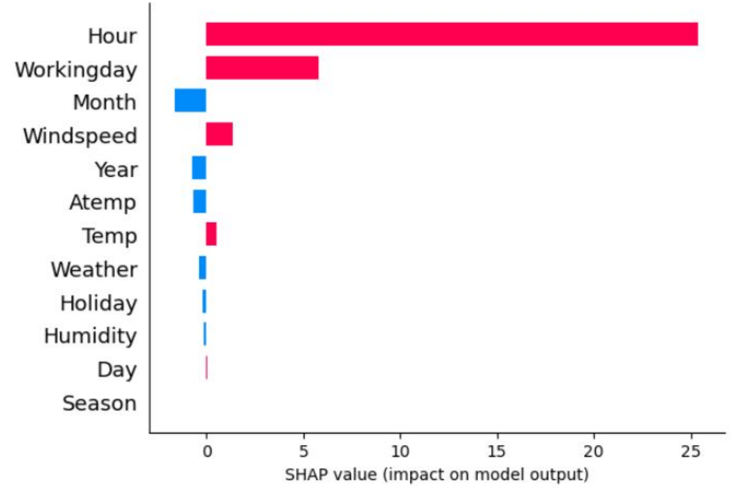
```
explainer = shap.TreeExplainer(model, data=x_bkg)
shap_values = explainer.shap_values(x_test)
```

```
shap.summary_plot(shap_values, x_test, plot_type="violin")
```



When averaging SHAP values, their contributions can cancel out: Look at 'Year', which has a binary distribution. The mean SHAP value for 'Year' is misleading; 2011 was a bad year, while 2012 was a good year.

```
shap_mean = np.mean(shap_values, axis=0)
shap.bar_plot(shap_mean, feature_names=features, max_display=12)
```

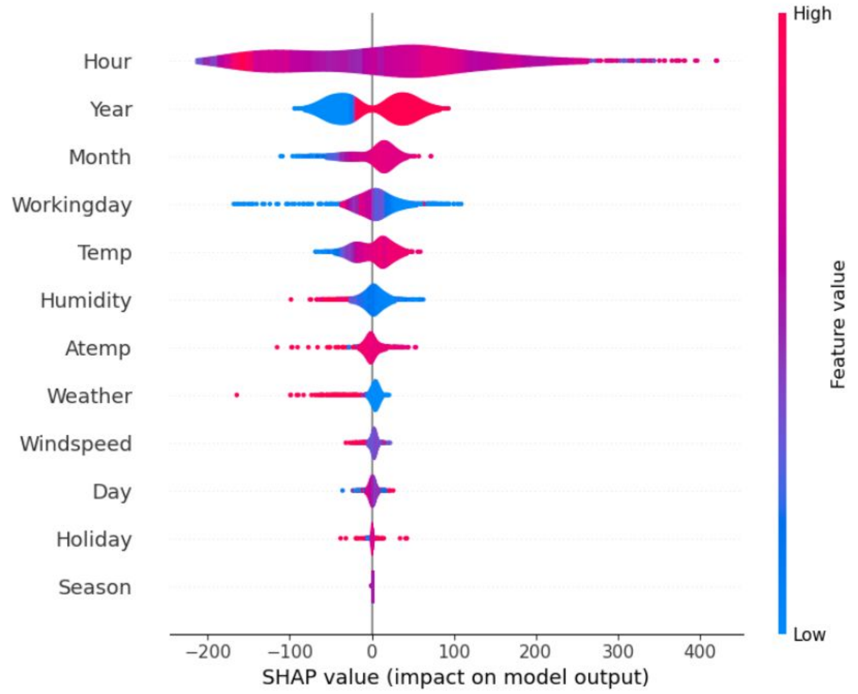


```
df["Year"]
```

```
0        2011
1        2011
2        2011
3        2011
4        2011
         ...
10881    2012
10882    2012
10883    2012
10884    2012
10885    2012
Name: Year, Length: 10886, dtype: int32
```

```
explainer = shap.TreeExplainer(model, data=x_bkg)
shap_values = explainer.shap_values(x_test)
```

```
shap.summary_plot(shap_values, x_test, plot_type="violin")
```

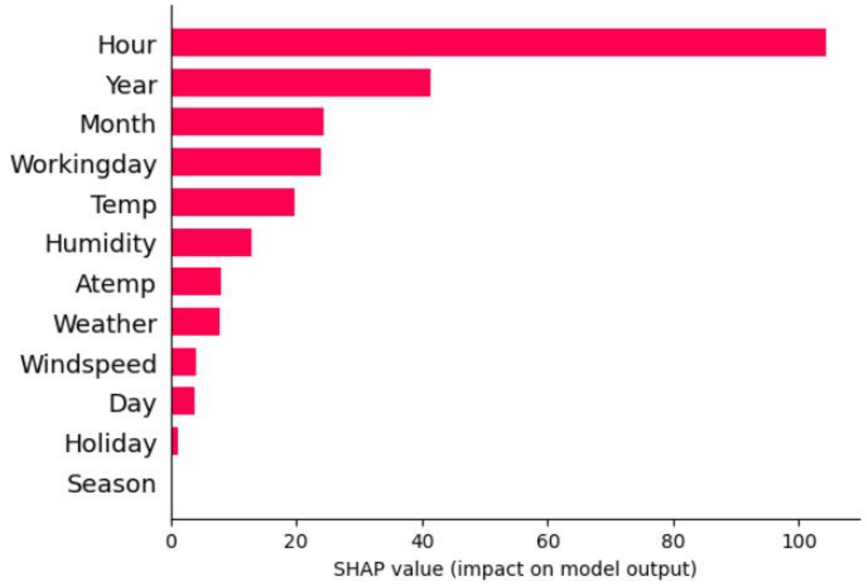


norwegian open ai lab NTNU

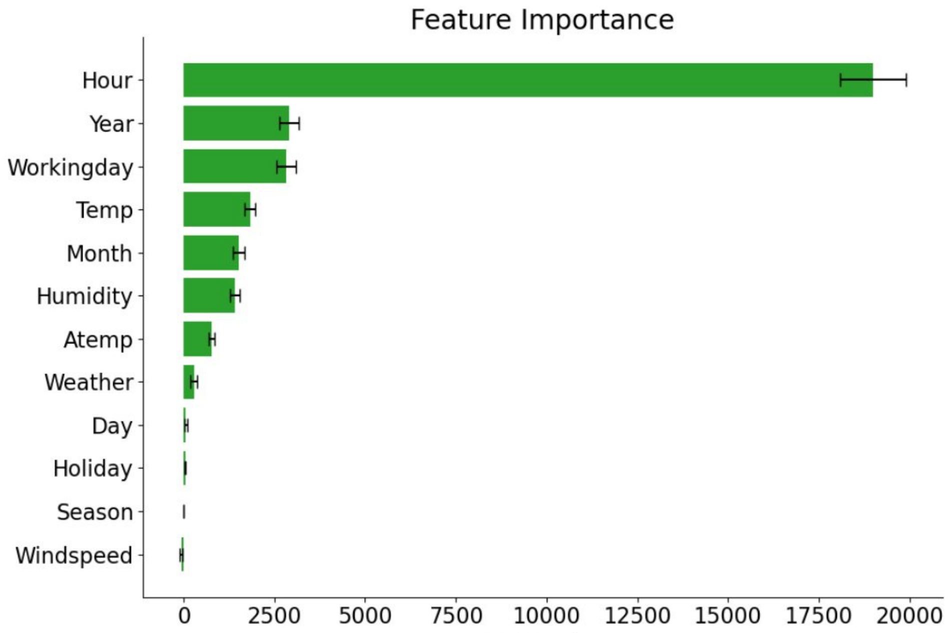
Interpreting the absolute SHAP value as a feature's absolute contribution to the model prediction, we can compare the mean absolute SHAP values to the SAGE values.

Would you rather use SHAP or SAGE for *feature selection*?

```
shap_mean = np.mean(np.abs(shap_values), axis=0)
shap.bar_plot(shap_mean, feature_names=features, max_display=12)
```

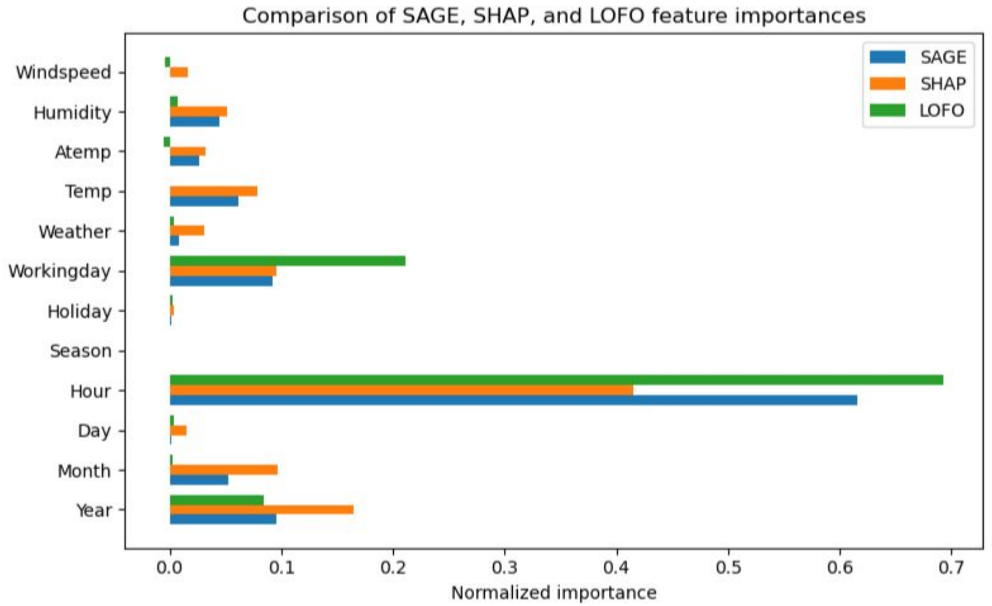


```
imputer = sage.MarginalImputer(model, X[:512])
estimator = sage.PermutationEstimator(imputer, "mse")
sage_values = estimator(X, Y)
sage_values.plot(features)
```



Comparison to SHAP and LOFO

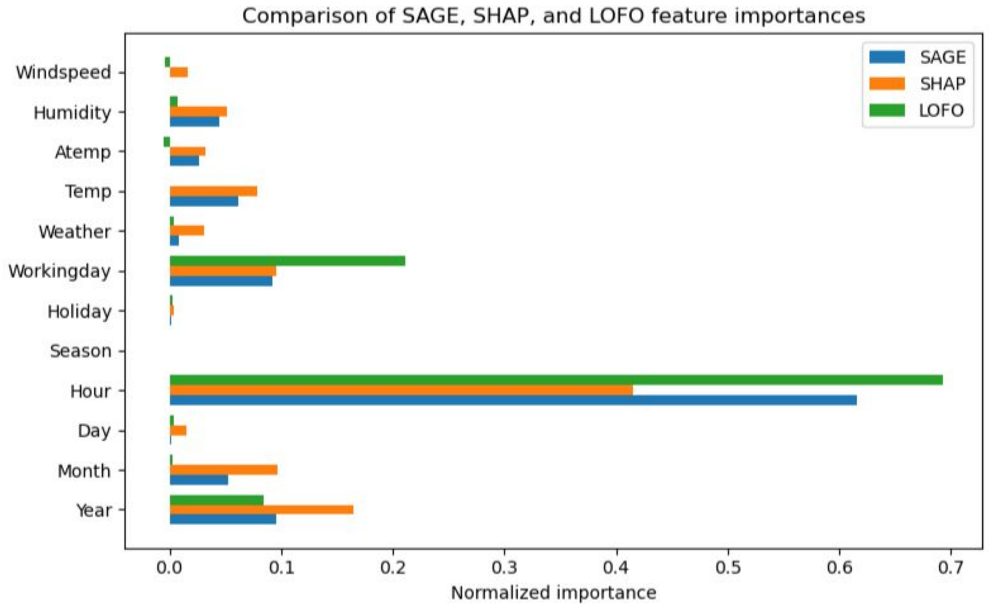
SAGE, SHAP and LOFO (normalised)



Do the methods agree?

Did you expect them to agree?

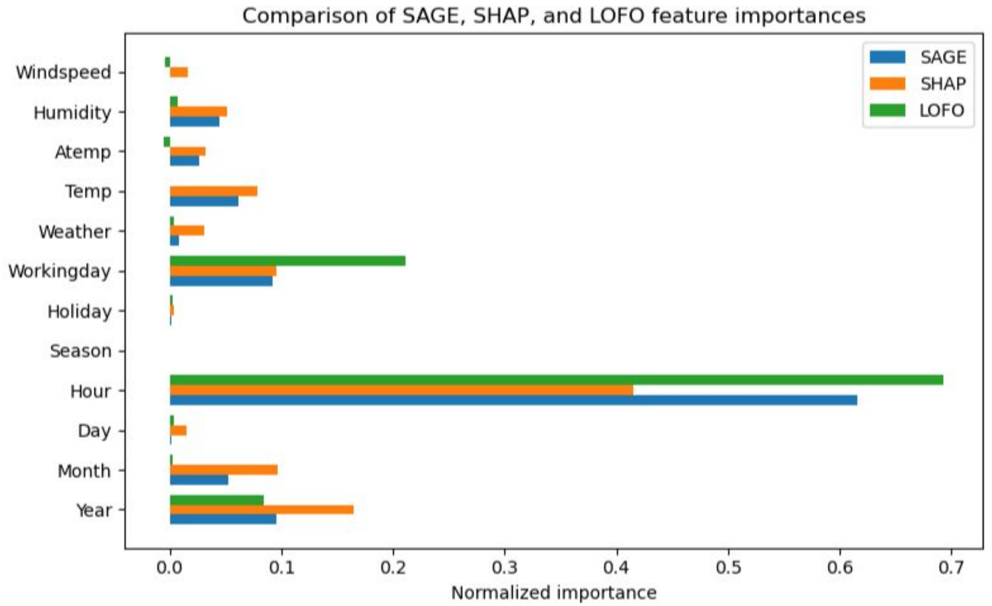
SAGE, SHAP and LOFO (normalised)



Look at the feature 'Month'.

What does it mean that it has a high SHAP value but low LOFO importance?

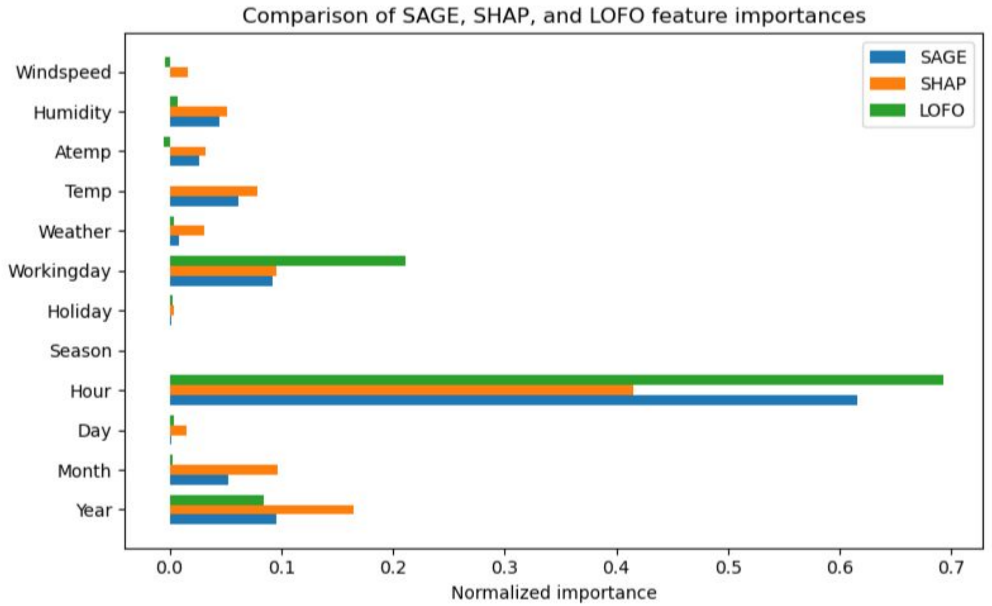
SAGE, SHAP and LOFO (normalised)



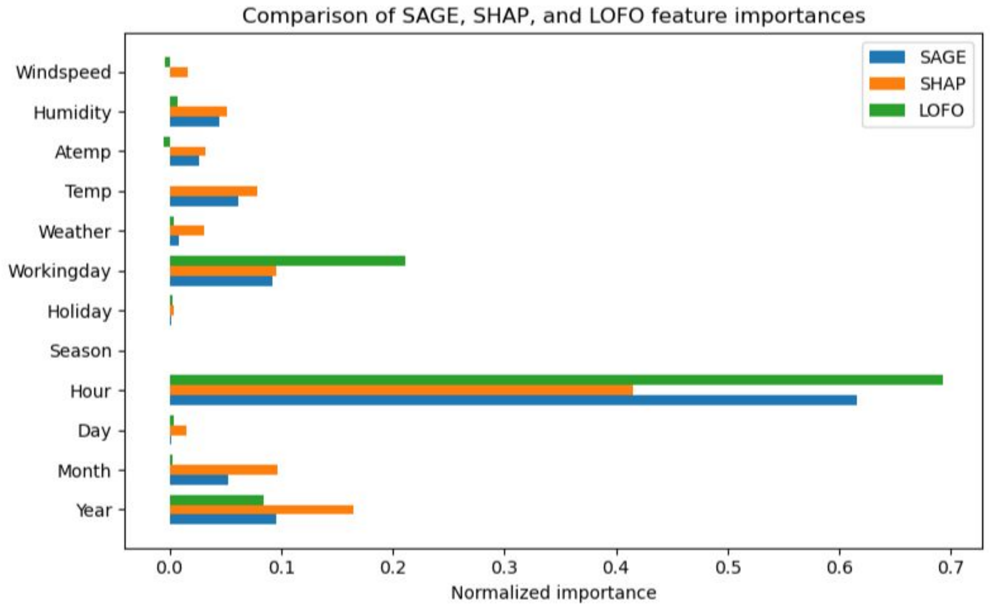
Look at the feature 'Month'.

This feature seems to strongly affect predictions (high SHAP) but have a small effect on the model loss during retraining (low LOFO).

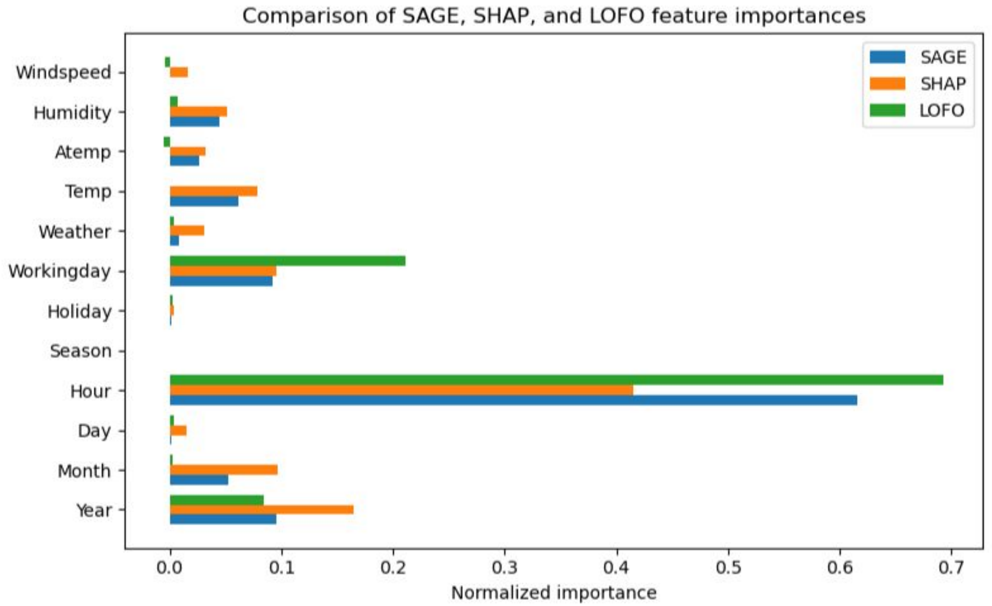
Method	Asks	Measures
SAGE		
SHAP		
LOFO		



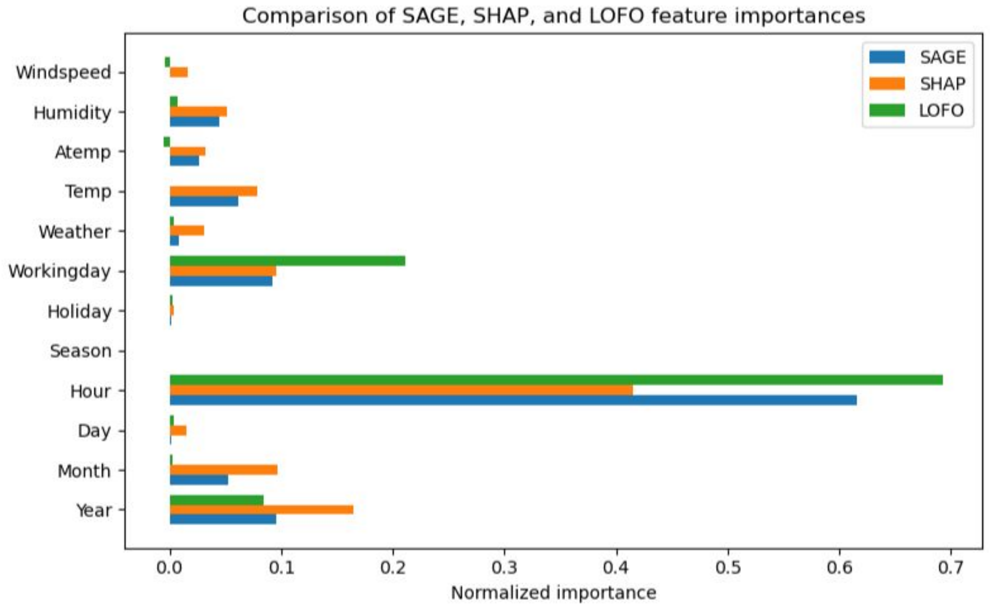
Method	Asks	Measures
SAGE	*"How much does this feature reduce expected loss?"*	Utility for accuracy
SHAP		
LOFO		



Method	Asks	Measures
SAGE	*"How much does this feature reduce expected loss?"*	Utility for accuracy
SHAP	*"How much does this feature move model predictions?"*	Influence on prediction(s)
LOFO		



Method	Asks	Measures
SAGE	*"How much does this feature reduce expected loss?"*	Utility for accuracy
SHAP	*"How much does this feature move model predictions?"*	Influence on prediction(s)
LOFO	*"How much does the loss increase if we remove this feature and retrain?"*	Replaceability



Homework, part 2:

Study the correlation structure of the data. What can you say about the feature ‘Month′?

Find an instance where the SHAP value of ‘Month′ is high and display its SHAP values. What do you find?

Based on this, can you say something about why the SHAP, SAGE and LOFO values might disagree about this feature?

Shapley additive global importance (SAGE)

Where do we place this in the taxonomy?

Post-hoc?

Model-agnostic or model-specific?

Local or global?

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		

Shapley additive global importance (SAGE)

Where do we place this in the taxonomy?

Post-hoc: *We got a model and want to explain its behaviour. No retraining required.*

Model-agnostic or model-specific?

Local or global?

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		

Shapley additive global importance (SAGE)

Where do we place this in the taxonomy?

Post-hoc: *We got a model and want to explain its behaviour. No retraining required.*

Model-agnostic: *The feature importances do not depend on the model structure.*

Local or global?

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		

Shapley additive global importance (SAGE)

Where do we place this in the taxonomy?

Post-hoc: *We got a model and want to explain its behaviour. No retraining required.*

Model-agnostic: *The feature importances do not depend on the model structure.*

Global: *Retraining the model and studying the overall loss tells us about its global performance.*

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		

Shapley additive global importance (SAGE)

Where do we place this in the taxonomy?

Post-hoc: *We got a model and want to explain its behaviour. No retraining required.*

Model-agnostic: *The feature importances do not depend on the model structure.*

Global: *Retraining the model and studying the overall loss tells us about its global performance.*

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global	SAGE	

Story of model agnostic explanations
The End

XAI Metrics

Evaluating explanations

This concludes the lectures on model agnostic explanations.

After learning about so many XAI methods, we need to ask ourselves:

How do we know that our explanations are any good?

How do we know that they tells us what we think they tell us, that they're not based on unrealistic data or spurious correlations, if they behave inconsistently across data points, or fails under small perturbations?

Such issues cannot be detected without systematic assessment; we cannot look at an explanation and know whether it is stable, faithful to the model, representative of our data, etc

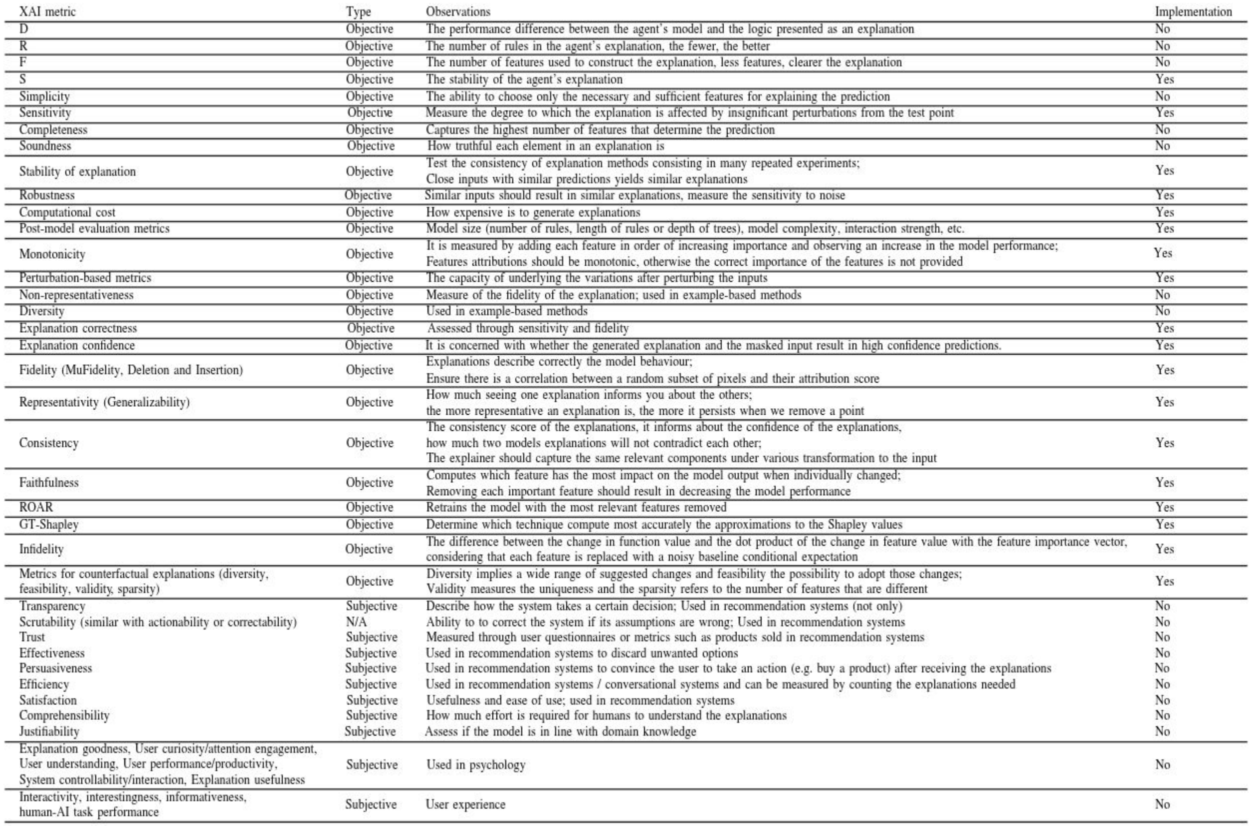
Evaluating explanations

When we've implemented a data based method and want to know if it's any good, what do we usually do?

We use metrics.

What are good XAI metrics?

Evaluation Metrics in Explainable Artificial Intelligence (XAI) (2022)



XAI metric	Type	Observations	Implementation
D	Objective	The performance difference between the agent's model and the logic presented as an explanation	No
R	Objective	The number of rules in the agent's explanation, the fewer, the better	No
F	Objective	The number of features used to construct the explanation, less features, clearer the explanation	No
S	Objective	The stability of the agent's explanation	Yes
Simplicity	Objective	The ability to choose only the necessary and sufficient features for explaining the prediction	No
Sensitivity	Objective	Measure the degree to which the explanation is affected by insignificant perturbations from the test point	Yes
Completeness	Objective	Captures the highest number of features that determine the prediction	No
Soundness	Objective	How truthful each element in an explanation is	No
Stability of explanation	Objective	Test the consistency of explanation methods consisting in many repeated experiments; Close inputs with similar predictions yields similar explanations	Yes
Robustness	Objective	Similar inputs should result in similar explanations, measure the sensitivity to noise	Yes
Computational cost	Objective	How expensive is to generate explanations	Yes
Post-model evaluation metrics	Objective	Model size (number of rules, length of rules or depth of trees), model complexity, interaction strength, etc.	Yes
Monotonicity	Objective	It is measured by adding each feature in order of increasing importance and observing an increase in the model performance; Features attributions should be monotonic, otherwise the correct importance of the features is not provided	Yes
Perturbation-based metrics	Objective	The capacity of underlying the variations after perturbing the inputs	Yes
Non-representativeness	Objective	Measure of the fidelity of the explanation; used in example-based methods	No
Diversity	Objective	Used in example-based methods	No
Explanation correctness	Objective	Assessed through sensitivity and fidelity	Yes
Explanation confidence	Objective	It is concerned with whether the generated explanation and the masked input result in high confidence predictions.	Yes
Fidelity (MuFidelity, Deletion and Insertion)	Objective	Explanations describe correctly the model behaviour; Ensure there is a correlation between a random subset of pixels and their attribution score	Yes
Representativity (Generalizability)	Objective	How much seeing one explanation informs you about the others; the more representative an explanation is, the more it persists when we remove a point	Yes
Consistency	Objective	The consistency score of the explanations, it informs about the confidence of the explanations, how much two models explanations will not contradict each other; The explainer should capture the same relevant components under various transformation to the input	Yes
Faithfulness	Objective	Computes which feature has the most impact on the model output when individually changed; Removing each important feature should result in decreasing the model performance	Yes
ROAR	Objective	Retrains the model with the most relevant features removed	Yes
GT-Shapley	Objective	Determine which technique compute most accurately the approximations to the Shapley values	Yes
Infidelity	Objective	The difference between the change in function value and the dot product of the change in feature value with the feature importance vector, considering that each feature is replaced with a noisy baseline conditional expectation	Yes
Metrics for counterfactual explanations (diversity, feasibility, validity, sparsity)	Objective	Diversity implies a wide range of suggested changes and feasibility the possibility to adopt those changes; Validity measures the uniqueness and the sparsity refers to the number of features that are different	Yes
Transparency	Subjective	Describe how the system takes a certain decision; Used in recommendation systems (not only)	No
Scrutability (similar with actionability or correctability)	N/A	Ability to to correct the system if its assumptions are wrong; Used in recommendation systems	No
Trust	Subjective	Measured through user questionnaires or metrics such as products sold in recommendation systems	No
Effectiveness	Subjective	Used in recommendation systems to discard unwanted options	No
Persuasiveness	Subjective	Used in recommendation systems to convince the user to take an action (e.g. buy a product) after receiving the explanations	No
Efficiency	Subjective	Used in recommendation systems / conversational systems and can be measured by counting the explanations needed	No
Satisfaction	Subjective	Usefulness and ease of use; used in recommendation systems	No
Comprehensibility	Subjective	How much effort is required for humans to understand the explanations	No
Justifiability	Subjective	Assess if the model is in line with domain knowledge	No
Explanation goodness, User curiosity/attention engagement, User understanding, User performance/productivity, System controllability/interaction, Explanation usefulness	Subjective	Used in psychology	No
Interactivity, interestingness, informativeness, human-AI task performance	Subjective	User experience	No

A jungle of metrics

There are loads of papers, blog posts and also libraries (Holistic AI, Quantus, Saliency-Bench, ...)

Let's take a step back: *Why do we use metrics? As in, why don't we use the loss function as a metric?*

A jungle of metrics

There are loads of papers, blog posts and also libraries (Holistic AI, Quantus, Saliency-Bench, …)

Metrics serve to evaluate and report.

The loss function should not be used as a metric partially because the loss is usually not interpretable *(most people don't know what is a "good" MSE value, and without a baseline, it's not very informative)* but primarily because **optimizing and measuring performance using the same criterion leads to goal hacking**.

It's important to evaluate using a criterion that was not directly optimised for.

Also, for gradient descent, the loss function must be differentiable, which doesn't apply to the metric.

Always remember: **Loss guides learning; metrics judge results.**

XAI metrics

There exists no universally accepted definition of what a “correct” explanation is, or what properties an explanation should fulfil.

While the explanation was generated using some criterion (as part of the XAI method), evaluation metrics can be chosen freely.

How do we choose our metric(s)?

This depends on what properties we want our explanations to have. We need to know **what properties a “good” explanation has for our model and use case.**

Let’s look at four commonly used *quantitative* metrics.

Faithfulness / fidelity

Faithfulness / fidelity quantifies how faithful the explanation is to the model's actual behaviour. **Higher is better.**

If changing the input causes the model prediction to change, the explanation should reflect this. Also, if changing the input does not affect the model prediction, the explanation should reflect this as well.

What is a faithful explanation in the case of feature importances?

Faithfulness / fidelity

Faithfulness / fidelity quantifies how faithful the explanation is to the model's actual behaviour. **Higher is better.**

If changing the input causes the model prediction to change, the explanation should reflect this. Also, if changing the input does not affect the model prediction, the explanation should reflect this as well.

In the case of feature importance, a faithful explanation is one where a perturbation of an important feature causes the model prediction to change more, and the perturbation of a less important feature causes the prediction to change little or not at all.

Faithfulness / fidelity

Faithfulness / fidelity quantifies how well the explanation aligns with the model's actual behaviour.

It is often measured through the Prediction Gap on Important features (PGI) and the Prediction Gap on Unimportant features (PGU).

Here, X: input data, e_X : corresponding explanation, *k* number of features, *f*: model, X′: perturbed input,

$$\mathrm{PGI}(X, f, e_X, k) = E_{X' \sim perturb(X, e_X, top-k)}[|f(X) - f(X')|]$$

$$\mathrm{PGU}(X, f, e_X, k) = E_{X' \sim perturb(X, e_X, bottom-k)}[|f(X) - f(X')|]$$

And *perturb* is a perturbation function.

In short: how much prediction changes when top/bottom *k* features are perturbed.

Should these values be high? low?

Faithfulness / fidelity

Faithfulness / fidelity quantifies how well the explanation aligns with the model's actual behaviour.

It is often measured through the Prediction Gap on Important features (PGI) and the Prediction Gap on Unimportant features (PGU).

Here, X: input data, e_X : corresponding explanation, *k* number of features, *f*: model, X′: perturbed input,

$$\mathrm{PGI}(X, f, e_X, k) = E_{X' \sim perturb(X, e_X, top-k)}[|f(X) - f(X^{'})|]$$
$$\mathrm{PGU}(X, f, e_X, k) = E_{X' \sim perturb(X, e_X, bottom-k)}[|f(X) - f(X^{'})|]$$

And *perturb* is a perturbation function.

In short: how much prediction changes when top/bottom *k* features are perturbed.

We want a high prediction gap on important features, and a low one on unimportant features.

Robustness / stability

Robustness / stability measures how much explanations change under small perturbations of the input.

Robust / stable explanations change little or nothing under noise or small adversarial-like perturbations of the model input. The change in explanation can be compared to the actual change in input, the change in model prediction, or other stuff we care about.

What is a robust explanation in the case of feature importances?

Robustness / stability

Robustness / stability measures how much explanations change under small perturbations of the input.

Robust / stable explanations change little or nothing under noise or small adversarial-like perturbations of the model input. The change in explanation can be compared to the actual change in input, the change in model prediction, or other stuff we care about.

In the case of feature importance, a perturbation of the input that doesn't affect the prediction should leave the feature importance order (and preferably magnitudes) ~unchanged.

Robustness / stability

Robustness / stability measures how much explanations change under small perturbations of the input.

This can be calculated using for example Input Stability (RIS) or Relative Output Stability (ROS).

$$\mathrm{RIS}(X, X', e_X, e_{X'}) = \max_{X'} \frac{\left\| \frac{e_X - e_{X'}}{e_X} \right\|_p}{\max\left(\left\| \frac{X - X'}{X} \right\|_p, \epsilon_{\min} \right)},$$

$$\mathrm{ROS}(X, X', e_X, e_{X'}) = \max_{X'} \frac{\left\| \frac{e_X - e_{X'}}{e_X} \right\|_p}{\max\left(\left\| \frac{f(X) - f(X')}{f(X)} \right\|_p, \epsilon_{\min} \right)}$$

How sensitive an explanation is to small changes in the input, relative to…

RIS: how much the input itself actually changed.
ROS: how much the model prediction changed

What does a low RIS represent?
What does a low ROS represent?
Do we want high or low values?

Robustness / stability

Robustness / stability measures how much explanations change under small perturbations of the input.

This can be calculated using for example Input Stability (RIS) or Relative Output Stability (ROS).

$$\mathrm{RIS}(X, X', e_X, e_{X'}) = \max_{X'} \frac{\left\| \frac{e_X - e_{X'}}{e_X} \right\|_p}{\max\left(\left\| \frac{X - X'}{X} \right\|_p, \epsilon_{\min} \right)},$$

$$\mathrm{ROS}(X, X', e_X, e_{X'}) = \max_{X'} \frac{\left\| \frac{e_X - e_{X'}}{e_X} \right\|_p}{\max\left(\left\| \frac{f(X) - f(X')}{f(X)} \right\|_p, \epsilon_{\min} \right)}$$

How sensitive an explanation is to small changes in the input, relative to…

RIS: how much the input itself actually changed.
ROS: how much the model prediction changed

Low RIS → explanations are stable when the input barely changes.
Low ROS → explanations change little unless the model's prediction changes.
We want low values.

Complexity / sparsity

Complexity / sparsity measures how concise explanations are.

Remember the LIME discussion? There, we denoted the surrogate model complexity as $\mathbf{\Omega}(g)$, and a part of the LIME objective was to minimise $\mathbf{\Omega}(g)$.

Still, what exactly complexity means must be defined! For a linear surrogate model, this is often number of covariates (features) or their interactions, and for a tree the number of splits per feature or total depth.

Complexity / sparsity

Complexity / sparsity measures how concise explanations are.

Remember the LIME discussion? There, we denoted the surrogate model complexity as $\mathbf{\Omega}(g)$, and a part of the LIME objective was to minimise $\mathbf{\Omega}(g)$.

Still, what exactly complexity means must be defined! For a linear surrogate model, this is often number of covariates (features) or their interactions, and for a tree the number of splits per feature or total depth.

Counting included features *can* be useful when interpretability actually improves with fewer features in the explanation. However, if many features are actually important, this can make the explanation misleading:

A sparse heatmap is not necessarily better; in a medical setting, pathological regions may not be sparse or clearly bounded.

Axiomatic

Axiomatic refers to whether an explanation methods fulfills properties that are postulated *a priori*.

It can be argued that we should agree on what properties we want explanations should have, and only accept explanation methods that have these properties, or develop methods using these as requirements.

Axiomatic

Axiomatic refers to whether an explanation methods fulfills properties that are postulated *a priori*.

It can be argued that we should agree on what properties we want explanations should have, and only accept explanation methods that have these properties, or develop methods using these as requirements.

For example:

Shapley value based explanations are often motivated by their axiomatic foundation. Some papers speculated the distributive fairness might be necessary to fulfill GDPR requirements (not enacted).

Integrated Gradients - *which we will learn about in the lecture about XAI for CNNs* - was designed guided by the axioms of Sensitivity and Implementation Invariance (Sundararajan et al, 2017).

The Shapley axioms - recap

Dummy: The importance attribution is zero for features that don't contribute.

Symmetry: Features that contribute equally should be attributed equal importance.

Efficiency: The feature attributions sum to what they explain (deviation from mean, predictive power, …)

Additivity: Given several models, the sum of importance attributions of a feature across all models is the same as the feature's attribution in an ensemble of the models.

The Sensitivity and Implementation Invariance axioms

As defined by Sundararajan et al (2017) - in the paper describing an XAI method we will learn -:

Sensitivity: "if for every input and baseline that differ in one feature but have different predictions then the differing feature should be given a non-zero attribution"

in short: *features that alone can cause the model prediction to change must be given non-zero importance.*

Implementation Invariance: "Two networks are functionally equivalent if their outputs are equal for all inputs, despite having very different implementations. Attribution methods should satisfy Implementation Invariance, i.e., the attributions are always identical for two functionally equivalent networks."

in short: *feature attributions should be the same for models that behave the same.* (this is fulfilled trivially by model-agnostic methods)

Summary of four quantitative metrics

Faithfulness aka fidelity quantifies how well the explanation represents the model's actual behaviour.

Robustness aka stability measures the stability of explanations are under perturbations of the input.

Complexity aka sparsity measures how concise explanations are.

Axiomatic refers to whether an explanation methods fulfills properties that are postulated *a priori*.

XAI metrics: function, human and application level

These metrics are examples of function level metrics. They are quantitative, and evaluate to a number that can be compared for different models and data.

Beyond these, if the model is planned as part of an application with users, two more levels are relevant:

- The human level: this involves having laypersons evaluate the explanation and provide feedback or a score. A score is quantitative and can easily be compared across methods. This was done in a 2022 NeurIPS paper, where 1150 Amazon Turk workers were contracted to score various explanations.

- The application level: this involves implementing the explanation into a product and have it tested in a software engineering setting. (The experimental requirements are often more stringent than those for the function and human level.)

[1] What I Cannot Predict, I Do Not Understand: A Human-Centered Evaluation Framework for Explainability Methods

That’s it

Now you know at least four XAI metrics. If you run out of homework, you can pick your favourite dataset, model and explanation method, and evaluate it using the metrics.

Today concludes the lectures on model agnostic explanations.

Next, we start on model-specific explanations for different neural network architectures.

Will the model-agnostic methods we have studied work for these neural networks?

That’s it

Now you know at least four XAI metrics. If you run out of homework, you can pick your favourite dataset, model and explanation method, and evaluate it using the metrics.

Today concludes the lectures on model agnostic explanations.

Next, we start on model-specific explanations for different neural network architectures.

The model-agnostic methods we have studied work regardless of model architecture, also for neural networks. That's what makes them model-agnostic :)

We will extend our toolkit with methods designed for specific architectures, extracting mechanistic signals (like neuron- or layer-level attributions) from the models.

Kthnx :)