Multidimensional Scaling, Data Visualization and Explainable AI

Tomasz Walkowiak
Mateusz Gniewkowski

Wroclaw University of Science and Technology
CLARIN-PL
tomasz.walkowiak@pwr.edu.pl

CLARIN

CLARIN-PL
Common Language Resources and Technology Infrastructure

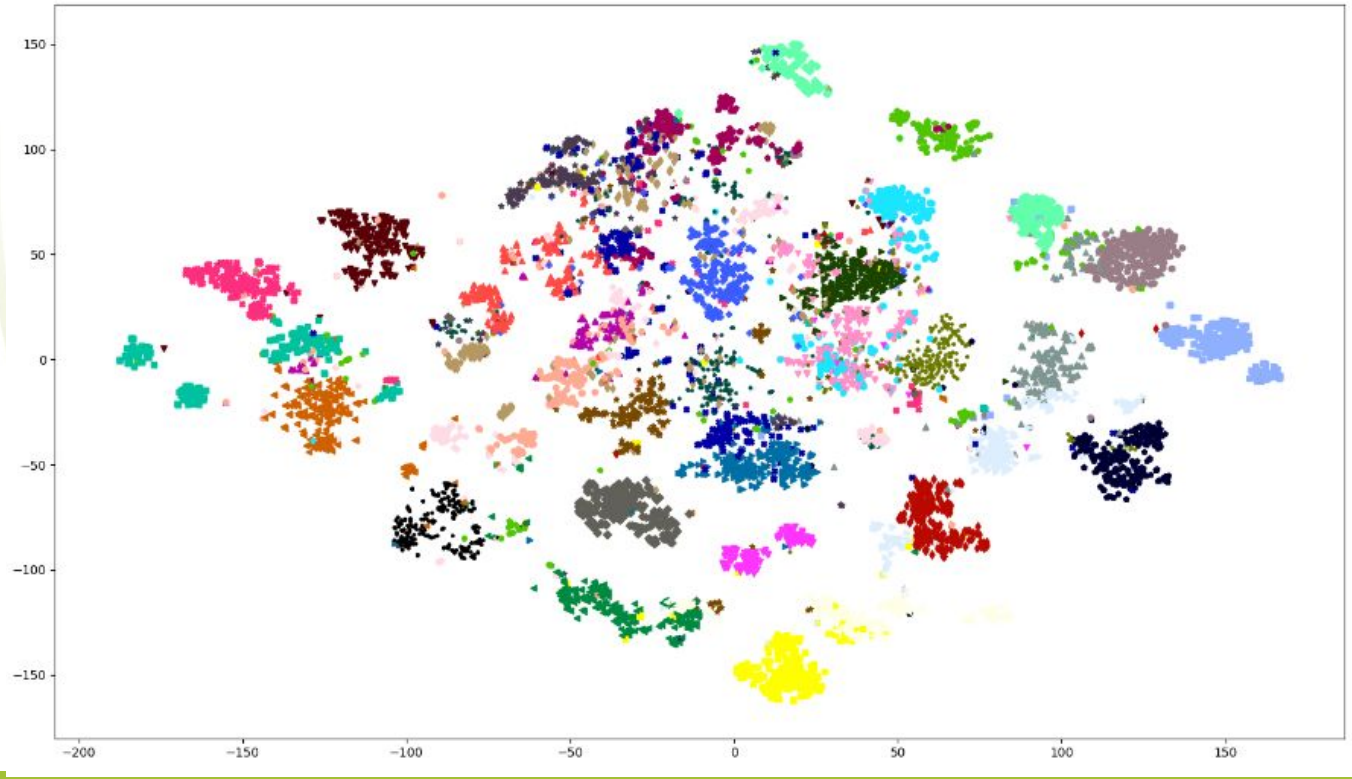
Wrocław University of Science and Technology

Visualization

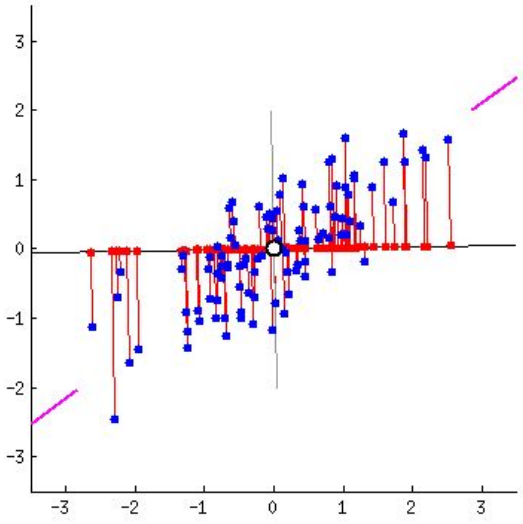
- The goal is to visualize high-dimensional spaces in two or three dimensions
- Dimensionality reduction, manifold learning, multidimensional scaling
  - X_{nxp} n - number of samples, p-original dimensionality
  - Y_{nxk} k<<p
- Will involve compromises
  - not all „relations" between points (observed objects) will be preserved
  - some "features" of the original space are preserved (distance, similarity, neighborhood, …)
  - minimizing some loss function
- Visualization can give a feel for what is in your data or how (deep) models work
- There are many visualization techniques
  - https://scikit-learn.org/stable/modules/manifold.html
  - we will focus only on: PCA, UMAP

Example

- Points - text documents from Wikipedia
- Selected 34 classes – colors
- Vectors in original space – averaged fastText embeddings



PCA, Classical (Torgerson) MDS



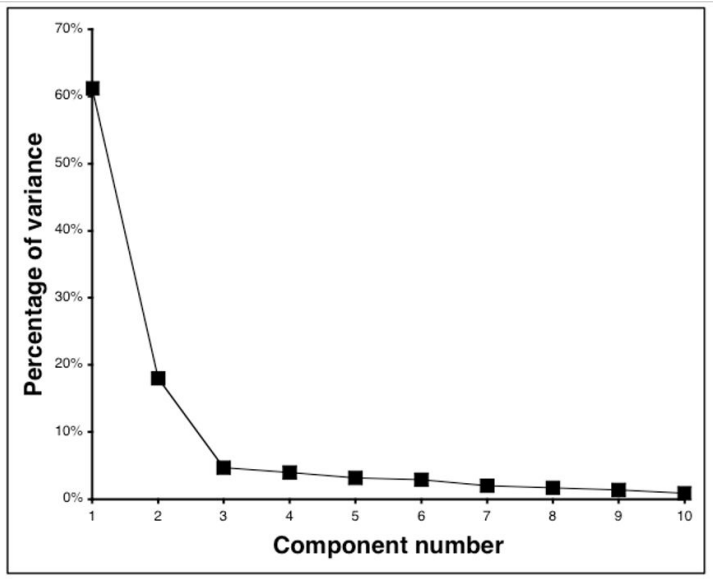
- Linear orthogonal, projection of centered vectors
- PCA finds the directions of maximum variance in high-dimensional data
- projects it onto a new subspace with equal or fewer dimensions than the original one
- Procedure:
  1. Calculate the correlation matrix of centered vectors $X^T X$
  2. Calculate the eigenvalues and eigenvectors
  3. Order eigenvectors by eigenvalues
  4. The resulting data is a linear combination of original (centered) vectors and k-th eigenvectors

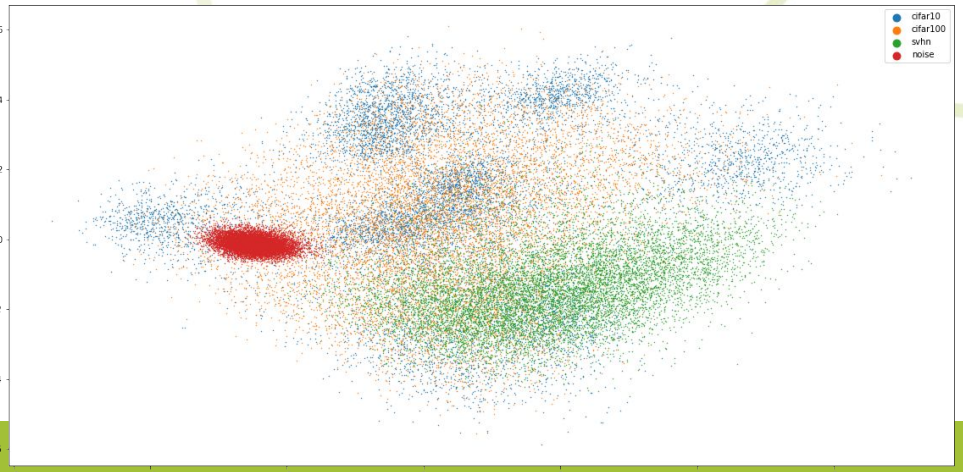
PCA in practice

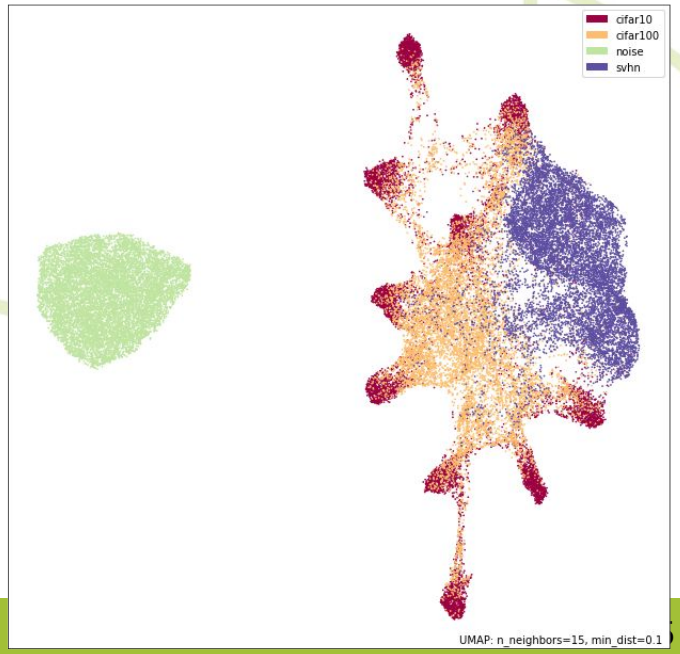
- sklearn.decomposition.PCA

```
pca = PCA(n_components=2)
pca.fit(X)
```
Centering is done by default
```
print(pca.explained_variance_ratio_)
X_2=pca.transform(X)
```

- Remarks
  - good interpretation
  - relatively fast
  - visualization are … often poor



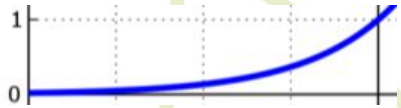




Uniform Manifold Approximation and Projection for Dimension Reduction

- UMAP (2018)
  - https://umap-learn.readthedocs.io/en/latest/
  - input parameter: a number of neighbors, *k*
- High-dimensional data represented as a weighted k-nearest neighbor graph
- Weights of the edges of the graph (for k-nearest neighbors)

$$w((x_i, x_{i_j})) = \exp\left(\frac{-\max(0, d(x_i, x_{i_j}) - \rho_i)}{\sigma_i}\right)$$



- Where "shape" parameters (delta, rho):

$$\rho_i = \min\{d(x_i, x_{i_j}) \mid 1 \le j \le k, d(x_i, x_{i_j}) > 0\}$$

$$\sum_{j=1}^{k} \exp\left(\frac{-\max(0, d(x_i, x_{i_j}) - \rho_i)}{\sigma_i}\right) = \log_2(k)$$

- Convert distances to similarity
- And keeps them in low dimension

UMAP (2)

force-directed

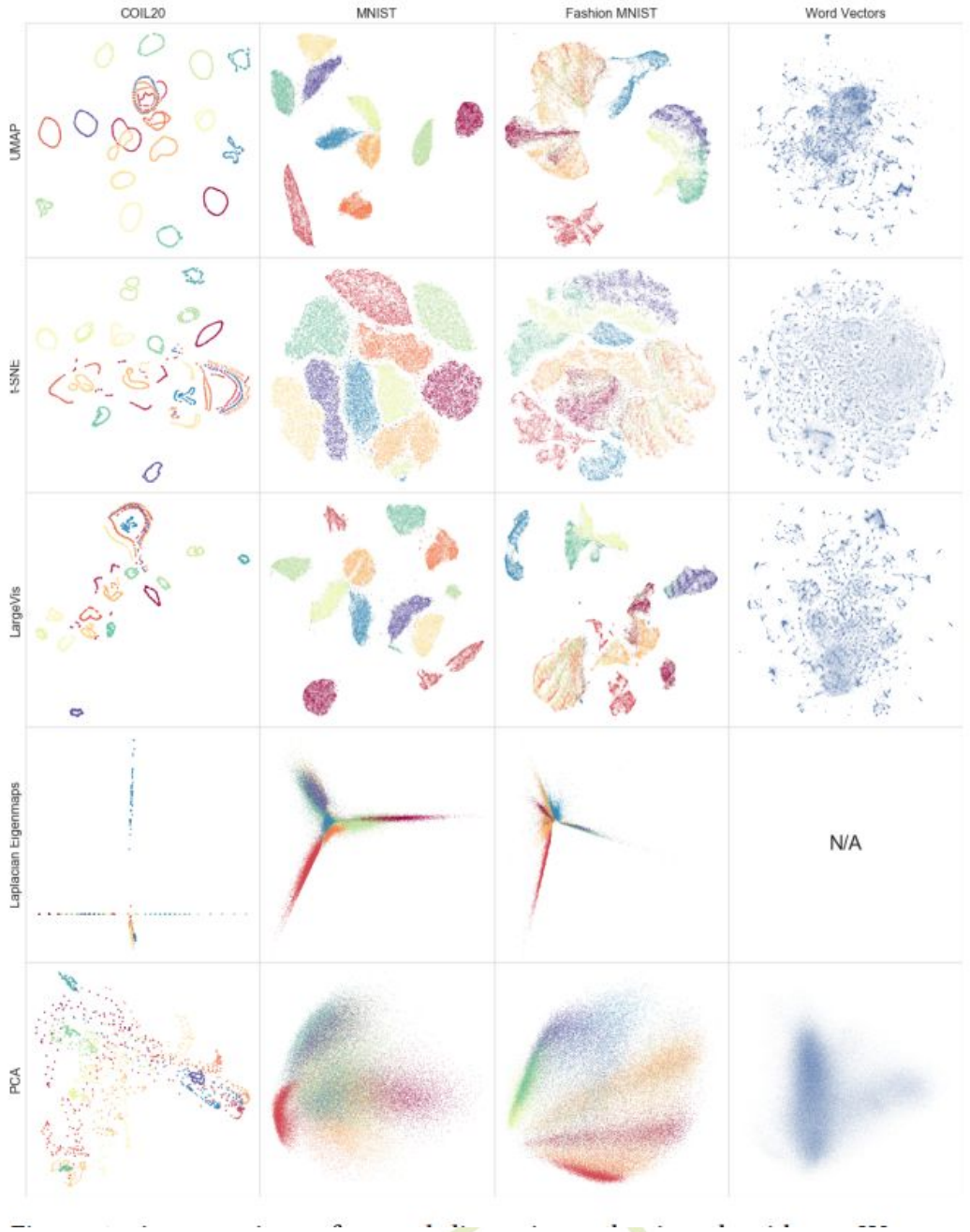
- We draw the graph in 2D (or any other dimension) with the force-directed algorithm (being physical simulations)
- attraction $$\frac{-2ab\|\mathbf{y_i}-\mathbf{y_j}\|_2^{2(b-1)}}{1+\|\mathbf{y_i}-\mathbf{y_j}\|_2^2} w((x_i,x_j))\,(\mathbf{y_i}-\mathbf{y_j})$$
- repulsion $$\frac{2b}{(\epsilon+\|\mathbf{y_i}-\mathbf{y_j}\|_2^2)\,(1+a\|\mathbf{y_i}-\mathbf{y_j}\|_2^{2b})}\,(1-w((x_i,x_j)))\,(\mathbf{y_i}-\mathbf{y_j})$$
- The starting point is the spectral embedding method
  - eigenvectors of a normalized graph Laplacian of the sparse affinity matrix
- a, b - hyper-parameters

Example:

```
import umap
fit = umap.UMAP()
u = fit.fit_transform(data)
```

Remarks

- Python implementation
- Good code and tutorials
- Much faster than t-SNE
- Could work as a transformation, and with class knowledge
- Consider data standardization (z-scores)
  - not needed for BERT
- Resulting chart analysis requires caution



Explainable AI

why my classifier thinks it is something, something, darkside

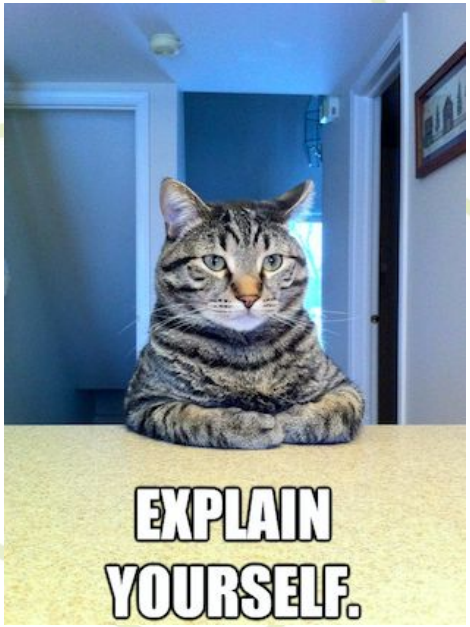


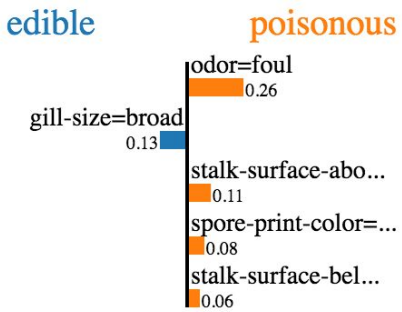
image source: https://luminovo.ai/blog-posts/ai-and-the-question-of-explainability

(a) Original Image

(b) Explaining *Electric guitar*

(c) Explaining *Acoustic guitar*

(d) Explaining *Labrador*



Feature	Value
odor=foul	True
gill-size=broad	True
stalk-surface-above-ring=silky	True
spore-print-color=chocolate	True
stalk-surface-below-ring=silky	True

““Why Should I Trust You?”; https://arxiv.org/pdf/1602.04938.pdf

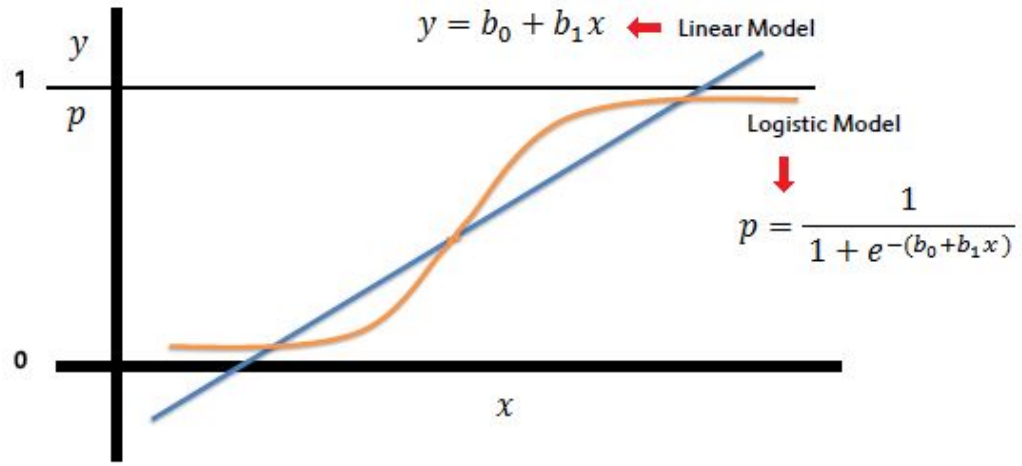
Methods (described in this presentation)

1. Visualisation (covered)
2. Model-specific
   a. Explaining DT
   b. Explaining LR
3. Model-agnostic
   a. LIME
   b. SHAP

Sources:
1 (great book). https://christophm.github.io/interpretable-ml-book/
2. (LIME paper, also great): https://arxiv.org/abs/1602.04938
3. (SHAP paper, again, great): https://arxiv.org/abs/1705.07874

Logistic regression

1. In Linear regression we can interpret coefitniets as they are
2. In Logistic regression coefitniets represents log odds



$$ln\left(\frac{P(y=1)}{1-P(y=1)}\right) = log\left(\frac{P(y=1)}{P(y=0)}\right) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$$

$$\frac{odds_{x_j+1}}{oddsx_j} = \frac{exp\left(\beta_0 + \beta_1 x_1 + \ldots + \beta_j(x_j+1) + \ldots + \beta_p x_p\right)}{exp\left(\beta_0 + \beta_1 x_1 + \ldots + \beta_j x_j + \ldots + \beta_p x_p\right)}$$

$$\frac{exp(a)}{exp(b)} = exp(a-b)$$

$$\frac{P(y=1)}{1-P(y=1)} = odds = exp\left(\beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p\right)$$

$$\frac{odds_{x_j+1}}{oddsx_j} = exp\left(\beta_j(x_j+1) - \beta_j x_j\right) = exp\left(\beta_j\right)$$

image source: https://saedsayad.com/logistic_regression.htm

DIY

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression


data = load_iris()
X = data.data
y = data.target

clf = LogisticRegression(random_state=0).fit(X, y)
pd.DataFrame(
    np.exp(clf.coef_),
    index=data.target_names,
    columns=data.feature_names
)
```

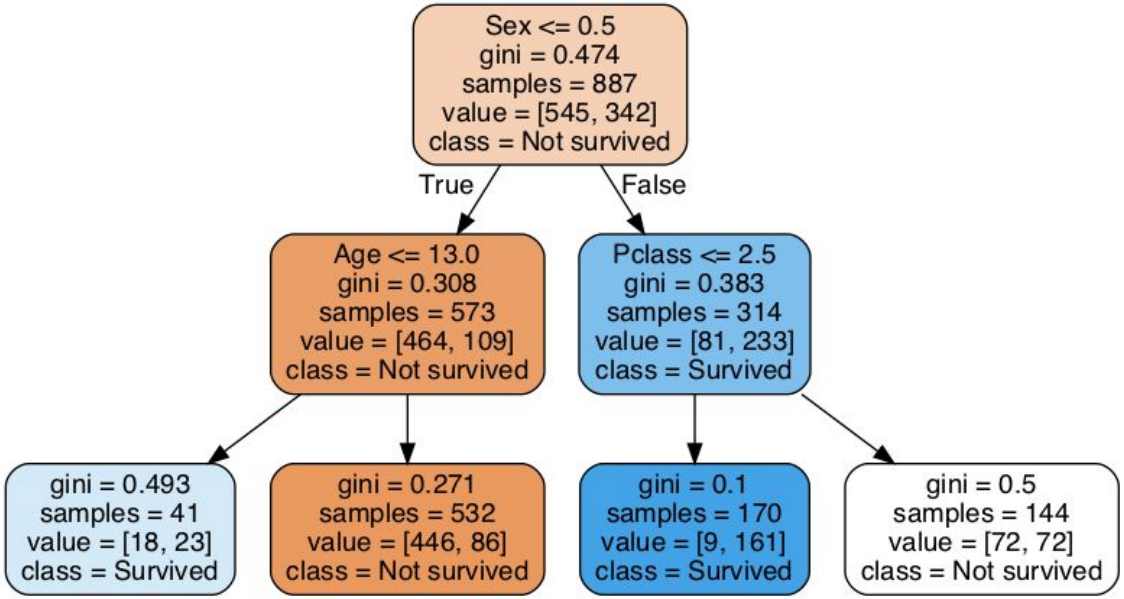
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
setosa	0.657784	2.630322	0.080385	0.338172
versicolor	1.701099	0.729940	0.818559	0.387319
virginica	0.893691	0.520839	15.197687	7.634717

If only petal_width increases by one, the probability that the iris is “virginica” is 7.63 times larger than it is not.

Decision Tree

Feature importance:

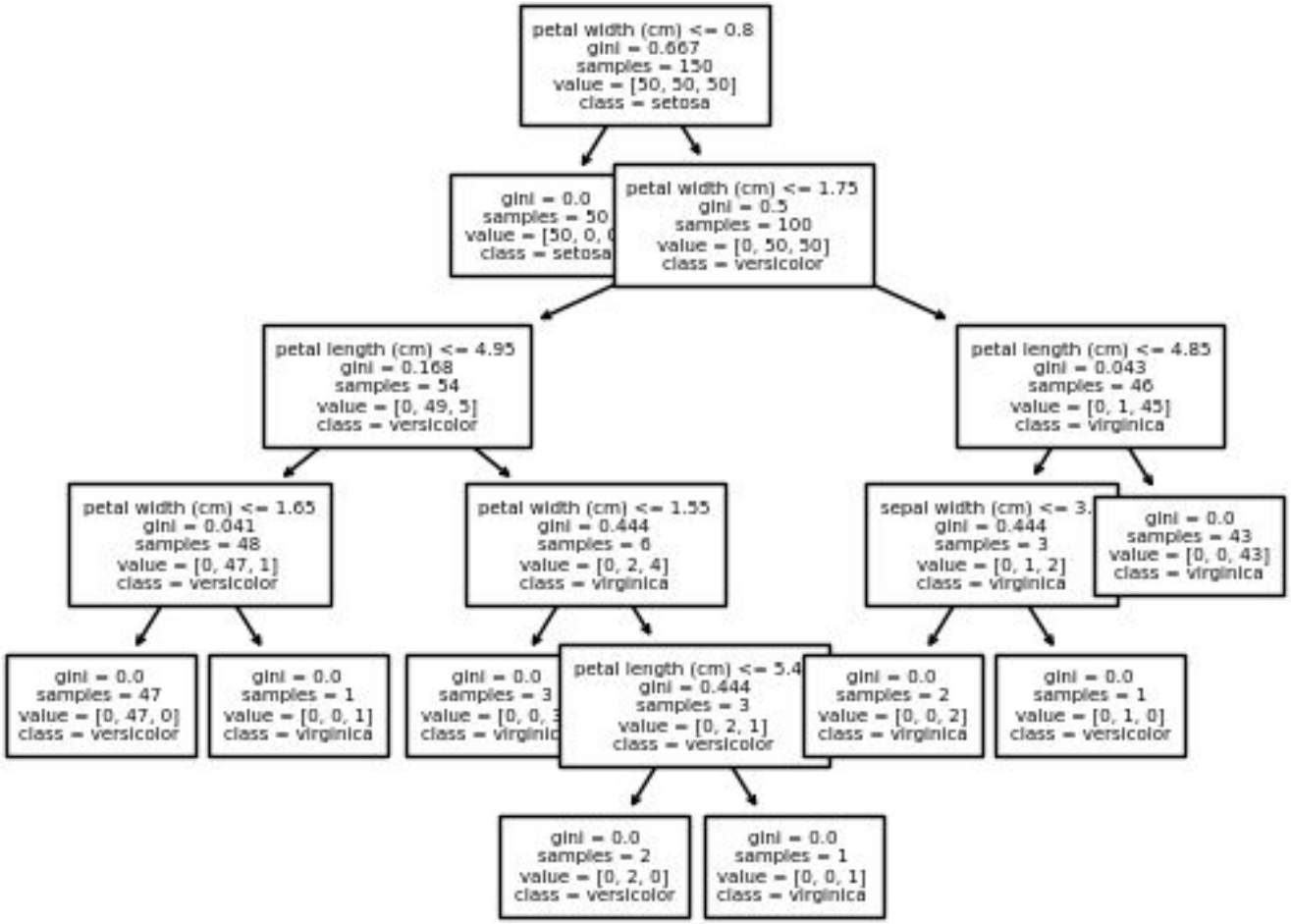
1. Measure how the variance (or Gini index) is reduced in given split (compared to a parent node)
2. Sum all the importances
3. Scale the obtained values



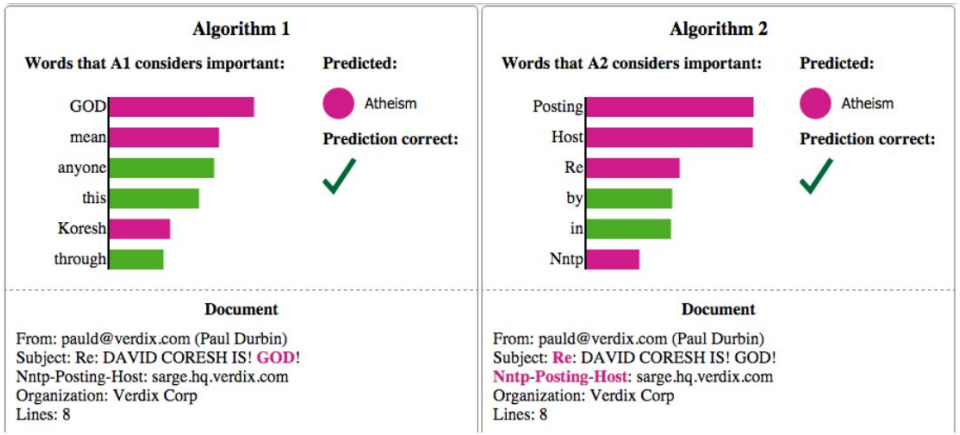
DIY

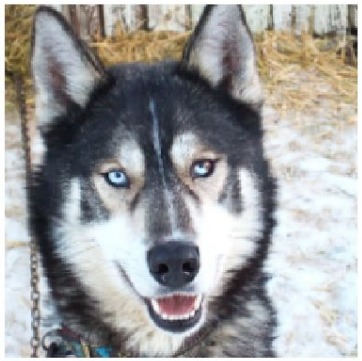
```
import pandas as pd
import numpy as np

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

data = load_iris()

X = data.data
y = data.target

data.feature_names
data.target_names

clf = DecisionTreeClassifier(
    random_state=0
).fit(X, y)
pd.DataFrame(
    clf.feature_importances_,
    index=data.feature_names
)
```

	Importance
sepal length (cm)	0.000000
sepal width (cm)	0.013333
petal length (cm)	0.064056
petal width (cm)	0.922611


petal width (cm) <= 0.8
gini = 0.667
samples = 150
value = [50, 50, 50]
class = setosa
gini = 0.0
samples = 50
value = [50, 0, 0]
class = setosa
petal width (cm) <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
class = versicolor
petal length (cm) <= 4.95
gini = 0.168
samples = 54
value = [0, 49, 5]
class = versicolor
petal length (cm) <= 4.85
gini = 0.043
samples = 46
value = [0, 1, 45]
class = virginica
petal width (cm) <= 1.65
gini = 0.041
samples = 48
value = [0, 47, 1]
class = versicolor
petal width (cm) <= 1.55
gini = 0.444
samples = 6
value = [0, 2, 4]
class = virginica
sepal width (cm) <= 3.
gini = 0.444
samples = 3
value = [0, 1, 2]
class = virginica
gini = 0.0
samples = 43
value = [0, 0, 43]
class = virginica
gini = 0.0
samples = 47
value = [0, 47, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
petal length (cm) <= 5.4
gini = 0.444
samples = 3
value = [0, 2, 1]
class = versicolor
gini = 0.0
samples = 2
value = [0, 0, 2]
class = virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor
gini = 0.0
samples = 2
value = [0, 2, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica

LIME (local interpretable model-agnostic explanations)



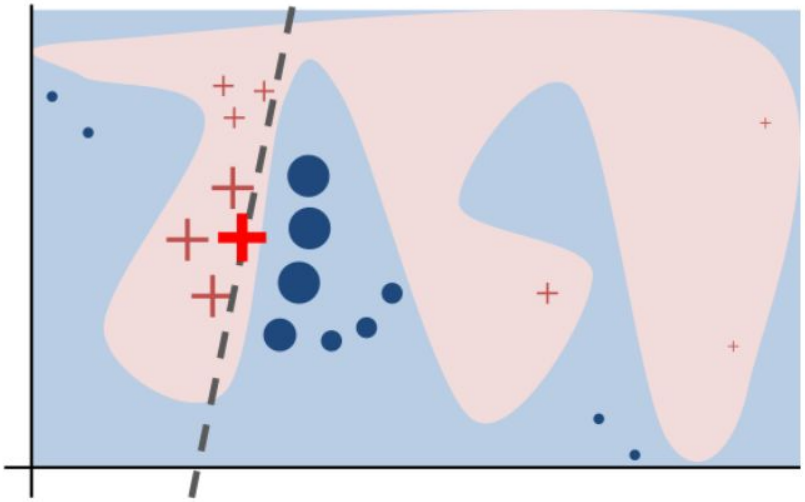


(a) Husky classified as wolf

(b) Explanation

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \ \mathcal{L}(f, g, \pi_x) + \Omega(g)$$

$$\mathcal{L}(f, g, \pi_x) = \sum_{z, z' \in \mathcal{Z}} \pi_x(z) \left(f(z) - g(z')\right)^2$$



- Select your instance of interest for which you want to have an explanation of its black box prediction.
- Perturb your dataset and get the black box predictions for these new points.
- **Weight the new samples according to their proximity to the instance of interest.**
- Train a weighted, interpretable model on the dataset with the variations.
- Explain the prediction by interpreting the local model.

DIY

Official repository, tons of examples: https://github.com/marcotcr/lime

I know, I am lazy

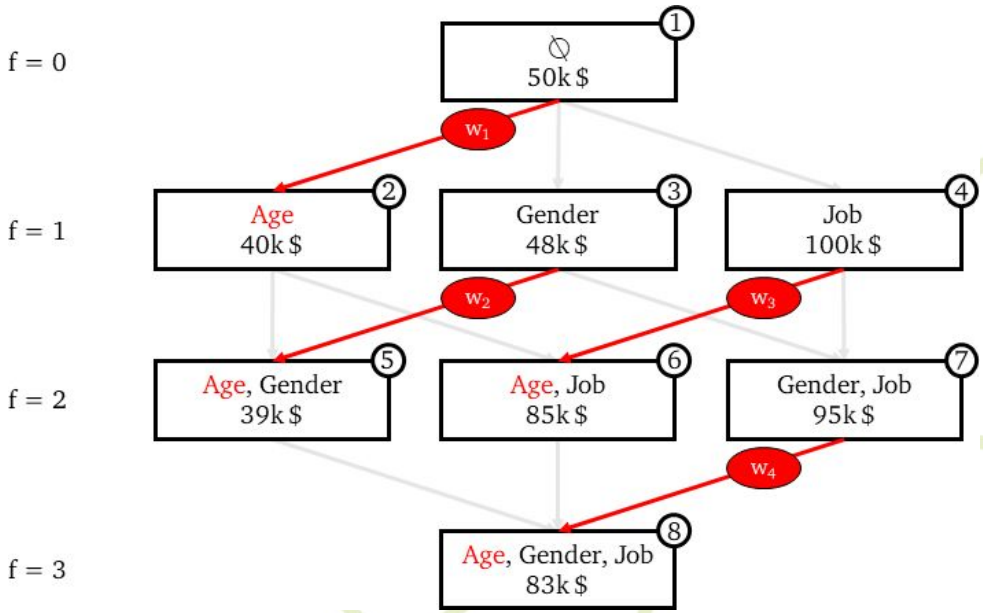
SHAP(ley values)

Those comes from the game-theory approach, where:

1. The “game” is to reproduce output of a model (single sample)
2. The “players” are features the model knows

SHAP(ley values)

$$SHAP_{Age}(x_0) = w_1 \times MC_{Age,\{Age\}}(x_0) + w_2 \times MC_{Age,\{Age,Gender\}}(x_0) + w_3 \times MC_{Age,\{Age,Job\}}(x_0) + w_4 \times MC_{Age,\{Age,Gender,Job\}}(x_0)$$



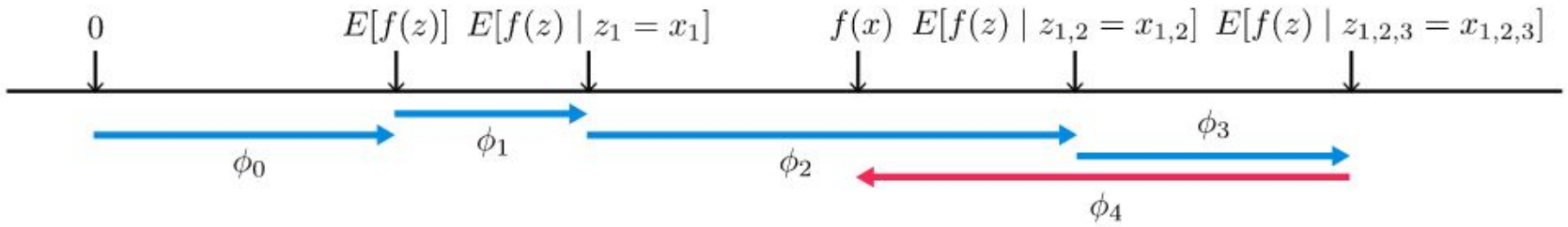
$$SHAP_{Age}(x_0) = -11.33k\$$$

MC - marginal contribution

https://towardsdatascience.com/shap-explained-the-way-i-wish-someone-explained-it-to-me-ab81cc69ef30

SHAP

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j$$



2^|features| sets - that is a lot!

We need better computer!

We do not have one, nobody has!

We need some approximation!

Here you go

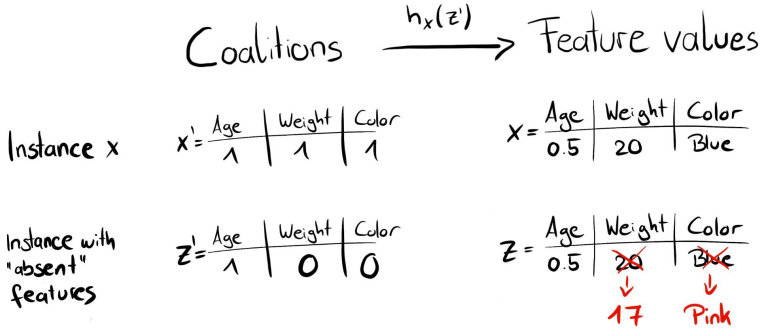
Kernel SHAP (Linear LIME + Shapley values)

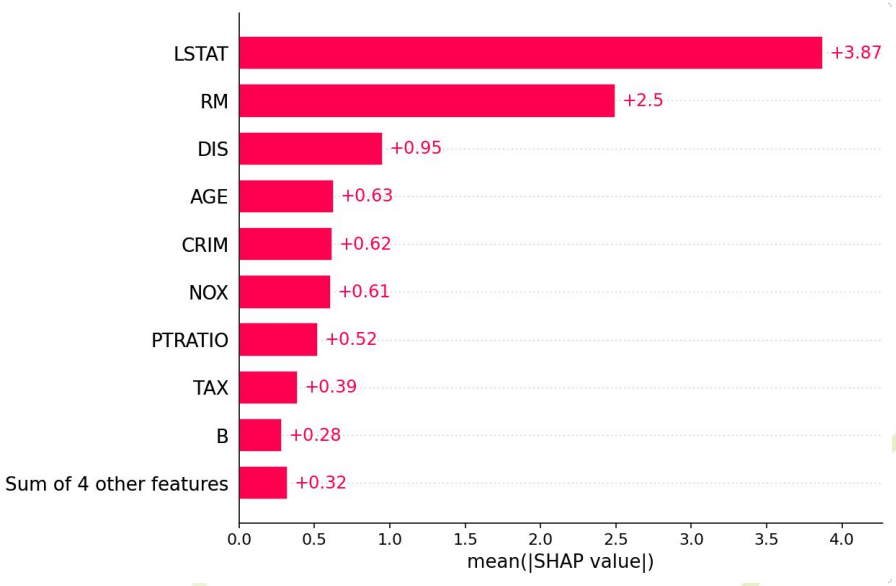
1. Sample coalitions (1 = feature present in coalition, 0 = feature absent). $z'_k \in \{0,1\}^M, \quad k \in \{1,\ldots,K\}$
2. Get prediction for each by first converting to the original feature space and then applying model
3. **Compute the weight for each z' with the SHAP kernel**
4. Fit weighted linear model.
5. Return Shapley values ϕ_k the coefficients from the linear model.

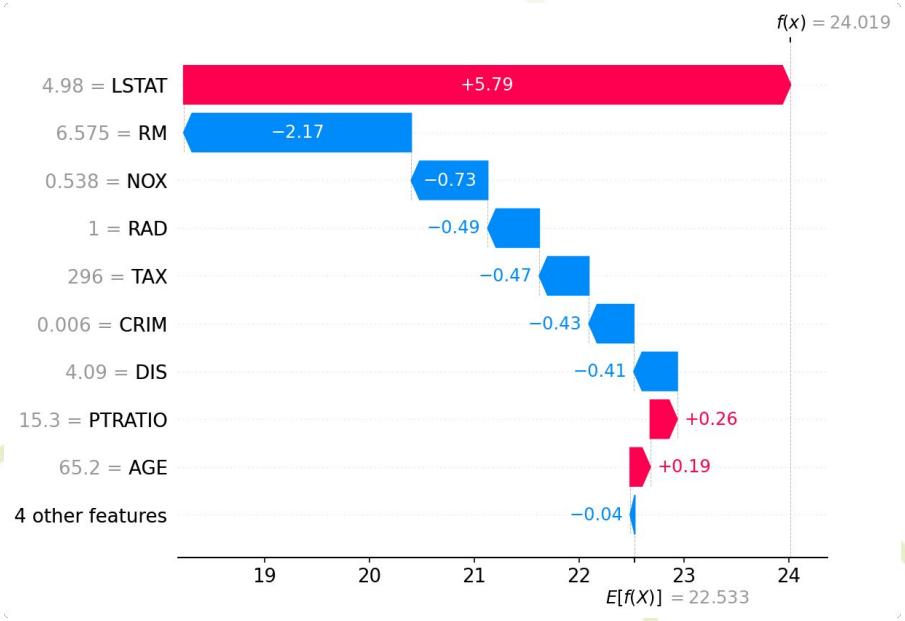
$$\Omega(g) = 0,$$

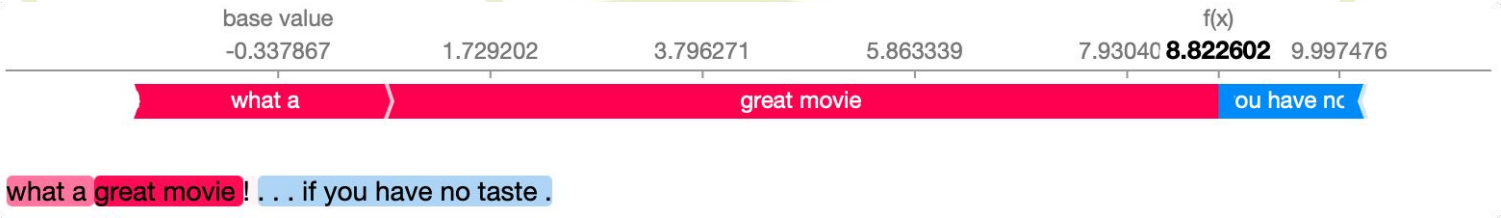
$$\pi_{x'}(z') = \frac{(M-1)}{(M \text{ choose } |z'|)|z'|(M-|z'|)},$$

$$L(f,g,\pi_{x'}) = \sum_{z' \in Z} \left[f(h_x(z')) - g(z')\right]^2 \pi_{x'}(z'),$$









what a great movie ! . . . if you have no taste .

DIY

Docs (with examples): https://shap.readthedocs.io/en/latest/

AND

We will use SHAP on Friday :)

Questions?

1. What about texts?
2. What about global explanation?
3. …!?
4. ?
5. …