Business 4720 - Class 19

Interpretable Machine Learning – Explainable AI

Joerg Evermann

Faculty of Business Administration
Memorial University of Newfoundland
jevermann@mun.ca

MEMORIAL UNIVERSITY

This Class

What You Will Learn:

- Introduction to Interpetability and Explainability
- Model specific and Model agnostic methods
- Global explainability
- Local explainability

MEMORIAL UNIVERSITY

Based On

Molnar, Christoph: *Interpretable Machine Learning* (2023)

https://christophm.github.io/interpretable-ml-book/

(CC BY-NC-SA License)

Gareth James, Daniel Witten, Trevor Hastie and Robert Tibshirani: *An Introduction to Statistical Learning with Applications in R*. 2nd edition, corrected printing, June 2023. (ISLR2)

https://www.statlearning.com

Chapter 8

Kevin P. Murphy: *Probabilistic Machine Learning – An Introduction*. MIT Press 2022.

https://probml.github.io/pml-book/book1.html

Chapter 18

Additional Materials

SciKit Learn

A machine learning framework for Python that also provides some interpretable ML functions.

https://scikit-learn.org/stable/user_guide.html

LIME

A Python package to compute Local Interpretable Model Explanations (a local model-agnostic method).

https://github.com/marcotcr/lime

SHAP

A Python package to compute Shapley Additive Explanations (a local model-agnostic interpretation method).

https://shap.readthedocs.io/en/latest/

Tools

Install required Python packages:

```bash
pip install statsmodels matplotlib scikit-learn \
    PyALE lime shap
```

Importance of Interpretability

Human understanding of how the AI works and arrives at its results (decisions, predictions, . . .)

- Curiosity
- Human learning
- Human sensemaking of events and phenomena
- Knowledge extraction for scientific progress
- Safety and compliance assessment
- Reliability and robustness evaluation
- Identify knowledge limits
- Auditability
- Bias detection & ensuring fairness
- Trust and acceptance
- Debugging & failure analysis
- Legal obligations ("right to explanation")

Model Interpretability

Distinctions

- Intrinsic $\leftrightarrow$ Post-hoc
- Local $\leftrightarrow$ Global

MEMORIAL UNIVERSITY

Intrinsically Interpretable Models

Algorithm	Linear	Monotone	Interaction
Linear regression	Yes	Yes	No
Logistic regression	No	Yes	No
Decision trees	No	Some	Yes
RuleFit	Yes	No	Yes
Naive Bayes	No	Yes	No
k-NN	No	No	No

Source:

`https://christophm.github.io/interpretable-ml-book/simple.html`

Linear Regression

Using R:

```r
# Load the bike rental data set
d <- read.csv('https://evermann.ca/busi4720/bike.csv')
# Perform the regression and summarize results
summary(lm(cnt~season+temp, data=d))
```

Results:

```text
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    3151.02     169.35  18.606  < 2e-16 ***
seasonSPRING   -494.15     163.28  -3.026  0.00256 **
seasonSUMMER   -852.68     209.82  -4.064 5.35e-05 ***
seasonWINTER -1342.87     164.59  -8.159 1.49e-15 ***
temp            132.79      11.02  12.046  < 2e-16 ***
---
Residual standard error: 1433 on 726 degrees of freedom
Multiple R-squared: 0.4558, Adjusted R-squared: 0.4528
```

Linear Regression

- **Algorithmic transparency**: The ordinary least squares loss function is clear and intuitive; provides optimality guarantees
- **Coefficients** β
  - An increase of one unit of a predictor increases the prediction by β, *assuming all other predictors remain the same* ("ceteris paribus")
  - Switching from the reference category (see "contrasts") to another category increases the prediction by β, *assuming all other predictors remain the same* ("ceteris paribus")
  - **Intercept** is the predicted value when all other predictors are 0. Is this reasonable?
- R^2 is the amount of explained variance; model weights should only be interpreted when R^2 reasonable size.
- **Relative feature importance** is given by the $t = \frac{\hat{\beta}}{SE(\hat{\beta})}$ statistic.

Linear Regression

Dimension reduction to improve interpretability:

- Manual feature selection, e.g. based on effect size
- Automatic feature selection (forwards or backwards)
- Regression with PCA components
- Penalized regression with LASSO

Be aware of bias-variance trade-off with all of these.

Decision Trees

Decision Tree Types

- Regression trees
- Classification trees

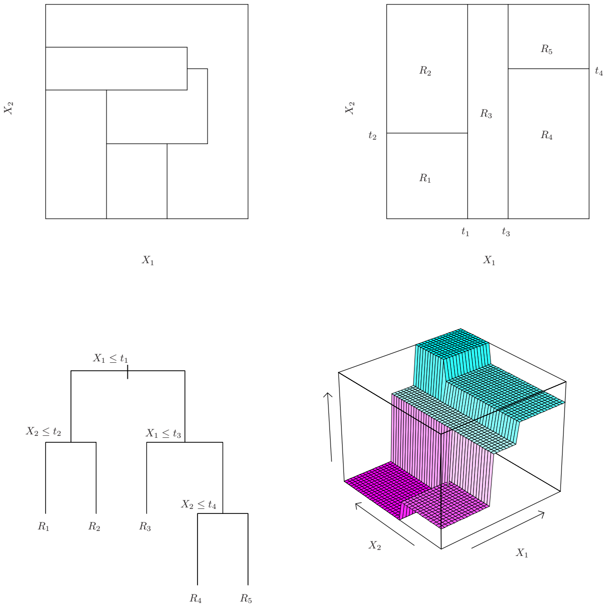
Strengths

- Intrinsically interpretable and visualizable
- Individual predictions explained by path through tree
- Captures feature interactions
- No need to transform features

Weaknesses

- Unstable (high variance)
- Tend to overfit
- Predictions are piecewise constant

Regression Trees



Source: ISLR2
Figure 8.3

Regression Trees

1. Recursively divide the predictor space into J distinct and non-overlapping regions $R_1, R_2, \ldots, R_j$
   - For every predictor j and split point s define regions

     $$R_1(j,s) = \{X | X_j < s\} \quad \text{and} \quad R_2(j,s) = \{X | X_j \geq s\}$$

   - Choose j and s to minimize variance in each region:

     $$\sum_{i:x_i \in R_1(j,s)} (y_i - \bar{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \bar{y}_{R_2})^2$$

2. For every observation that falls into region R_j, prediction is the mean of the targets of training observations in R_j

Regression Trees – Python

Prepare data:

```python
import matplotlib.pyplot as plt
import pandas as pd
d=pd.read_csv('https://evermann.ca/busi4720/bike.csv')
x=d[['temp', 'hum']]
y=d['cnt']
```

Fit unpruned tree:

```python
from sklearn.tree import DecisionTreeRegressor
regr = DecisionTreeRegressor()
regr.fit(x, y)
```

Print the MSE:

```python
from sklearn.metrics import mean_squared_error
mean_squared_error(regr.predict(x), y)
```

Print the tree:

```python
from sklearn.tree import export_text
print (export_text(regr,feature_names=x.columns))
```

Early stopping can prevent overfitting and maintain interpretability.

Maximum depth:

Python
```
regr = DecisionTreeRegressor(max_depth=3).fit(x, y)
```

Minimum samples per leaf:

Python
```
regr = DecisionTreeRegressor(min_samples_leaf=10).fit(x, y)
```

Maximum number of leaf nodes:

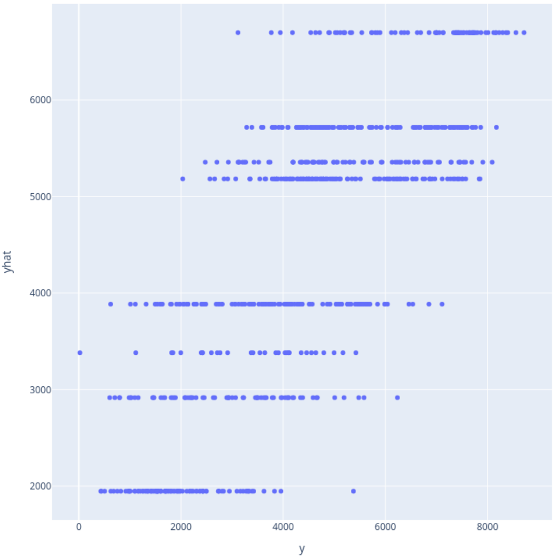
Python
```
regr = DecisionTreeRegressor(max_leaf_nodes=8).fit(x, y)
```

Plot fitted versus true values:

Python
```
import plotly.express as px
px.scatter(
    pd.DataFrame([y, regr.predict(x)], index=['y', 'yhat']) \
    .transpose(),
    x='y', y='yhat').show()
```

MEMORIAL UNIVERSITY

Regression Trees – Python



Hands-On Exercises

1. Fit regression trees to the *entire bike rental dataset* on the previous slides. Calculate the MSE as you vary:
   - **max_depth**: choose values 1, 3, 5, 7
   - **min_samples_leaf**: choose values 1, 5, 10, 20
   - **max_leaf_nodes**: choose values 2, 8, 16, 32
2. Split the data into a test and training data set, like this:

Python
```python
train=df.sample(frac=0.8)
test=df.drop(train.index)
```

3. Repeat exercise (1) by fitting the tree to the *training data* and calculate the MSE for the *test data*.

Classification Trees

- Proportion of observations of class k in leaf node m:

$$p_{km}$$

- Predict the most common (majority) class in a leaf node:

$$k(m) = \operatorname*{argmax}_k p_{km}$$

- Probability of choosing an item with label i in node m is

$$p_{im}$$

- Probability of misclassifying that item in node m is

$$1 - p_{im} = \sum_{k \neq i} p_{km}$$

Classification Trees [cont'd]

- Use **Gini impurity** G (node purity) for node m to determine splits:

$$
\begin{aligned}
G_m = \sum_{i=1}^{J} p_{im}(1 - p_{im}) = \sum_{i=1}^{J}(p_{im} - p_{im}^2) = \sum_{i-1}^{J} - \sum_{i=1}^{J} p_{im}^2 \\
= 1 - \sum_{i=1}^{J} p_{im}^2
\end{aligned}
$$

- Use **Cross-Entropy** H for node m to determine splits.

$$
H_m = -\sum_{i=1}^{J} p_{im} \log p_{im}
$$

Classification Trees [cont'd]

Stock market data (from R ISLR2 package):

Python
```python
d=pd.read_csv('https://evermann.ca/busi4720/Smarket.csv')
x=d[['Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5', 'Volume']]
y=d['Direction']
```

Fit a classification tree:

Python
```python
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(max_depth=5)
clf.fit(x, y)
```

Classifier has similar tree control options to regressor.

Training accuracy:

Python
```python
from sklearn.metrics import accuracy_score
accuracy_score(clf.predict(x), y)
```

MEMORIAL UNIVERSITY

Hands-On Exercises

1. Fit classification trees to the *entire stock market dataset* on the previous slides. Calculate the accuracy as you vary:
   - **max_depth**: choose values 1, 3, 5, 7
   - **min_samples_leaf**: choose values 1, 5, 10, 20
   - **max_leaf_nodes**: choose values 2, 8, 16, 32
2. Split the data into a test and training data set, like this:

Python
```python
train = d.iloc[:3*d.shape[0]//4,:]
test = d.iloc[3*d.shape[0]//4:,:]
```

3. Repeat exercise (1) by fitting the tree to the *training data* and calculate the accuracy for the *test data*.

Tree Ensemble Methods

Problem: Trees are unstable, have high variance, prone to overfitting

Solution: Fit multiple trees

Aggregation:

$$f(y|x) = \frac{1}{|M|} \sum_{m \in M} f_m(y|x)$$

- Average prediction for regression trees
- Majority vote for classification models

MEMORIAL UNIVERSITY

Stacking:

$$f(y|x) = \frac{1}{|M|} \sum_{m \in M} w_m f_m(y|x)$$

- Train weights w_m using separate data set

Tree Ensemble Methods [cont'd]

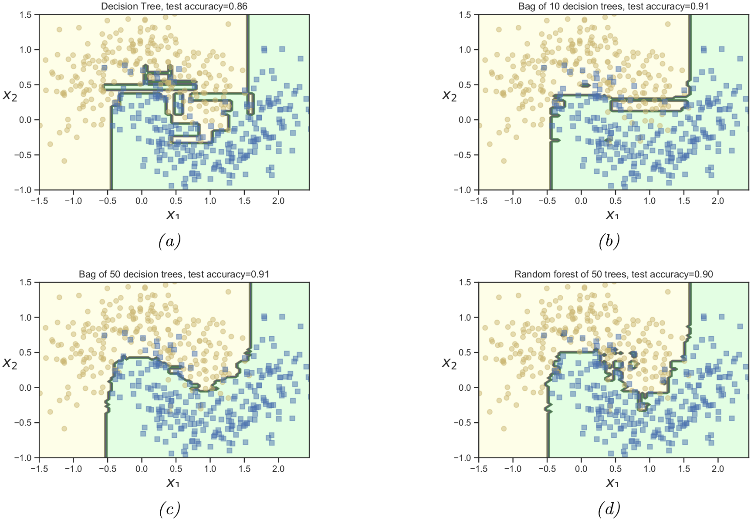
Bagging:

- "Bootstrap Aggregating"
- Randomly sample data with replacement (bootstrapping)
- OOB ("out-of-bag") error serves as test error

Random Forests:

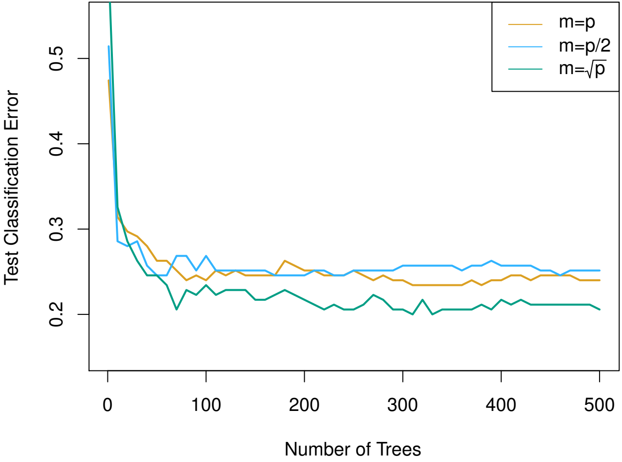
- "Decorrelate" trees
- At each split, only consider a random sample of m predictors as split candidates
- Typically $m \approx \sqrt{p}$

Tree Ensemble Methods [cont'd]



Source: Murphy Figure 18.4

Example: Impact of *m* on test error:



Source: ISLR 2, Figure 8.10

Bagging in Python:

```python
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bagging = BaggingClassifier(DecisionTreeClassifier(max_depth=3),
    n_estimators=20,
    max_samples=2/3,
    max_features=2/3)
bagging.fit(x, y)
accuracy_score(bagging.predict(x), y)
```

Random forest regression in Python:

Python
```
from sklearn.ensemble import RandomForestRegressor

d=pd.read_csv('https://evermann.ca/busi4720/bike.csv')
x=d[['temp', 'hum']]
y=d['cnt']
```

Python
```
regr = RandomForestRegressor(n_estimators=10,
                             max_features="sqrt")
regr = regr.fit(x, y)
mean_squared_error(regr.predict(x), y)
```

Random forest classification in Python:

Python
```
from sklearn.ensemble import RandomForestClassifier

d=pd.read_csv('https://evermann.ca/busi4720/Smarket.csv')
x=d[['Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5', 'Volume']]
y=d['Direction']
```

Python
```
clf = RandomForestClassifier(n_estimators=10,
                             max_features="sqrt")
clf = clf.fit(x, y)
accuracy_score(clf.predict(x), y)
```

Hands-On Exercise

- Using the random forest regressor and classifier in Python, vary the number of estimators and the maximum number of features to determine their impact on the *training performance* (MSE or accuracy).
- Split the data into a test and training data set, like this:

Python
```
train=df.sample(frac=0.8)
test=df.drop(train.index)
```

- Using the random forest regressor and classifier in Python, vary the number of estimators and the maximum number of features to determine their impact on the *test performance* (MSE or accuracy).

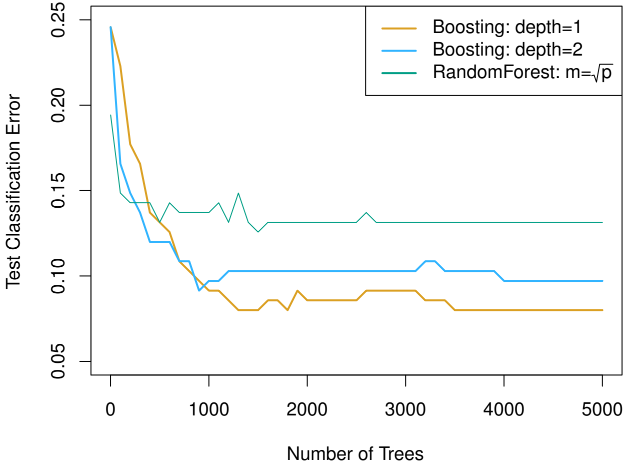
MEMORIAL UNIVERSITY

Boosting:

- Iteratively build B simple ("weak") trees using residuals

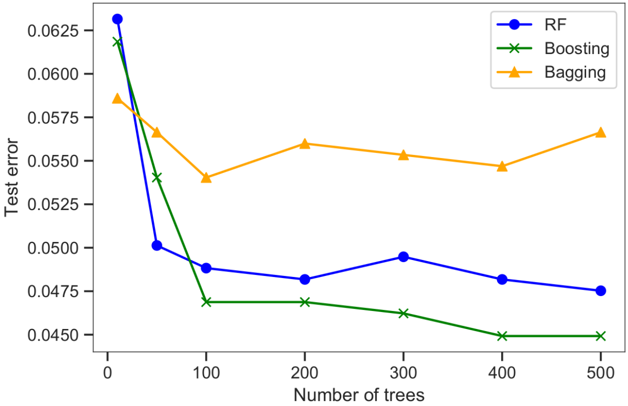
1. Set $\hat{f}(x) = 0$; $r_i = y_i$
2. For $b = 1, 2, \ldots, B$, repeat:
   (a) Fit tree $\hat{f}^b$ with d splits to data (X, r). Typically, $d = 1$ or 2.
   (b) Update $\hat{f}$ by adding a "shrunken" version of the new tree. Typically, $\lambda = 0.01$ or $\lambda = 0.001$.
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$
   (c) Update the residuals
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$
3. Output the boosted model:
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

Example: Impact of *d* on test error:



Source: ISLR 2, Figure 8.11

Example: Comparison of Bagging, Random Forest, and Boosting:



Source: Murphy Figure 18.5

Boosting classification in Python:

```python
from sklearn.ensemble import GradientBoostingClassifier

d=pd.read_csv('https://evermann.ca/busi4720/Smarket.csv')
x=d[['Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5', 'Volume']]
y=d['Direction']
```

```python
clf = GradientBoostingClassifier(
         learning_rate=0.01,
         n_estimators=10,
         max_depth=3)

clf = clf.fit(x, y)
accuracy_score(clf.predict(x), y)
```

Hands-On Exercise

- Using the gradient boosting regressor and classifier in Python, vary the learning rate and the maximum tree depth to determine their impact on the *training performance* (MSE or accuracy).
- Split the data into a test and training data set, like this:

Python
```python
train=df.sample(frac=0.8)
test=df.drop(train.index)
```

- Using the gradient boosting regressor and classifier in Python, vary the learning rate and the maximum tree depth to determine their impact on the *test performance* (MSE or accuracy).

MEMORIAL UNIVERSITY

Further reading:

https://scikit-learn.org/stable/modules/tree.html

https://scikit-learn.org/stable/modules/ensemble.html#

https://scikit-learn.org/stable/auto_examples/tree/plot_unveil_tree_structure.html

https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html

Global Model Agnostic Methods

- **Partial dependence plot (PDP)**
- **Individual conditional expectation (ICE) curves**
- **Accumulated local effects plot (ALE)**
- Feature interaction
- Functional decomposition
- **Permutation feature importance**
- **Global surrogate models**
- Prototypes

Partial Dependence Plot (PDP)

Marginal effect of one (or a few) features X_S on the outcome, marginalized (i.e. sum weighted by probability) over all other (complement) features X_C.

$$\hat{f}_S(X_S) = \mathbb{E}_{X_C}\left[\hat{f}(X_S, X_C))\right] = \int \hat{f}(X_S, X_C)p(X_C)dX_c$$

Estimated from sample data as:

$$\hat{f}_S(X_S) = \frac{1}{n}\sum_{i=1}^{n} \hat{f}(X_S, X_C^{(i)})$$

Shows how the *average* prediction changes when the focal predictor is changed (assuming feature independence).

Partial Dependence Plot (PDP)

Read the data set:

```
Python
import pandas as pd
d=pd.read_csv('https://evermann.ca/busi4720/bike.csv')
x=d[['temp', 'hum']]
y=d[['cnt']]
```

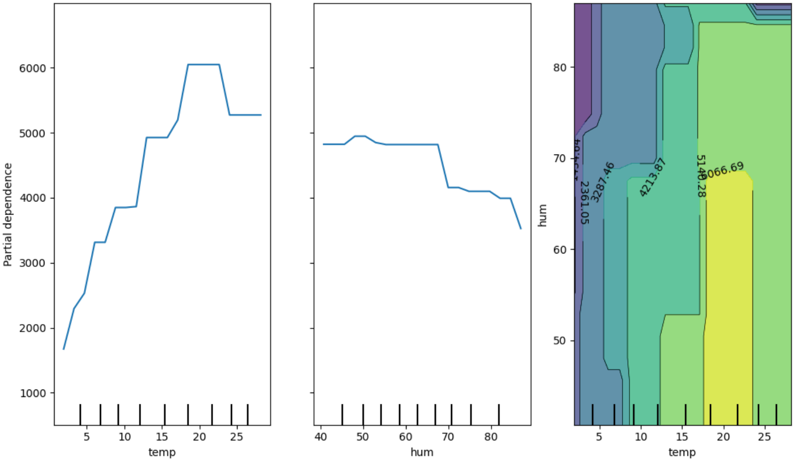
Fit a regression tree:

```
Python
from sklearn.tree import DecisionTreeRegressor
regr = DecisionTreeRegressor(max_depth=5).fit(x, y)
```

Show the PDP:

```
Python
import matplotlib.pyplot as plt
from sklearn.inspection import PartialDependenceDisplay
PartialDependenceDisplay \
    .from_estimator(regr, x, [0, 1, (0,1)],
        grid_resolution=20)
plt.show()
```

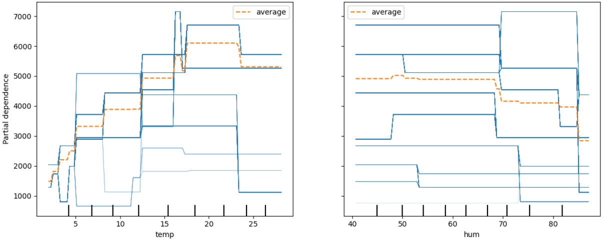
Partial Dependence Plot (PDP)



Individual Conditional Expectation (ICE) Plot

- Instead of the average effect of a feature, shows PDP for individual samples
- Identify individual **outlier** cases or **heterogeneous data**

Python
```
PartialDependenceDisplay \
    .from_estimator(regr, x, [0, 1], kind='both')
```



(Individual samples overlaid due to piece-wise constant regression)

Accumulated Local Effects (ALE) Plot

- Effects computed for a grid of intervals (a ”local window”) (instead of the entire domain, as in PDP)
- Does not construct unrealistic feature combinations
- Overcomes the problem of correlated features in PDP
- Focuses on difference in predictions

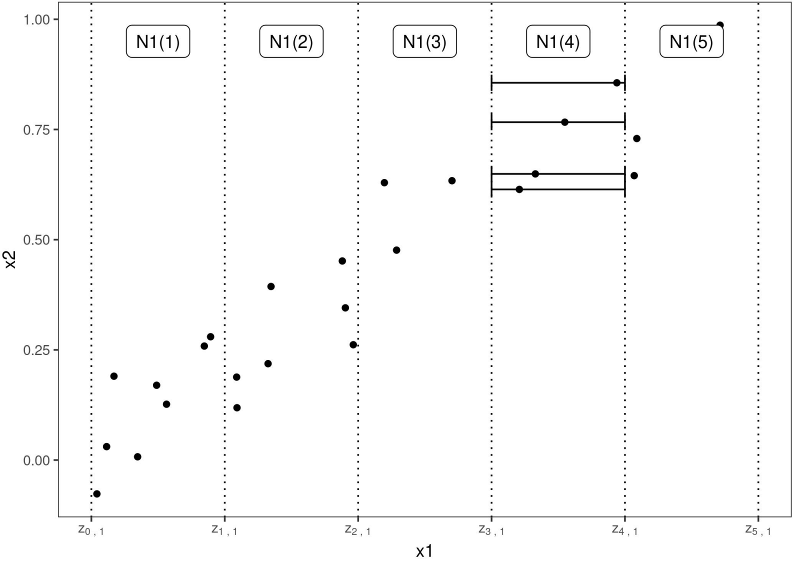
$$\hat{\tilde{f}}_{j,ALE}(X) = \sum_{k=1}^{k_j(x)} \frac{1}{n_j(k)} \sum_{i:x_j^{(i)} \in N_j(k)} \left[\hat{f}(z_{k,j}, x_j^{(i)}) - \hat{f}(z_{k-1,j}, x_j^{(i)})\right]$$

- Difference of predictions (in sq brackets) is *local* to ”neighbourhood” $N_j(k)$ of feature j around observation k
- Outer sum *accumulates* the local effects

Centering the effects to mean 0:

$$\hat{f}_{j,ALE}(x) = \hat{\tilde{f}}_{j,ALE}(x) - \frac{1}{n}\sum_{i=1}^{n} \hat{\tilde{f}}_{j,ALE}(x_j^{(i)})$$

ALE Plots



Source: Molnar, Fig. 8.7

Accumulated Local Effects (ALE) Plot

Train model:

```python
from sklearn.tree import DecisionTreeRegressor
regr=DecisionTreeRegressor(min_samples_leaf=10).fit(x,y)
```

Construct the ALE and plot:

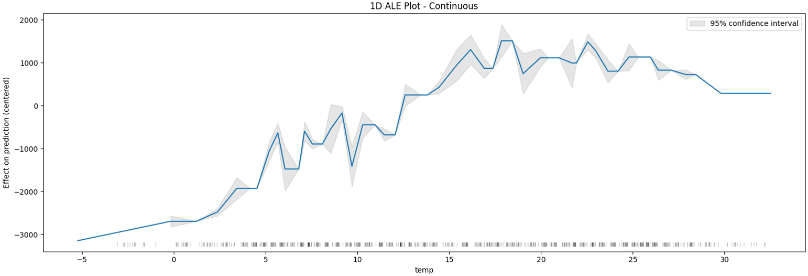
```python
import matplotlib.pyplot as plt
from PyALE import ale
ale_effects = ale(X=x, model=regr, \
    feature=['temp'], grid_size=50, include_CI=True)
plt.show()
```

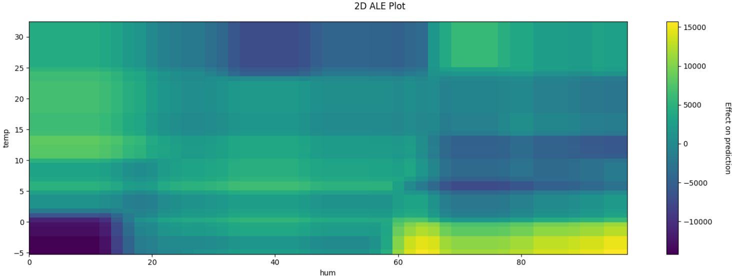
2D feature interactions:

```python
ale_effects = ale(X=x, model=regr, \
    feature=['temp', 'hum'], grid_size=50)
plt.show()
```

MEMORIAL UNIVERSITY

Accumulated Local Effects (ALE) Plot





UNIVERSITY

Permutation Feature Importance

Intuition

Calculate the increase in a model's prediction error when permuting a feature

1. Estimate model error on original data $e^{\text{orig}} = L(y, \hat{f}(X))$
2. For each feature j:
   - For each repetition k in $1 \cdots K$:
     - Generate $X_{j,k}^{\text{perm}}$ by permuting ("randomly shuffling") values of feature j
     - Estimate $e_{j,k}^{\text{perm}} = L(y, \hat{f}(X_{j,k}^{\text{perm}}))$
   - Calculate permutation feature importance as $i_j = e^{\text{orig}} - \frac{1}{K}\sum_k^K e_{j,k}^{\text{perm}}$

Calculate Permutation Feature Importance on *test* data

Permutation Feature Importance

Prepare data:

```python
import pandas as pd
d=pd.read_csv('https://evermann.ca/busi4720/bike.csv')
x=pd.get_dummies(d.drop(['yr','days_since_2011'],axis=1))
y=x.pop('cnt')
```

Train model:

```python
from sklearn.tree import DecisionTreeRegressor
regr=DecisionTreeRegressor(min_samples_leaf=10).fit(x,y)
```

Calculate permutation feature importance and sort them:

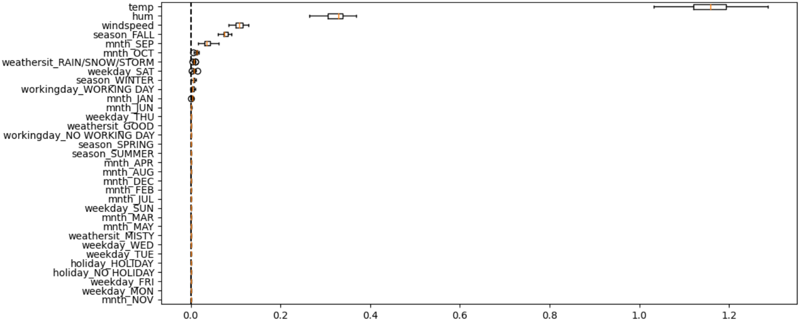
```python
from sklearn.inspection import permutation_importance
r = permutation_importance(regr, x, y, n_repeats=30)
r_idx = r.importances_mean.argsort()
```

Permutation Feature Importance

Produce a nice plot of sorted feature importance:

```python
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.boxplot(
    r.importances[r_idx].T,
    vert=False,
    labels=x.columns[r_idx])
ax.axvline(x=0, color="k", linestyle="--")
plt.show()
```

Permutation Feature Importance



Uncertainty due to multiple permutations (parameter `n_repeats`)

Global Surrogate Models

Intuition

Predict the predictions of a complex ”black box” model using an intrinsically interpretable model.

Example ”black box” model:

```python
from sklearn.neural_network import MLPRegressor
regr = MLPRegressor((4, 2,), max_iter=10000)
regr.fit(x, y)
preds = regr.predict(x)
```

Interpretable, linear model to explain predictions:

```python
from statsmodels.api import OLS
OLS(preds, x.astype(float)).fit().summary()
```

Global Model Agnostic Methods – Summary

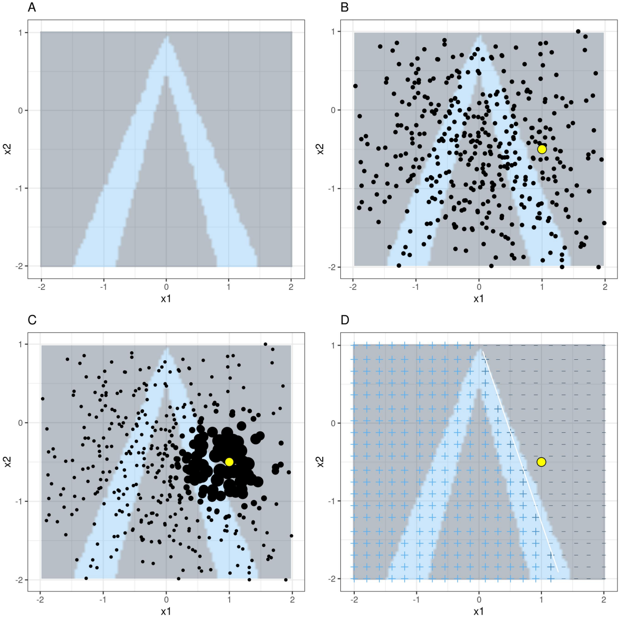
PDP/ICE	
Intuitive Clear interpretation Easy to implement	Limited number of features Assumes feature independence
ALE	
Unbiased for correlated features Clear interpretation Faster to compute than PDP	Local interpretation only ALE may differ from linear coefficients No ICE curves Unstable for large number of intervals
PFI	
Clear interpretation Concise, global measure Does not require retraining Takes into account all interactions	Linked to model error Requires access to true targets May be biased for correlated features
Global Surrogate Models	
Flexible Intuitive R-squared measure for fit	Conclusions about model, not data Unclear cut-off for goodness of fit

Local Interpretable Model-Agnostic Explanations

Idea

1. Choose an instance x of interest
2. Perturb data by turning on or off features i using randomized feature combination vector of $z_i \in [0, 1]$
3. Sample perturbed instances around x, weighted by kernel π_g,
4. Fit each perturbed instance using black-box model $f(z_i)$
5. Train local interpretable model on features z_i, targets $f(z_i)$ and weights $\pi_x(z_i)$

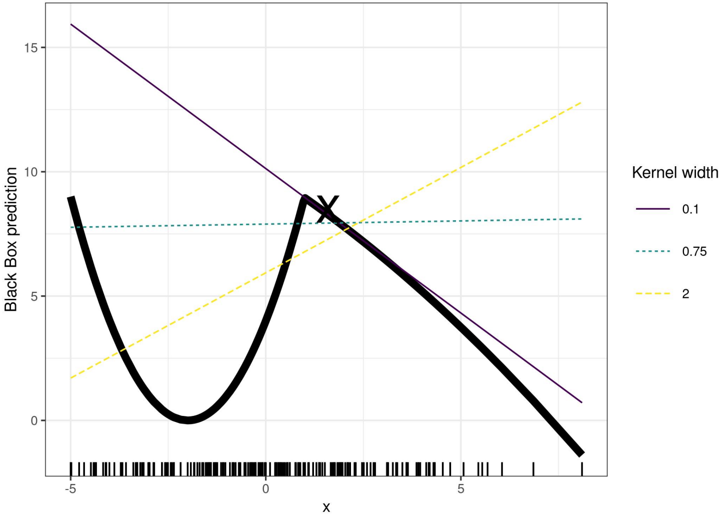
LIME – Example



Source: Molnar Figure 9.5

LIME – Example

- Weight function π is often an exponential smoothing kernel
- Kernel width is critical determinant of explanation



Source: Molnar Figure 9.6

LIME – Example

Using a deep decision tree as ”black box”:

Python
```python
import sklearn.tree
dt = sklearn.tree.DecisionTreeClassifier(max_depth=8)
dt.fit(x, y)
```

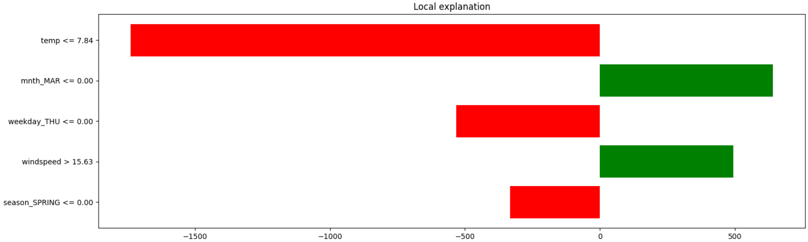
Create the explainer:

Python
```python
import lime, lime.lime_tabular
from sklearn.linear_model import Ridge

explainer = lime.lime_tabular.LimeTabularExplainer(
    x.to_numpy(),
    feature_names=x.columns,
    discretize_continuous = True,
    mode='regression',
    verbose=True)
```

Explain instance number 5:

Python
```
exp = explainer.explain_instance(
    x.to_numpy()[7],
    dt.predict,
    num_features=5,
    num_samples=1000,
    distance_metric='euclidean')
exp.as_list()
exp.as_pyplot_figure().show()
```



LIME for Images

LIME explanations for label "bagel" and "strawberries":

Molnar, Figure 9.8

Python Examples:

https://github.com/marcotcr/lime

Paper:

https://arxiv.org/abs/1602.04938

Shapley Values

Motivation

How much does *feature value* x_j contribute to the overall prediction compared to the average prediction?

Game Theory

- Players cooperate in a coalition and receive a certain profit from this cooperation.
- Method for assigning payouts to players depending on their contribution to the total payout.

Shapley Values

$$\phi_i(v) = \frac{1}{n} \sum_{S \subseteq N \setminus \{i\}} \binom{n-1}{|S|}^{-1} [v(S \cup \{i\}) - v(S)]$$

- $v(S \cup \{i\}) - v(S)$: marginal contribution of player i to coalition of players S
- $\binom{n-1}{|S|}$: number of possible ways to form a coalition of size $|S|$ of the set $N \setminus \{i\}$ of $n-1$ players (set N without player i)

Fairness Properties

- **Efficiency**: Contributions add up to total value
- **Symmetry**: If two players contribute equally to all possible coalitions, they have the same Shapley value
- **Dummy**: A player that does not contribute at all has a Shapley value of 0
- **Additivity**: For a game with combined payouts $v + w$, the Shapley values of players are $\phi^{(v)} + \phi^{(w)}$

Shapley Values in Interpretable ML

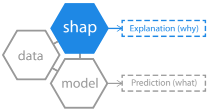
- Players are feature values
- Coalitions are combinations of feature values
- Presence in a coalition means we know the value
- Absence from a coalition means we don't know the value $\Rightarrow$ integrate/marginalize over all values of all features not in coalition S

$$v_x(S) = \int \cdots \int_{\mathbb{R}} \hat{f}(x_1, \ldots, x_p) d\mathbb{P}_{x \notin S} - E_x(\hat{f}(X))$$

- Expensive to compute $\Rightarrow$ in practice, approximation by sampling and permuting values (can make for unrealistic instances when features are correlated)

MEMORIAL UNIVERSITY

Shapley Additive Explanations (SHAP)



Paper

https://arxiv.org/abs/1705.07874

Documentation (Intro and Examples)

https://shap.readthedocs.io/en/latest/index.html

Python Code and Tutorials

https://github.com/shap/shap

MEMORIAL UNIVERSITY

SHAP Example

Fit a simple regression model to the California housing dataset:

```python
import sklearn
import shap

X, y = shap.datasets.california(n_points=1000)
model = sklearn.linear_model.LinearRegression()
model.fit(X, y)
```

Compute the SHAP values:

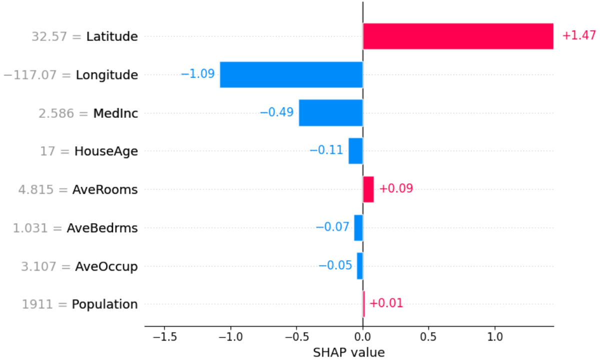
```python
X100 = shap.utils.sample(X, 100)
explainer = shap.Explainer(model.predict, X100)
shap_values = explainer(X)
```

MEMORIAL UNIVERSITY

SHAP Example

The **barplot** shows the importance of feature values for an individual prediction:

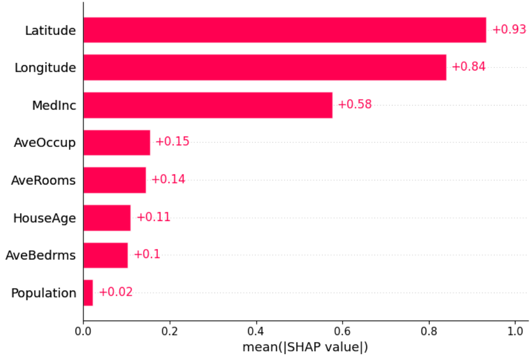
```python
shap.plots.bar(shap_values[20])
```



SHAP Example

The **barplot** can also show the importance of a feature by averaging over all instances (and their feature values):

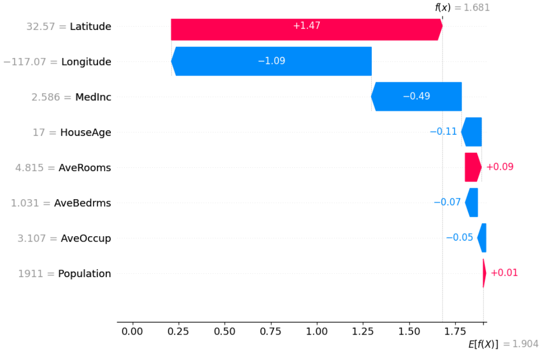
```python
shap.plots.bar(shap_values)
```



SHAP Example

Waterfall plots explain how feature values combine to produce an individual prediction:

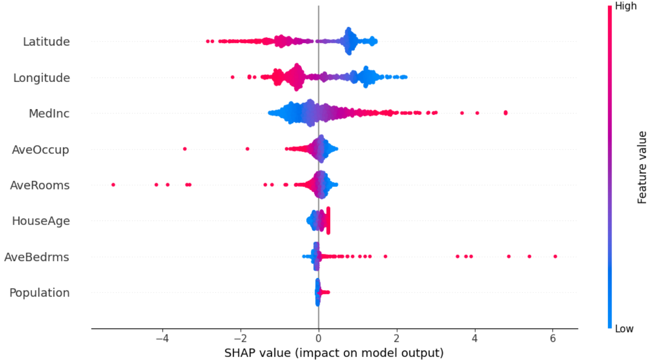
Python
```
sha.plots.waterfall(shap_values[20], max_display=14)
```



SHAP Example

Beeswarm plots explain all feature values for all instances (represented by a dot):

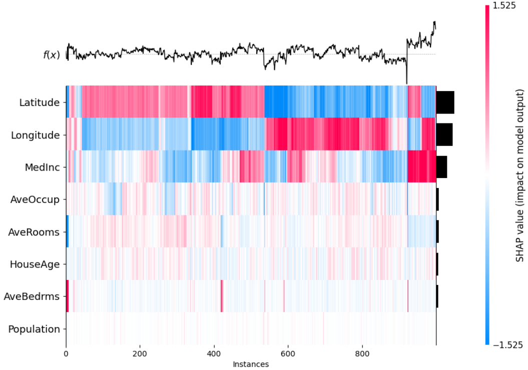
Python
```
shap.plots.beeswarm(shap_values)
```



SHAP Example

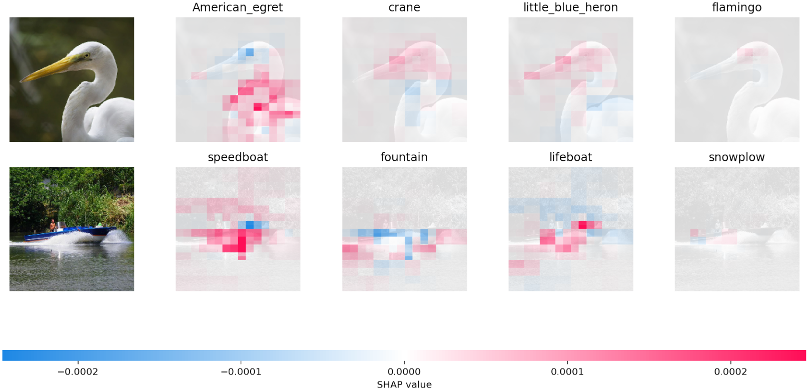
The **heatmap** shows SHAP values of feature values for all instances, and shows model prediction and global feature importance in rugs:

Python
```
shap.plots.heatmap(shap_values)
```



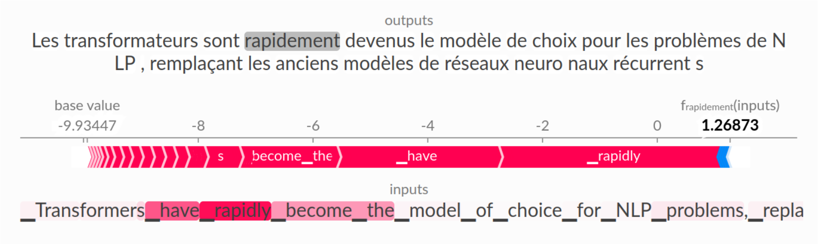
SHAP for Image Classification

- Presence/absence of features/pixels by masking parts of an image:



Source: `https://github.com/shap` (MIT License)

SHAP for Text Classification



Source: `https://shap.readthedocs.io/en/latest/text_examples.html` (MIT License)

Fixes for Matplotlib

Bash

```
sudo apt-get install -y qt6-base-dev xcb libxcb-cursor-dev
sudo ln -sf /usr/lib/x86_64-linux-gnu/qt6/plugins/platforms/ \
    /usr/bin/
pip install pyqt6
```