# Building a Robot Judge: Data Science for Decision-Making

9. Encoders and Explanations

# Q&A Page

https://padlet.com/eash44/ji7ehio1eksrm9o0

# IV is powerful but depends on quality of instruments

1. Relevance
   - Always report the first stage (F-test above 10)

# IV is powerful but depends on quality of instruments

1. Relevance
   - Always report the first stage (F-test above 10)
2. Exogeneity
   - Conditional exogeneity requires "no $Z$-confounders"
     - equivalent to having no unobserved confounders in the reduced form (RF) $Y \sim Z$
     - more precise to say **"no RF-confounders"**.

# IV is powerful but depends on quality of instruments

1. Relevance
   - Always report the first stage (F-test above 10)
2. Exogeneity
   - Conditional exogeneity requires "no $Z$-confounders"
     - equivalent to having no unobserved confounders in the reduced form (RF) $Y \sim Z$
     - more precise to say **"no RF-confounders"**.
   - is it plausible? could instrument be endogenous to outcome?
   - Check if instrument is correlated with predetermined variables

# IV is powerful but depends on quality of instruments

1. Relevance
   - Always report the first stage (F-test above 10)
2. Exogeneity
   - Conditional exogeneity requires "no $Z$-confounders"
     - equivalent to having no unobserved confounders in the reduced form (RF) $Y \sim Z$
     - more precise to say **"no RF-confounders"**.
   - is it plausible? could instrument be endogenous to outcome?
   - Check if instrument is correlated with predetermined variables
3. Exclusion restriction
   - instrument $Z$ only affects $Y$ through treatment $D$.
     - could be referred to as **"single mediator condition"**
     - $D$ is the only mediator between $Z$ and $Y$

# IV is powerful but depends on quality of instruments

1. Relevance
   - Always report the first stage (F-test above 10)
2. Exogeneity
   - Conditional exogeneity requires "no $Z$-confounders"
     - equivalent to having no unobserved confounders in the reduced form (RF) $Y \sim Z$
     - more precise to say **"no RF-confounders"**.
   - is it plausible? could instrument be endogenous to outcome?
   - Check if instrument is correlated with predetermined variables
3. Exclusion restriction
   - instrument $Z$ only affects $Y$ through treatment $D$.
     - could be referred to as **"single mediator condition"**
     - $D$ is the only mediator between $Z$ and $Y$
   - is it plausible? what are other possible links between $Z$ and $Y$?
   - Specify group which is affected by the instrument (local average treatment effect)

# Resume Group Activity on IV

- Rejoin the same breakout group as last time and finish group discussions of instrument validity from your custom causal graph.
- Link to posts (please add yours if it is not posted yet):

  https://padlet.com/eash44/zautjkdu6vnzg8qs

# Encoders and Explanations

This lecture is about:

1. encoding high-dimensional datasets down to lower dimensions (dimensionality reduction)
2. explaining the predictions of classifiers and regressors

# Encoders and Explanations

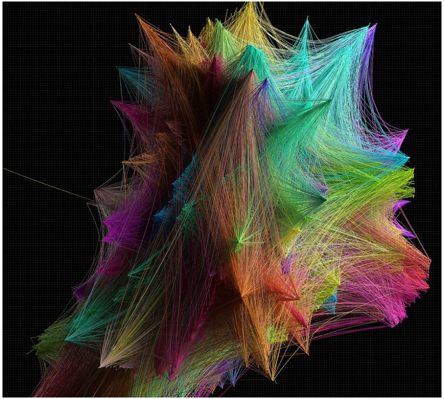
This lecture is about:

1. encoding high-dimensional datasets down to lower dimensions (dimensionality reduction)
2. explaining the predictions of classifiers and regressors

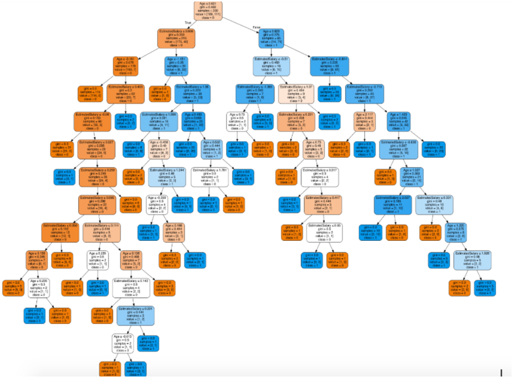
We will see that these are highly overlapping tasks.

# High-dimensional datasets and machine learning models are black boxes

High-Dimensional Datasets

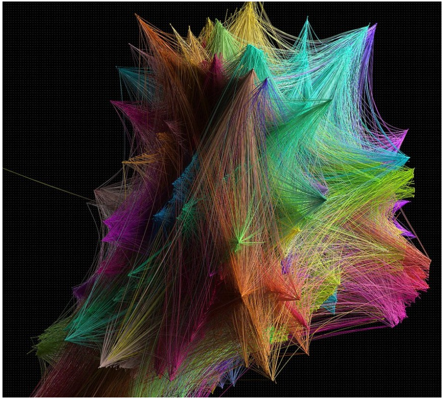


Big ML Models

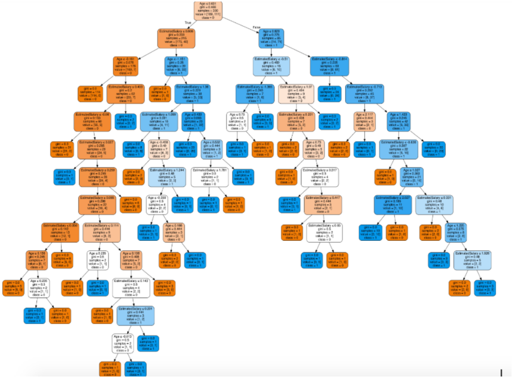


# High-dimensional datasets and machine learning models are black boxes

High-Dimensional Datasets



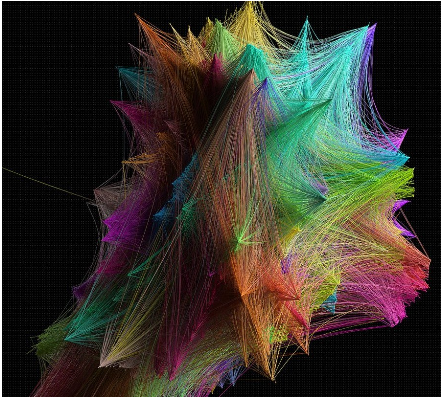
Big ML Models



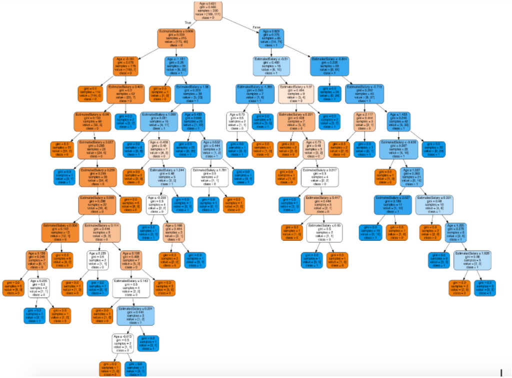
- dimensionality reduction helps us understand data.
- explanation/interpretability help us understand models.

# High-dimensional datasets and machine learning models are black boxes

High-Dimensional Datasets



Big ML Models



- dimensionality reduction helps us understand data.
- explanation/interpretability help us understand models.
- further: models are themselves a compressed representation of the data they are trained on.

# What have we been doing? ***Learning representations*** of the data

- e.g. a classifier: produces $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i; \hat{\theta})$, a vector of predicted probabilities across classes for each observation $i$.

# What have we been doing? ***Learning representations*** of the data

- e.g. a classifier: produces $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i; \hat{\theta})$, a vector of predicted probabilities across classes for each observation $i$.
  - This vector of class probabilities is a **compressed representation** of the outcome-predictive features $\boldsymbol{x}_i$

# What have we been doing? ***Learning representations*** of the data

- e.g. a classifier: produces $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i; \hat{\theta})$, a vector of predicted probabilities across classes for each observation $i$.
  - This vector of class probabilities is a **compressed** **representation** of the outcome-predictive features $\boldsymbol{x}_i$
  - the vector of features, $\boldsymbol{x}_i$, is itself a compressed representation (measurement) of some real-world complex object (e.g. a document) $\mathcal{D}_i$.

# What have we been doing? ***Learning representations*** of the data

- e.g. a classifier: produces $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i; \hat{\theta})$, a vector of predicted probabilities across classes for each observation $i$.
  - This vector of class probabilities is a **compressed representation** of the outcome-predictive features $\boldsymbol{x}_i$
  - the vector of features, $\boldsymbol{x}_i$, is itself a compressed representation (measurement) of some real-world complex object (e.g. a document) $\mathcal{D}_i$.
- Correspondingly: the learned parameters $\hat{\theta}$ can also be understood as a **learned compressed representation of the whole dataset**:
  - it contains information about the training dataset, and in particular how the features and outcomes are related.

# Information in $\hat{\theta}$

Say we train a multinomial logistic regression to classify observations $\boldsymbol{x}_i$ to classes $\boldsymbol{y}_i$:

$$\hat{y}_k(\boldsymbol{x}_i) = \frac{\exp(\theta_k \cdot \boldsymbol{x}_i)}{\sum_{l=1}^{n_y} \exp(\theta_l \cdot \boldsymbol{x}_i)}$$

- Let $\theta$ be the learned matrix of parameters relating features to outcomes:

$$\begin{bmatrix} \theta_{11} & & \cdots & & \theta_{1n_y} \\ & \ddots & & & \\ \vdots & & \theta_{jk} & & \vdots \\ & & & \ddots & \\ \theta_{n_x 1} & & \cdots & & \theta_{n_x n_y} \end{bmatrix}$$

  - It contains $n_y$ columns, which are $n_x$-vectors representing the outcome classes.

# Information in $\hat{\theta}$

Say we train a multinomial logistic regression to classify observations $\boldsymbol{x}_i$ to classes $\boldsymbol{y}_i$:

$$\hat{y}_k(\boldsymbol{x}_i) = \frac{\exp(\theta_k \cdot \boldsymbol{x}_i)}{\sum_{l=1}^{n_y} \exp(\theta_l \cdot \boldsymbol{x}_i)}$$

- Let $\theta$ be the learned matrix of parameters relating features to outcomes:

$$\begin{bmatrix} \theta_{11} & & \cdots & & \theta_{1n_y} \\ & \ddots & & & \\ \vdots & & \theta_{jk} & & \vdots \\ & & & \ddots & \\ \theta_{n_x 1} & & \cdots & & \theta_{n_x n_y} \end{bmatrix}$$

  - It contains $n_y$ columns, which are $n_x$-vectors representing the outcome classes.
  - It contains $n_x$ rows, which are $n_y$-vectors representing each feature input.

# Information in $\hat{\theta}$

Say we train a multinomial logistic regression to classify observations $\boldsymbol{x}_i$ to classes $\boldsymbol{y}_i$:

$$\hat{y}_k(\boldsymbol{x}_i) = \frac{\exp(\theta_k \cdot \boldsymbol{x}_i)}{\sum_{l=1}^{n_y} \exp(\theta_l \cdot \boldsymbol{x}_i)}$$

- Let $\theta$ be the learned matrix of parameters relating features to outcomes:

$$\begin{bmatrix} \theta_{11} & & \cdots & & \theta_{1n_y} \\ & \ddots & & & \\ \vdots & & \theta_{jk} & & \vdots \\ & & & \ddots & \\ \theta_{n_x 1} & & \cdots & & \theta_{n_x n_y} \end{bmatrix}$$

  - It contains $n_y$ columns, which are $n_x$-vectors representing the outcome classes.
  - It contains $n_x$ rows, which are $n_y$-vectors representing each feature input.
- The columns and rows of $\theta$ contain **interpretable** information about the task-relevant dimensions of the data.
- E.g., could compute cosine similarity between the vectors:
  - column vectors are informative about which outcomes are similar/related.
  - row vectors are informative about which features are similar/related.

# Information in causal estimate $\hat{\rho}$

Correspondingly, say we estimate a differences-in-differences regression:

$$Y_{it} = \alpha_i + \alpha_t + \rho D_{it} + \epsilon_{it}$$

where we obtain an OLS estimate $\hat{\rho}$ with standard error $\hat{\sigma}_\rho$.

# Information in causal estimate $\hat{\rho}$

Correspondingly, say we estimate a differences-in-differences regression:

$$Y_{it} = \alpha_i + \alpha_t + \rho D_{it} + \epsilon_{it}$$

where we obtain an OLS estimate $\hat{\rho}$ with standard error $\hat{\sigma}_\rho$.

- the statistics ($\hat{\rho}$,$\hat{\sigma}_\rho$) are an extremely **compressed** representation of the task-relevant (i.e. *policy-relevant*) features of the dataset.
  - $\hat{\rho}$ is **interpretable** as a counterfactual prediction for the average treatment effect on $Y_{it}$ of intervening on $D_{it}$.
  - $\hat{\sigma}_\rho$ is **interpretable** as the degree of uncertainty in that counterfactual prediction.

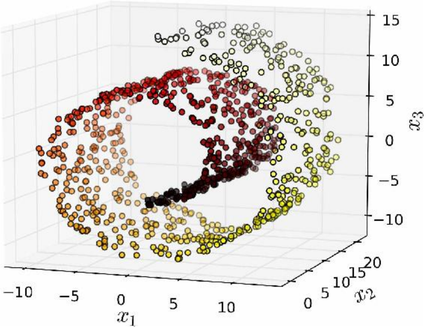
# Outline

Dimensionality Reduction

Model Explanation

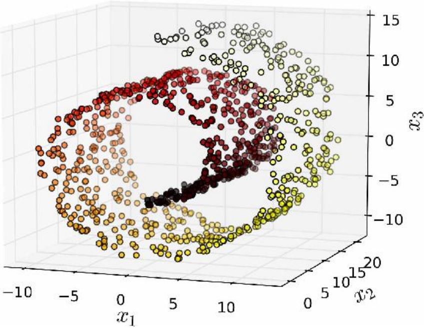
- Datasets are not distributed uniformly across the feature space.
- They have a lower-dimensional latent structure – a **manifold** – that can be learned.

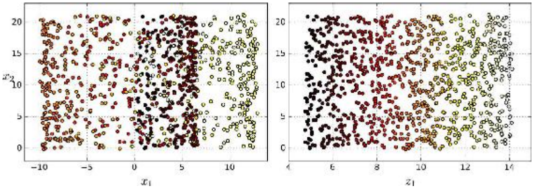
**"The Swiss Roll"**



- Datasets are not distributed uniformly across the feature space.
- They have a lower-dimensional latent structure – a **manifold** – that can be learned.

"The Swiss Roll"




- **Dimensonality reduction** makes data more interpretable – for example by projecting down to two dimensions for visualization.
- improves computational tractability.
- can improve model performance.

# Feature Importance / Selection

- For supervised learning tasks, doing feature selection to drop weak predictors has intuitive appeal.
  - Lasso models (with L1 penalty) do feature selection and produce sparse models.

# Feature Importance / Selection

- For supervised learning tasks, doing feature selection to drop weak predictors has intuitive appeal.
  - Lasso models (with L1 penalty) do feature selection and produce sparse models.

Feature selection can be done as a pre-processing step:

```
from sklearn.feature_selection import SelectKBest, chi2
selector = SelectKBest(chi2, k=10)
X_train_filtered = selector.fit_transform(X_train,y_train)
```

- $\chi^2$ (used for classification) is fast but features must be non-negative. With negative predictors, can use F-statistics (**f_classif** for classification, **f_regression** for regression).

# Feature Importance / Selection

- For supervised learning tasks, doing feature selection to drop weak predictors has intuitive appeal.
  - Lasso models (with L1 penalty) do feature selection and produce sparse models.

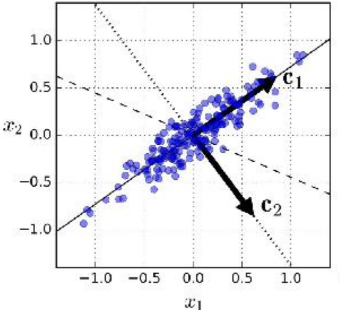
Feature selection can be done as a pre-processing step:

```
from sklearn.feature_selection import SelectKBest, chi2
selector = SelectKBest(chi2, k=10)
X_train_filtered = selector.fit_transform(X_train,y_train)
```

- $\chi^2$ (used for classification) is fast but features must be non-negative. With negative predictors, can use F-statistics (**`f_classif`** for classification, **`f_regression`** for regression).
- `chi2`, `f_classif`, and `f_regression` measure linear correlations. Mutual information captures higher-order dependencies (**`mutual_info_classif`**, **`mutual_info_regression`**). Slower to compute.

# PCA (principal component analysis) / SVD (singular value decomposition)

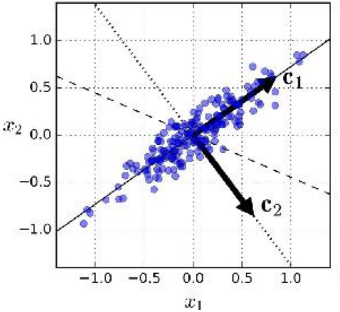
# PCA (principal component analysis) / SVD (singular value decomposition)




- PCA computes the dimension in data explaining most variance.

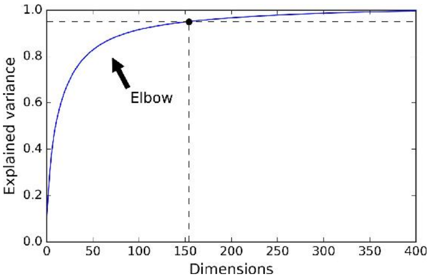
```
from sklearn.decomposition import PCA
pca = PCA(n_components=10)
X_train_pca = pca.fit_transform(X_train)
```

# PCA (principal component analysis) / SVD (singular value decomposition)




- PCA computes the dimension in data explaining most variance.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=10)
X_train_pca = pca.fit_transform(X_train)
```




- after the first component, subsequent components learn the (orthogonal) dimensions explaining most variance in dataset after projecting out first component.

# PCA for Dimension Reduction

- Data can be reduced by projecting down to first principal component dimensions.
  - can be used as predictors instead of the original matrix.
  - distance metrics between observations are approximately preserved.

# PCA for Dimension Reduction

- Data can be reduced by projecting down to first principal component dimensions.
  - can be used as predictors instead of the original matrix.
  - distance metrics between observations are approximately preserved.
- Can produce lossy *"reconstruction"* back in original space using `pca.inverse_transform()` method.

# PCA for Dimension Reduction

- Data can be reduced by projecting down to first principal component dimensions.
  - can be used as predictors instead of the original matrix.
  - distance metrics between observations are approximately preserved.
- Can produce lossy *"reconstruction"* back in original space using `pca.inverse_transform()` method.

**Caveats**:

- Standard PCA requires whole dataset in memory and is computationally costly.
  - use sklearn's `IncrementalPCA` to train in mini-batches.
  - `MiniBatchSparsePCA` learns regularized sparse components using an L1 penalty.

# PCA for Dimension Reduction

- Data can be reduced by projecting down to first principal component dimensions.
  - can be used as predictors instead of the original matrix.
  - distance metrics between observations are approximately preserved.
- Can produce lossy *"reconstruction"* back in original space using `pca.inverse_transform()` method.

**Caveats**:

- Standard PCA requires whole dataset in memory and is computationally costly.
  - use sklearn's `IncrementalPCA` to train in mini-batches.
  - `MiniBatchSparsePCA` learns regularized sparse components using an L1 penalty.
- PCA might destroy (a lot of) predictive information in your dataset.
  - compromise: use feature selection to keep strong predictors, and take principal components of weak predictors.

# PCA for Dimension Reduction

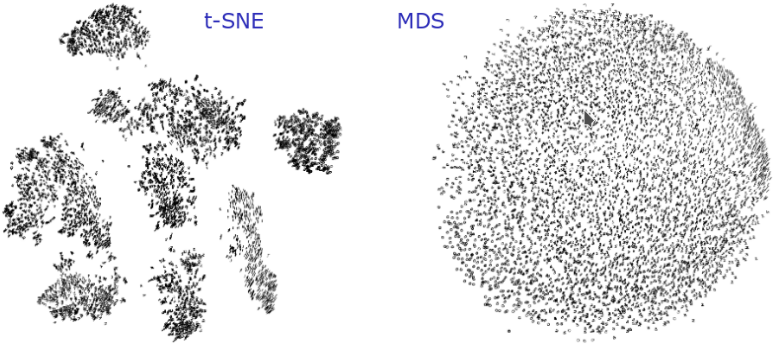
- Data can be reduced by projecting down to first principal component dimensions.
  - can be used as predictors instead of the original matrix.
  - distance metrics between observations are approximately preserved.
- Can produce lossy *"reconstruction"* back in original space using `pca.inverse_transform()` method.

**Caveats**:

- Standard PCA requires whole dataset in memory and is computationally costly.
  - use sklearn's `IncrementalPCA` to train in mini-batches.
  - `MiniBatchSparsePCA` learns regularized sparse components using an L1 penalty.
- PCA might destroy (a lot of) predictive information in your dataset.
  - compromise: use feature selection to keep strong predictors, and take principal components of weak predictors.
- dimensions are not interpretable.
  - For non-negative data (e.g. counts or frequencies), **Non-negative Matrix Factorization (NMF)** provides more interpretable factors than PCA.

# Dimension Reduction for Visualization: t-SNE and MDS

# Dimension Reduction for Visualization: t-SNE and MDS




From: L. Van der Maaten & G. Hinton, Visualizing Data using t-SNE, Journal of Machine Learning Research 9 (2008) 2579-2605

- **t-Distributed Stochastic Neighbor Embedding (t-SNE)** tries to keep similar observations close and dissimilar observations apart.
  - Useful for visualizing clusters of observations in high-dimensional space
- **Multidimensional Scaling (MDS)** tries to preserve distances between observations .

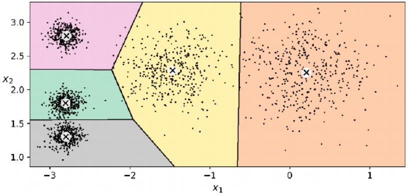
# $k$-means clustering separates observations into $k$ groups

# $k$-means clustering separates observations into $k$ groups

- Matrix of predictors treated as a Euclidean space (should standardize all columns)
- algorithm: initialize cluster centroids randomly, then shift around to minimize sum of within-cluster squared distance

# $k$-means clustering separates observations into $k$ groups

- Matrix of predictors treated as a Euclidean space (should standardize all columns)
- algorithm: initialize cluster centroids randomly, then shift around to minimize sum of within-cluster squared distance

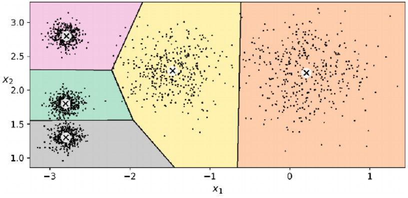



*K-Means decision boundaries (Voronoi tessellation)*

```
from sklearn.cluster import Kmeans
kmeans = KMeans(n_clusters=10)
kmeans.fit(X)
assigned_cluster = kmeans.labels_
```

# $k$-means clustering separates observations into $k$ groups

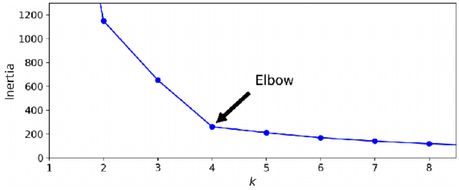
- Matrix of predictors treated as a Euclidean space (should standardize all columns)
- algorithm: initialize cluster centroids randomly, then shift around to minimize sum of within-cluster squared distance



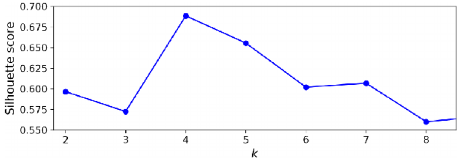

*K-Means decision boundaries (Voronoi tessellation)*

```
from sklearn.cluster import Kmeans
kmeans = KMeans(n_clusters=10)
kmeans.fit(X)
assigned_cluster = kmeans.labels_
```

$k$ (number of clusters) is the only hyperparameter, can select using:




*Selecting the number of clusters k using the "elbow rule"*




*Selecting the number of clusters k using the silhouette score*

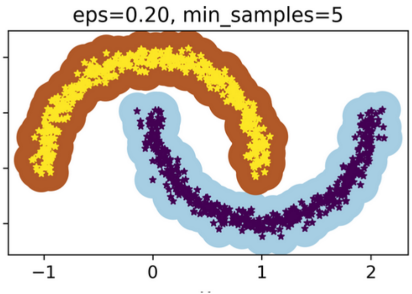
# Other clustering algorithms

- "k-medoid" clustering use L1 distance rather than Euclidean distance; produces the "medoid" (median vector) for each cluster rather than "centroid" (mean vector).
  - less sensitive to outliers, and medoid can be used as representative data point.

# Other clustering algorithms

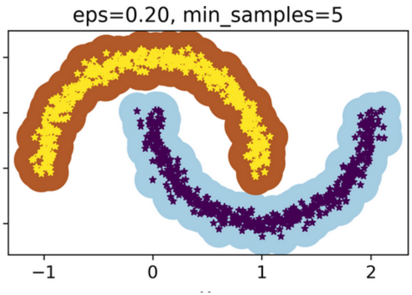
- "k-medoid" clustering use L1 distance rather than Euclidean distance; produces the "medoid" (median vector) for each cluster rather than "centroid" (mean vector).
  - less sensitive to outliers, and medoid can be used as representative data point.

- DBSCAN defines clusters as continous regions of high density.
  - detects and excludes outliers automatically



# Other clustering algorithms

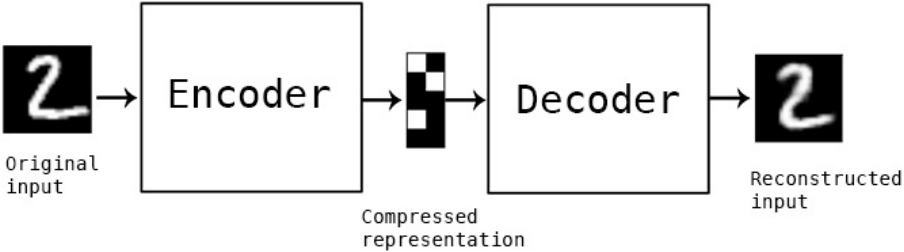
- "k-medoid" clustering use L1 distance rather than Euclidean distance; produces the "medoid" (median vector) for each cluster rather than "centroid" (mean vector).
  - less sensitive to outliers, and medoid can be used as representative data point.
- DBSCAN defines clusters as continous regions of high density.
  - detects and excludes outliers automatically




- Agglomerative (hierarchical) clustering makes nested clusters.
- can use assigned clusters, or distance to cluster centroids, as features in ML models.

# Autoencoders: Optimal Compression Algorithms

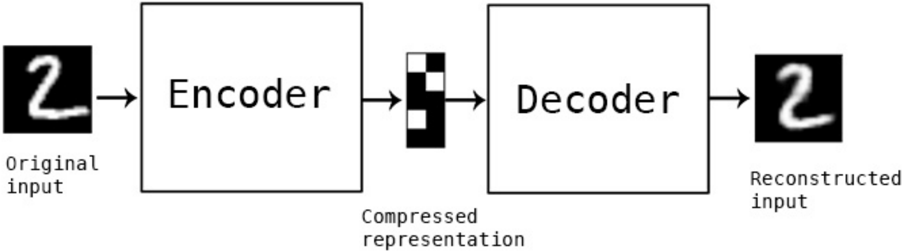
- Autoencoders = neural nets that perform optimal domain-specific lossy compression:




- Learned encodings can be decoded back to a *reconstruction* – a (minimally) lossy representation of the original data.

# Autoencoders: Optimal Compression Algorithms

- Autoencoders = neural nets that perform optimal domain-specific lossy compression:




- Learned encodings can be decoded back to a *reconstruction* – a (minimally) lossy representation of the original data.
- AE's can memorize complex, unstructured data.

# Autoencoder Architecture

- Stacked layers gradually decrease in dimensionality to create the compressed representation
- then gradually increase in dimensionality to try to reconstruct the input.
- can "tie weights" to make encoding layers and decoding layers symmetric.

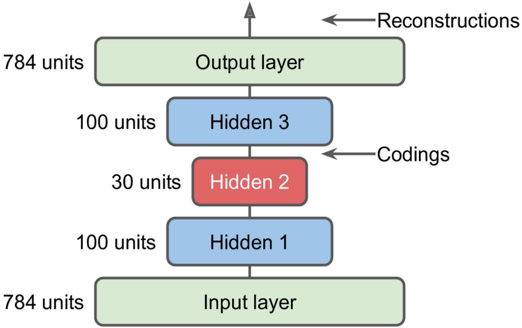



*Figure 17-3. Stacked autoencoder*

# Reconstruction from encoded vector

*Figure 17-4. Original images (top) and their reconstructions (bottom)*

- autoencoder encodings can be used the same way as principal components – as dimension-reduced features, or for computing similarity metrics.

# Autoencoders for Data Visualization

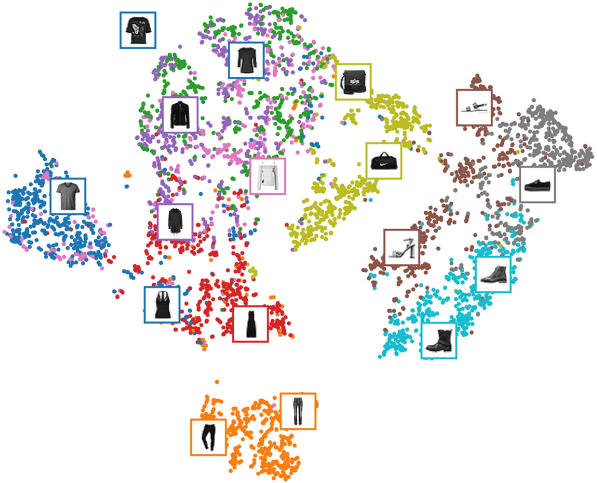
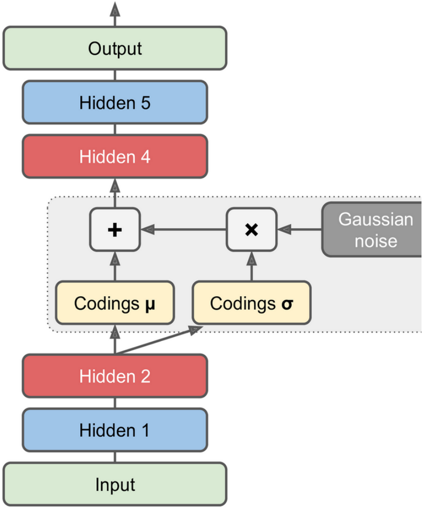


Figure 17-5. Fashion MNIST visualization using an autoencoder followed by t-SNE

- Decent baseline for visualizing the encodings:
  - use an autoencoder to compress your data to relatively low dimension (e.g. 32 dimensions)
  - then use t-SNE for mapping the compressed data to a 2D plane.

# Variational Autoencoders

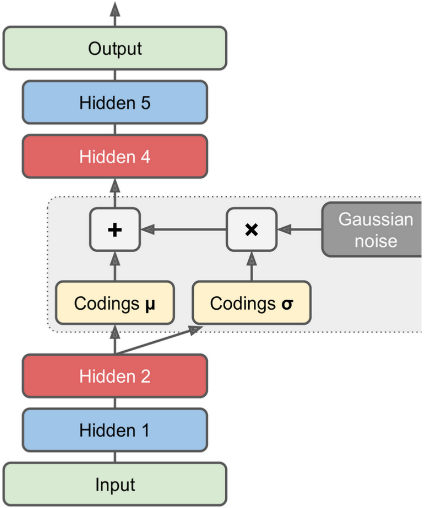
Encodings taken as parameters of a gaussian (means $\mu$ and variances $\sigma^2$)




Decoder draws from the distribution to produce first layer.

# Variational Autoencoders

Encodings taken as parameters of a gaussian (means $\mu$ and variances $\sigma^2$)

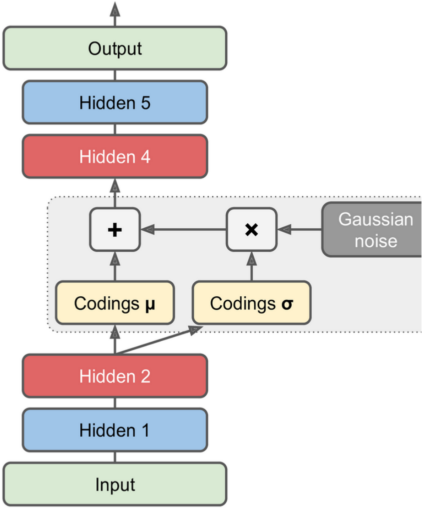



Decoder draws from the distribution to produce first layer.

- Can then sample from the normal distribution (or just choose numbers) and generate reconstructions.

# Variational Autoencoders

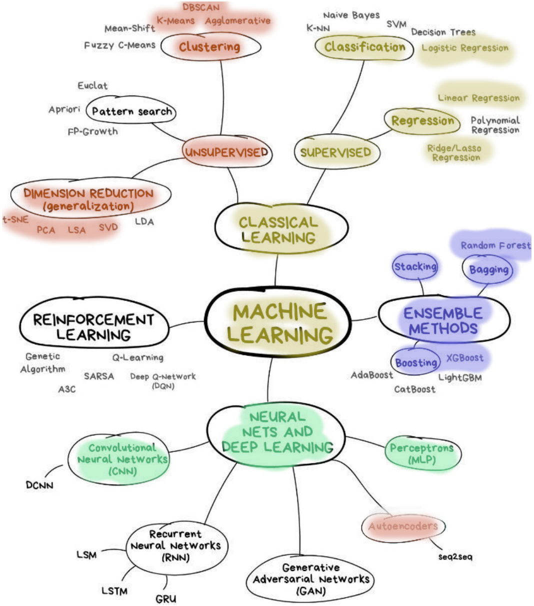
Encodings taken as parameters of a gaussian (means $\mu$ and variances $\sigma^2$)




Decoder draws from the distribution to produce first layer.

- Can then sample from the normal distribution (or just choose numbers) and generate reconstructions.
- VAE's do *semantic interpolation*: picking an encoding vector between two encodings will produce a reconstruction that is "between" the associated images

# Recap: The ML Landscape



# Outline

Dimensionality Reduction

Model Explanation

# Explaining Model Predictions

(a) Husky classified as wolf (b) Explanation

**Figure 11: Raw data and explanation of a bad model's prediction in the "Husky vs Wolf" task.**

- Machine learning models often make decisions for the wrong reasons.

$\leftarrow$ for example, classifying a dog image as a wolf because there is snow in the background (a correlated feature).

# Explaining Model Predictions

(a) Husky classified as wolf (b) Explanation

**Figure 11: Raw data and explanation of a bad model's prediction in the "Husky vs Wolf" task.**

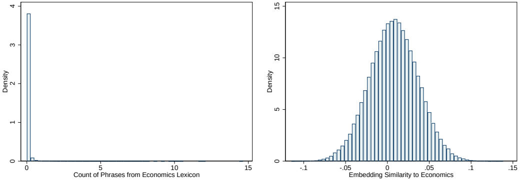
- Machine learning models often make decisions for the wrong reasons.

$\leftarrow$ for example, classifying a dog image as a wolf because there is snow in the background (a correlated feature).

- These problems, along with the black box nature of ML models, is a major hurdle to making these technologies trustworthy enough to use to support high-stakes decisions, like those in courts.

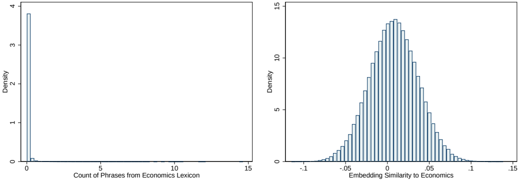
# A lot of what we do is interpreting/explaining

Summary statistics / plotting data:

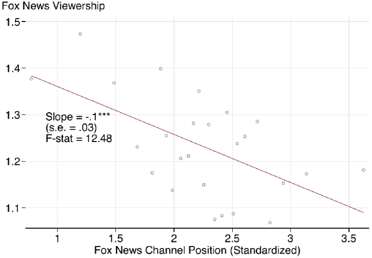


# A lot of what we do is interpreting/explaining

Summary statistics / plotting data:

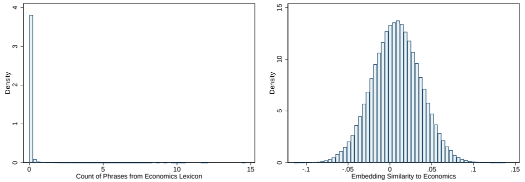



Scatter plots / regression plots:

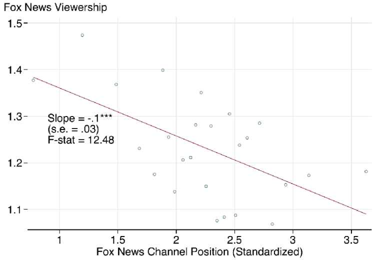


# A lot of what we do is interpreting/explaining

Summary statistics / plotting data:

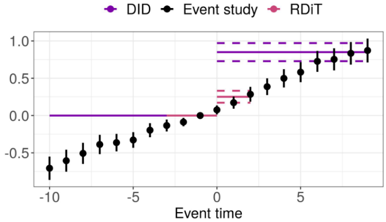



Scatter plots / regression plots:



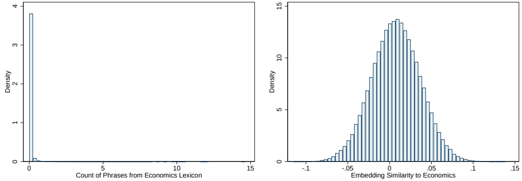

Event study plots "explain"

difference-in-difference regression estimates:

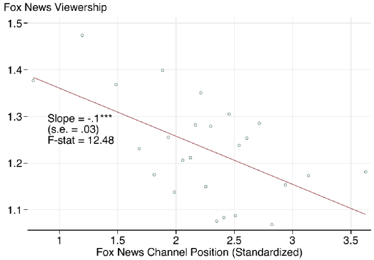


# A lot of what we do is interpreting/explaining

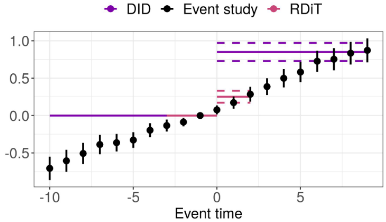
## Summary statistics / plotting data:



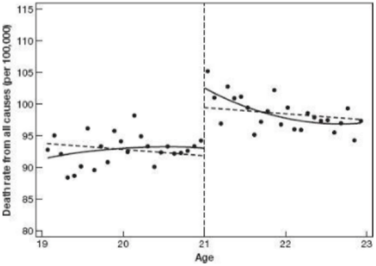

## Scatter plots / regression plots:




## Event study plots "explain" difference-in-difference regression estimates:




## Regression discontinuity plots "explain" regression discontinuity estimates:



# “Good” explanations

# “Good” explanations

- **Selective**: explanations should be short
  - i.e. low-dimensional.

# “Good” explanations

- **Selective**: explanations should be short
  - i.e. low-dimensional.
- **Social**: explanations should be targeted to the relevant audience.
  - e.g., different explanation for data scientists vs for lawyers.

# “Good” explanations

- **Selective**: explanations should be short
  - i.e. low-dimensional.
- **Social**: explanations should be targeted to the relevant audience.
  - e.g., different explanation for data scientists vs for lawyers.
- **Contrastive**: explains not just why a certain prediction was made, but why it was made instead of other predictions.

Perspective | Published: 13 May 2019

# Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead

Cynthia Rudin

*Nature Machine Intelligence* **1**, 206–215 (2019) | Cite this article



https://www.nature.com/articles/s42256-019-0048-x

# Linear Models

- Linear models (e.g. linear regression, logistic regression) are (relatively) interpretable:
  - coefficients (and t-statistics) provide some idea of the important features.

# Linear Models

- Linear models (e.g. linear regression, logistic regression) are (relatively) interpretable:
  - coefficients (and t-statistics) provide some idea of the important features.
- Caveats:
  - have to scale features for coefficients to be comparable
  - only small models (few predictors) are interpretable

# Linear Models

- Linear models (e.g. linear regression, logistic regression) are (relatively) interpretable:
  - coefficients (and t-statistics) provide some idea of the important features.
- Caveats:
  - have to scale features for coefficients to be comparable
  - only small models (few predictors) are interpretable
  - excludes interactions $\rightarrow$ often have bad ML performance

# Feature Importance / Selection

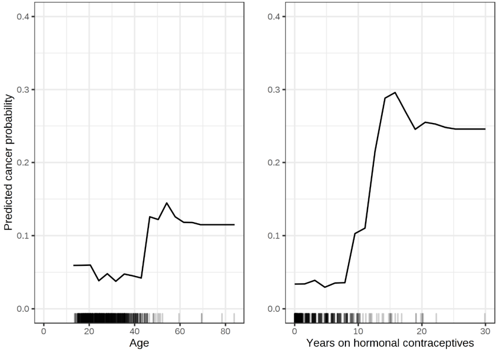
- Previously, we used feature selection metrics to drop weak predictors and reduce dimensionality:
  - (e.g. $\chi^2$, F-statistic, mutual information)
- the metrics for feature selection are also an indication of feature **importance** to the model, and hence provide interpretability.

# Visualizing Marginal Effects on Predictions: Partial Dependence Plots

- Select feature(s) to analyze. Take averages of all other features, and then form predictions $\hat{y}$ along the range of the analyzed feature. Tells you how model uses the feature.

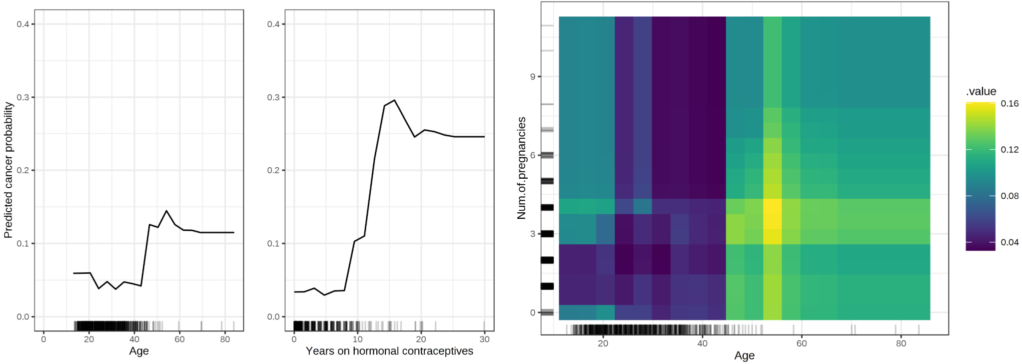
# Visualizing Marginal Effects on Predictions: Partial Dependence Plots

- Select feature(s) to analyze. Take averages of all other features, and then form predictions $\hat{y}$ along the range of the analyzed feature. Tells you how model uses the feature.



# Visualizing Marginal Effects on Predictions: Partial Dependence Plots

- Select feature(s) to analyze. Take averages of all other features, and then form predictions $\hat{y}$ along the range of the analyzed feature. Tells you how model uses the feature.



# Permutation Feature Importance

- What features are most important for prediction?
  - **`sklearn.feature_selection`** metrics can be done before training a model, but they exclude any interaction effects between predictors.

# Permutation Feature Importance

- What features are most important for prediction?
  - **`sklearn.feature_selection`** metrics can be done before training a model, but they exclude any interaction effects between predictors.

Solution: **Permutation feature importance** (Fisher, Rudin, and Dominici 2018):

# Permutation Feature Importance

- What features are most important for prediction?
  - **`sklearn.feature_selection`** metrics can be done before training a model, but they exclude any interaction effects between predictors.

Solution: **Permutation feature importance** (Fisher, Rudin, and Dominici 2018):

1. Estimate any model, compute performance metric.
2. For each feature $j$:
   - generate new dataset where feature $j$ is permuted (scrambled)
   - generate predictions and estimate new metric.
   - feature importance of $j$ is decrease in performance.

# Permutation Feature Importance

- What features are most important for prediction?
  - `sklearn.feature_selection` metrics can be done before training a model, but they exclude any interaction effects between predictors.

Solution: **Permutation feature importance** (Fisher, Rudin, and Dominici 2018):

1. Estimate any model, compute performance metric.
2. For each feature $j$:
   - generate new dataset where feature $j$ is permuted (scrambled)
   - generate predictions and estimate new metric.
   - feature importance of $j$ is decrease in performance.

Apply to trained `model` using test set:

```
from eli5.sklearn import PermutationImportance
perm = PermutationImportance(model)
perm.fit(X_test, y_test)
eli5.show_weights(perm)
```

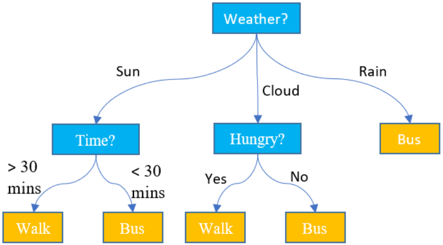
Out[20]:

| Weight | Feature |
|---|---|
| 0.4700 ± 0.0614 | OverallQual |
| 0.1439 ± 0.0156 | GrLivArea |
| 0.0499 ± 0.0034 | 2ndFlrSF |
| 0.0363 ± 0.0091 | TotalBsmtSF |
| 0.0294 ± 0.0032 | 1stFlrSF |
| 0.0271 ± 0.0102 | BsmtFinSF1 |
| 0.0166 ± 0.0028 | Fireplaces |
| 0.0130 ± 0.0068 | GarageArea |
| 0.0130 ± 0.0044 | YearBuilt |
| 0.0115 ± 0.0071 | LotArea |
| 0.0105 ± 0.0048 | GarageCars |
| 0.0105 ± 0.0048 | YearRemodAdd |

- could also check that model uses similar features in train/test set.

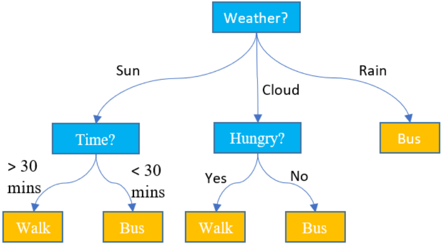
# Trees and Tree Ensembles

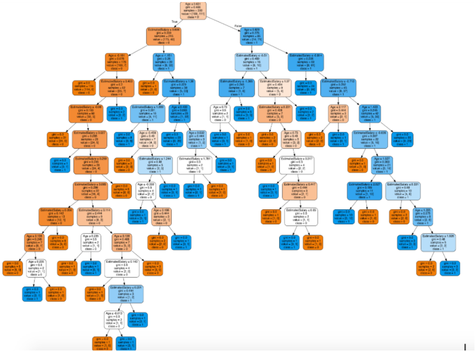
- Small decision trees have the advantage of being highly interpretable.



# Trees and Tree Ensembles

- Small decision trees have the advantage of being highly interpretable.



- Larger trees and ensembles (e.g. XGBoost) are not interpretable.
- Best-performing ML models are hard to interpret because they use lots of features and exploit non-linearities and interactions.

# Interpreting Tree Ensembles

XGBoost's Feature Importance Metric:

- At each decision node, compute **information gain** for feature $j$ **(change in predicted probability)**.
- Average across all nodes for each $j$.

Ranks predictors by their relative contributions.

```
from xgboost import plot_importance
plot_importance(xgb_reg, max_num_features=10)
```

# Clusters Provide Prototypes

- Clustering (e.g. $k$-means) can be used for description / explanation:
  - show selected variable values from centroid (even better, medoid) data points
  - medoids for large clusters are representative "prototypes" for the whole dataset (Molnar ch. 6.3).

# Clusters Provide Prototypes

- Clustering (e.g. $k$-means) can be used for description / explanation:
  - show selected variable values from centroid (even better, medoid) data points
  - medoids for large clusters are representative “prototypes” for the whole dataset (Molnar ch. 6.3).
- Conversely, data points that don’t fit well into a cluster (far from any centroid, or dropped by dbscan) are outliers (“criticisms”).

# Global Surrogate Model Explanation

## Approximate a black box model with an interpretable model

# Global Surrogate Model Explanation

## Approximate a black box model with an interpretable model

1. Get predictions $\hat{y}$ of the black box model from the data $X$.

# Global Surrogate Model Explanation

## Approximate a black box model with an interpretable model

1. Get predictions $\hat{y}$ of the black box model from the data $X$.
2. Train an interpretable model (lasso, decision tree, etc) on $X$ with $\hat{y}$ as the label.

# Global Surrogate Model Explanation

## Approximate a black box model with an interpretable model

1. Get predictions $\hat{y}$ of the black box model from the data $X$.
2. Train an interpretable model (lasso, decision tree, etc) on $X$ with $\hat{y}$ as the label.
3. Validate that the surrogate model replicates the predictions of the black box model
   - e.g., compute $R^2$ between black box $\hat{y}$ and surrogate $\hat{\hat{y}}$

# Local Feature Importance

**Local Surrogate Model**
(LIME = local interpretable model-agnostic explanations.)

# Local Feature Importance

**Local Surrogate Model**
(LIME = local interpretable model-agnostic explanations.)

1. Select data point to explain

# Local Feature Importance

**Local Surrogate Model**
(LIME = local interpretable
model-agnostic explanations.)

1. Select data point to explain
2. Get black box predictions for data point and for sample of randomly perturbed points in the neighborhood.

# Local Feature Importance

**Local Surrogate Model**
(LIME = local interpretable model-agnostic explanations.)

1. Select data point to explain
2. Get black box predictions for data point and for sample of randomly perturbed points in the neighborhood.
3. Use perturbed dataset to train interpretable surrogate model to predict $\hat{y}$.
   - e.g., lasso with high L1 penalty.

# Local Feature Importance

**Local Surrogate Model**
(LIME = local interpretable model-agnostic explanations.)

1. Select data point to explain
2. Get black box predictions for data point and for sample of randomly perturbed points in the neighborhood.
3. Use perturbed dataset to train interpretable surrogate model to predict $\hat{y}$.
   - e.g., lasso with high L1 penalty.

**Shapley Values**

Assigns importance to features by relative contribution to prediction (complicated formula based on solution concept in game theory)

# Local Feature Importance

**Local Surrogate Model**
(LIME = local interpretable model-agnostic explanations.)

1. Select data point to explain
2. Get black box predictions for data point and for sample of randomly perturbed points in the neighborhood.
3. Use perturbed dataset to train interpretable surrogate model to predict $\hat{y}$.
    - e.g., lasso with high L1 penalty.

**Shapley Values**

Assigns importance to features by relative contribution to prediction (complicated formula based on solution concept in game theory)

- (sometimes) better than LIME because accounts for interactions
- slower to compute
- default local importance metric on Google AI Platform

# Counterfactual Explanations (e.g. Wachter et al 2017)

Find the "closest" data point $X'$ to the current one $X$ that would change the outcome from $\hat{y}(X)$ to an alternative $y^*$:

# Counterfactual Explanations (e.g. Wachter et al 2017)

Find the “closest” data point $X'$ to the current one $X$ that would change the outcome from $\hat{y}(X)$ to an alternative $y^*$:

- that is, for ML prediction function $\hat{y}(\cdot)$, find counterfactual $X'$ that solves

$$\min_{X'} \lambda(\hat{y}(X') - y^*) + d(X, X')$$

where $d(\cdot)$ is a distance metric and $\lambda \geq 0$ calibrates the relative importance of label change and feature distance.

# Counterfactual Explanations (e.g. Wachter et al 2017)

Find the "closest" data point $X'$ to the current one $X$ that would change the outcome from $\hat{y}(X)$ to an alternative $y^*$:

- that is, for ML prediction function $\hat{y}(\cdot)$, find counterfactual $X'$ that solves

$$\min_{X'} \lambda(\hat{y}(X') - y^*) + d(X, X')$$

  where $d(\cdot)$ is a distance metric and $\lambda \geq 0$ calibrates the relative importance of label change and feature distance.
- To improve simplicity/interpretability, Wacher et al (2017) suggest:
  1. defining $d(\cdot)$ as the simple sum of distances between $X$ and $X'$ in each dimension – that is, the Manhattan (L1) distance $d(X, X') = \sum_{j=1}^{n_x} |x_j - x'_j|$.

# Counterfactual Explanations (e.g. Wachter et al 2017)

Find the "closest" data point $X'$ to the current one $X$ that would change the outcome from $\hat{y}(X)$ to an alternative $y^*$:

- that is, for ML prediction function $\hat{y}(\cdot)$, find counterfactual $X'$ that solves

$$\min_{X'} \lambda(\hat{y}(X') - y^*) + d(X, X')$$

  where $d(\cdot)$ is a distance metric and $\lambda \geq 0$ calibrates the relative importance of label change and feature distance.
- To improve simplicity/interpretability, Wacher et al (2017) suggest:
  1. defining $d(\cdot)$ as the simple sum of distances between $X$ and $X'$ in each dimension – that is, the Manhattan (L1) distance $d(X, X') = \sum_{j=1}^{n_x} |x_j - x'_j|$.
  2. Standardize all features by their respective *mean absolute deviation*, so dimensions are comparable in L1 space.

# Counterfactual Explanations (e.g. Wachter et al 2017)

Find the "closest" data point $X'$ to the current one $X$ that would change the outcome from $\hat{y}(X)$ to an alternative $y^*$:

- that is, for ML prediction function $\hat{y}(\cdot)$, find counterfactual $X'$ that solves

$$\min_{X'} \lambda(\hat{y}(X') - y^*) + d(X, X')$$

  where $d(\cdot)$ is a distance metric and $\lambda \geq 0$ calibrates the relative importance of label change and feature distance.
- To improve simplicity/interpretability, Wacher et al (2017) suggest:
  1. defining $d(\cdot)$ as the simple sum of distances between $X$ and $X'$ in each dimension – that is, the Manhattan (L1) distance $d(X, X') = \sum_{j=1}^{n_x} |x_j - x_j'|$.
  2. Standardize all features by their respective *mean absolute deviation*, so dimensions are comparable in L1 space.
- The mlxtend package makes this easy to do:

```
from mlxtend.evaluate import create_counterfactual
x_prime = create_counterfactual(x_reference=x_ref, # current data point
                                y_desired=1, # alternative outcome
                                model=clf, # trained model
                                X_dataset=X, # dataset
                                lammbda=1) # hyperparameter
```

# Extensions to Counterfactual Explanations

- Dandl et al (2020) improve on the Wachter et al objective in three ways:
    1. use Gower distance, rather than L1 distance, to allow for categorical features.

# Extensions to Counterfactual Explanations

- Dandl et al (2020) improve on the Wachter et al objective in three ways:
    1. use Gower distance, rather than L1 distance, to allow for categorical features.
    2. penalize the number of predictors that are changed, to encourage changes along relatively few dimensions.

# Extensions to Counterfactual Explanations

- Dandl et al (2020) improve on the Wachter et al objective in three ways:
  1. use Gower distance, rather than L1 distance, to allow for categorical features.
  2. penalize the number of predictors that are changed, to encourage changes along relatively few dimensions.
  3. reward counterfactuals that are likely to be possible – measured by how close they are to at least one observed data point.

# Explanation Methods: Overview

| | Non-Contrastive | Contrastive |
|---|---|---|
| Global | Feature Importance | Surrogate Model |
| Local | Shapley Values | Counterfactual |

# Pair Activity: Explanations for Decision Support

- We will now divide up into pairs in zoom breakout rooms.
- In each pair, two tasks (decide in your pair who does what):
  1. a judge making a decision on parole — what differences should there be for an explanation for judges, vs an explanation for defendants?
  2. a doctor making a decision about treatment — what differences should there be for an explanation for doctors, vs an explanation for patients?
- take 5 minutes to answer the question.
- take 2-3 minutes presenting answers to your partner
  - make modifications/extensions if they come up.
  - notate similarities and differences in answers between the settings, and think about why.
- Post your answers here:

  `https://padlet.com/eash44/6jza9w1yyrmbx90x`