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

3. Machine Learning Essentials

# Notes

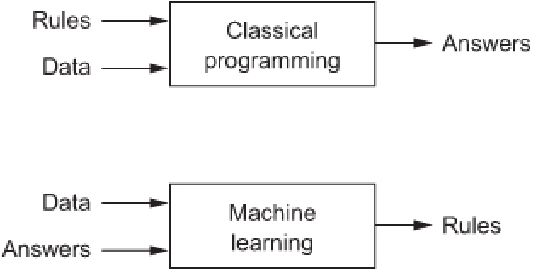
- Intuition on colliders – ask ChatGPT.
- First homework due tomorrow, submit if you want to continue participating in the class.
- Critical presentations:
  - Sign-up by tomorrow or be randomly assigned.
  - Template: `https://eash.cc/BRJ-pres-template`
  - First presentation video due this Saturday – any additional volunteers?

# Outline

# Learning Objectives

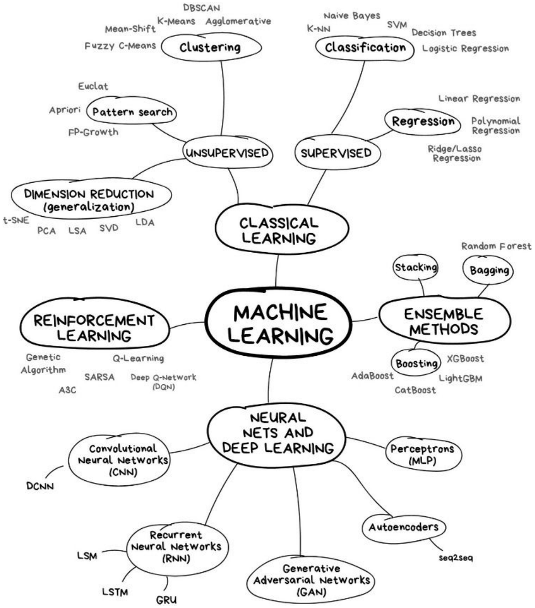
1. **Implement and evaluate machine learning pipelines.**
   - **Evaluate (find problems in) existing machine learning pipelines.**
   - **Design a pipeline to solve a given ML problem.**
   - **Implement some standard pipelines in Python.**
2. Implement and evaluate causal inference designs.
3. Understand how (not) to use data science tools (ML and CI) to support expert decision-making.

# What is machine learning?

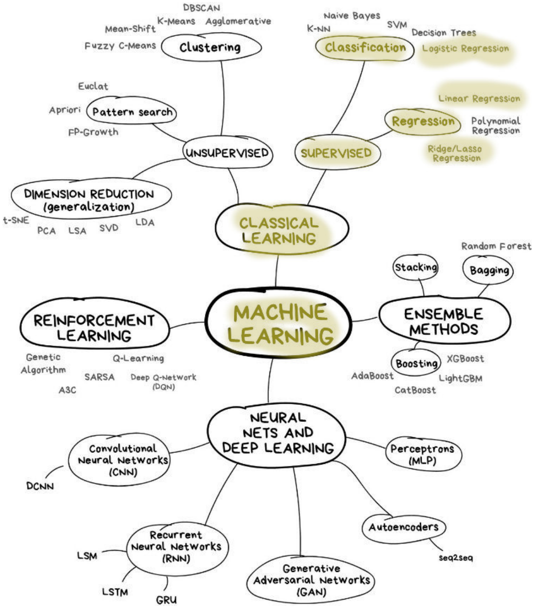



- In classical computer programming, humans input the rules and the data, and the computer provides answers.
- In machine learning, humans input the data and the answers, and the computer learns the rules.

# The Machine Learning Landscape



# What we will do today



# A Machine Learning Project, End-to-End

Aurelien Geron, *Hands-on machine learning with Scikit-Learn, Keras, & TensorFlow*, Chapter 2:

1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data to gain insights.
4. Prepare the data for Machine Learning algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.
8. Launch, monitor, and maintain your system.

# Three Types of (Standard) Machine Learning Problems

Determined by the data type of the outcome variable (or label):

- **Binary classification**: two choices, normalized to zero and one.
  - e.g., guilty or innocent

# Three Types of (Standard) Machine Learning Problems

Determined by the data type of the outcome variable (or label):

- **Binary classification**: two choices, normalized to zero and one.
  - e.g., guilty or innocent
- **Regression**: a one-dimensional, continuous, real-valued outcome.
  - e.g., number of days of prison assigned

# Three Types of (Standard) Machine Learning Problems

Determined by the data type of the outcome variable (or label):

- **Binary classification**: two choices, normalized to zero and one.
  - e.g., guilty or innocent
- **Regression**: a one-dimensional, continuous, real-valued outcome.
  - e.g., number of days of prison assigned
- **Multinomial Classification**: Three or more discrete, un-ordered outcomes.
  - e.g., predict what judge is assigned to a case: Alito, Breyer, or Cardozo

# What type of ML Problem is this?

- Based on defendant characteristics and the facts of the case, predict which charges the prosecutor will bring:
  - third degree murder (manslaughter)
  - second degree murder (crime of passion)
  - first degree murder (premeditated)

  **{binary classification, regression, or multinomial classification}**
- **write down your answer (30 secs)**

# What do ML Algorithms do? Minimize a cost function

# What do ML Algorithms do? Minimize a cost function

- A typical cost function (or loss function) for regression problems is Mean Squared Error (MSE):

$$\text{MSE}(\theta) = \frac{1}{n_D}\sum_{i=1}^{n_D}(h(x_i;\theta) - y_i)^2$$

  - $n_D$, the number of rows/observations
  - $x$, the matrix of predictors, with row $x_i$
  - $y$, the vector of outcomes, with item $y_i$
  - $h(x_i;\theta) = \hat{y}$ the model prediction (hypothesis)

# Loss functions, more generally

- The loss function $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ assigns a score based on prediction and truth:
  - Should be bounded from below, with the minimum attained only for cases where the prediction is correct.
- The average loss for the test set is

$$\mathcal{L}(\theta) = \frac{1}{n_D} \sum_{i=1}^{n_D} L(h(\boldsymbol{x}_i; \theta), \boldsymbol{y}_i)$$

- The estimated parameter matrix $\theta$ solves

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\theta)$$

$\hookrightarrow$ optimizes over parameter space; treats the data as constants.

# OLS Regression is Machine Learning

- Ordinary Least Squares Regression (OLS), also called simple linear regression, assumes the functional form $h(x;\theta) = x_i'\theta$ and minimizes the mean squared error (MSE)
$$\min_{\hat{\theta}} \frac{1}{n_D} \sum_{i=1}^{n_D} (x_i'\hat{\theta} - y_i)^2$$

# OLS Regression is Machine Learning

- Ordinary Least Squares Regression (OLS), also called simple linear regression, assumes the functional form $h(x;\theta) = x_i'\theta$ and minimizes the mean squared error (MSE)

$$\min_{\hat{\theta}} \frac{1}{n_D} \sum_{i=1}^{n_D} (x_i'\hat{\theta} - y_i)^2$$

- This minimand has a closed form solution

$$\hat{\theta} = (\boldsymbol{x}'\boldsymbol{x})^{-1}\boldsymbol{x}'\boldsymbol{y}$$

  - most machine learning models do **not** have a closed form solution $\rightarrow$ use numerical optimization (gradient descent).

$$\text{MSE}(\theta) = \frac{1}{n_D}\sum_{i=1}^{n_D}(h(\theta;\boldsymbol{x}_i) - y_i)^2$$

- The partial derivative for feature $j \in \{1, ..., n_x\}$ is

$$\frac{\partial \text{MSE}}{\partial \theta_j} = \frac{2}{n_D}\sum_{i=1}^{n_D}(\underbrace{h(\theta;\boldsymbol{x}_i) - y_i}_{\text{error for this obs}})\underbrace{\frac{\partial h(\theta;\boldsymbol{x}_i)}{\partial \theta_j}}_{\text{how } \theta_j \text{ shifts } h(\cdot)}$$

  - $\rightarrow$ estimates how changing $\theta_j$ would reduce the error across the whole dataset.

$$\mathrm{MSE}(\theta) = \frac{1}{n_D}\sum_{i=1}^{n_D}(h(\theta;\boldsymbol{x}_i) - y_i)^2$$

- The partial derivative for feature $j \in \{1, \ldots, n_x\}$ is

$$\frac{\partial \mathrm{MSE}}{\partial \theta_j} = \frac{2}{n_D}\sum_{i=1}^{n_D}(\underbrace{h(\theta;\boldsymbol{x}_i) - y_i}_{\text{error for this obs}})\underbrace{\frac{\partial h(\theta;\boldsymbol{x}_i)}{\partial \theta_j}}_{\text{how } \theta_j \text{ shifts } h(\cdot)}$$

  - $\rightarrow$ estimates how changing $\theta_j$ would reduce the error across the whole dataset.

- The ***gradient*** $\nabla$ gives the vector of these partial derivatives for all features:

$$\nabla_\theta \mathrm{MSE} = \begin{bmatrix} \frac{\partial \mathrm{MSE}}{\partial \theta_1} \\ \frac{\partial \mathrm{MSE}}{\partial \theta_2} \\ \vdots \\ \frac{\partial \mathrm{MSE}}{\partial \theta_{n_x}} \end{bmatrix}$$

$$\text{MSE}(\theta) = \frac{1}{n_D}\sum_{i=1}^{n_D}(h(\theta;\boldsymbol{x}_i) - y_i)^2$$

- The partial derivative for feature $j \in \{1, \dots, n_x\}$ is

$$\frac{\partial \text{MSE}}{\partial \theta_j} = \frac{2}{n_D}\sum_{i=1}^{n_D}\Big(\underbrace{h(\theta;\boldsymbol{x}_i) - y_i}_{\text{error for this obs}}\Big)\underbrace{\frac{\partial h(\theta;\boldsymbol{x}_i)}{\partial \theta_j}}_{\text{how } \theta_j \text{ shifts } h(\cdot)}$$

  - $\rightarrow$ estimates how changing $\theta_j$ would reduce the error across the whole dataset.

- The ***gradient*** $\nabla$ gives the vector of these partial derivatives for all features:

$$\nabla_\theta \text{MSE} = \begin{bmatrix} \frac{\partial \text{MSE}}{\partial \theta_1} \\ \frac{\partial \text{MSE}}{\partial \theta_2} \\ \vdots \\ \frac{\partial \text{MSE}}{\partial \theta_{n_x}} \end{bmatrix}$$

- ***Gradient descent*** nudges $\theta$ against the gradient (the direction that reduces MSE):

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \text{MSE}$$

  - $\eta$ = learning rate
- keep nudging until convergence.

$$\mathrm{MSE}(\theta) = \frac{1}{n_D}\sum_{i=1}^{n_D}(h(\theta;\boldsymbol{x}_i) - y_i)^2$$

- The partial derivative for feature $j \in \{1, ..., n_x\}$ is

$$\frac{\partial \mathrm{MSE}}{\partial \theta_j} = \frac{2}{n_D}\sum_{i=1}^{n_D}(\underbrace{h(\theta;\boldsymbol{x}_i) - y_i}_{\text{error for this obs}})\underbrace{\frac{\partial h(\theta;\boldsymbol{x}_i)}{\partial \theta_j}}_{\text{how } \theta_j \text{ shifts } h(\cdot)}$$

    - $\rightarrow$ estimates how changing $\theta_j$ would reduce the error across the whole dataset.

- The ***gradient*** $\nabla$ gives the vector of these partial derivatives for all features:

$$\nabla_\theta \mathrm{MSE} = \begin{bmatrix} \frac{\partial \mathrm{MSE}}{\partial \theta_1} \\ \frac{\partial \mathrm{MSE}}{\partial \theta_2} \\ \vdots \\ \frac{\partial \mathrm{MSE}}{\partial \theta_{n_x}} \end{bmatrix}$$

- ***Gradient descent*** nudges $\theta$ against the gradient (the direction that reduces MSE):

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathrm{MSE}$$

    - $\eta$ = learning rate
- keep nudging until convergence.

- ***Stochastic* gradient descent (SGD):** Compute gradient for single random instance (rather than whole dataset) at each iteration. Much faster, still works.

# Data Prep for Machine Learning

- Data Pre-Processing: See Geron Chapter 2 for pandas and sklearn syntax:
  - imputing missing values.
  - feature scaling (often helpful/necessary for ML models to work well)
    - if predictors are sparse (e.g. bag-of-words), use **`StandardScaler(with_mean=False)`**.
  - encoding categorical variables.

# Data Prep for Machine Learning

- Data Pre-Processing: See Geron Chapter 2 for pandas and sklearn syntax:
  - imputing missing values.
  - feature scaling (often helpful/necessary for ML models to work well)
    - if predictors are sparse (e.g. bag-of-words), use **`StandardScaler(with_mean=False)`**.
  - encoding categorical variables.
- Don't touch the data manually: use code for a reproducible data pipeline.

# Data Prep for Machine Learning

- Data Pre-Processing: See Geron Chapter 2 for pandas and sklearn syntax:
  - imputing missing values.
  - feature scaling (often helpful/necessary for ML models to work well)
    - if predictors are sparse (e.g. bag-of-words), use `StandardScaler(with_mean=False)`.
  - encoding categorical variables.
- Don't touch the data manually: use code for a reproducible data pipeline.
- Train/Test Split:
  - ML models can achieve arbitrarily high accuracy in-sample, so performance should be evaluated out-of-sample.

# Data Prep for Machine Learning

- Data Pre-Processing: See Geron Chapter 2 for pandas and sklearn syntax:
  - imputing missing values.
  - feature scaling (often helpful/necessary for ML models to work well)
    - if predictors are sparse (e.g. bag-of-words), use `StandardScaler(with_mean=False)`.
  - encoding categorical variables.
- Don't touch the data manually: use code for a reproducible data pipeline.
- Train/Test Split:
  - ML models can achieve arbitrarily high accuracy in-sample, so performance should be evaluated out-of-sample.
  - standard approach: randomly sample 80% training dataset to learn parameters, form predictions in 20% testing dataset for evaluating performance.

# Use Cross-Validation During Model Training

- Within the training set:
  - Use cross-validation with grid search to get model performance metrics across subsets of data using different hyperparameter specs.
  - Find the best hyperparameters for out-of-fold prediction in the training set.
- Then evaluate model performance in the test set using these hyperparameters.

# Use Cross-Validation During Model Training

- Within the training set:
  - Use cross-validation with grid search to get model performance metrics across subsets of data using different hyperparameter specs.
  - Find the best hyperparameters for out-of-fold prediction in the training set.
- Then evaluate model performance in the test set using these hyperparameters.
- Cross-validation is less common in deep learning, where training multiple models can be too computationally expensive.
  - instead, use dropout and early stopping.

# Model Evaluation in Test Set

Evaluating a "good" model is context-dependent. Here are some basics.

Regression:

- mean squared error (MSE)
- R-squared (same ranking as MSE, but units are more interpretable)
- mean absolute error (MAE, $\sum |\hat{y}(\theta) - y|$) is less sensitive to outliers.

# Model Evaluation in Test Set

Evaluating a "good" model is context-dependent. Here are some basics.

Regression:

- mean squared error (MSE)
- R-squared (same ranking as MSE, but units are more interpretable)
- mean absolute error (MAE, $\sum |\hat{y}(\theta) - y|$) is less sensitive to outliers.

Classification:

- more complicated, but accuracy is a good baseline:
  accuracy = (# correct test-set predictions) / (# of test-set observations)

# Model Evaluation in Test Set

Evaluating a "good" model is context-dependent. Here are some basics.

Regression:

- mean squared error (MSE)
- R-squared (same ranking as MSE, but units are more interpretable)
- mean absolute error (MAE, $\sum |\hat{y}(\theta) - y|$) is less sensitive to outliers.

Classification:

- more complicated, but accuracy is a good baseline:
  accuracy = (# correct test-set predictions) / (# of test-set observations)
- What if one of the outcomes is over-represented – e.g., 19 out of 20? Then I can guess the modal class and get 95% accuracy.
  - Some alternative classifier metrics designed to address class imbalance (more below and in week 5).

# Outline

# Regression models $\leftrightarrow$ Continuous outcome

- If the outcome is continuous (e.g., $Y$ = tax revenues collected, or criminal sentence imposed in months of prison):
- Need a regression model. Problems with OLS:
  - tends to over-fit training data.
  - cannot handle multicollinearity.

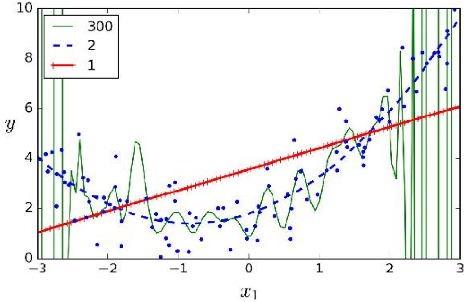


*Figure 4-14. High-degree Polynomial Regression*

# Regression models $\leftrightarrow$ Continuous outcome

- If the outcome is continuous (e.g., $Y =$ tax revenues collected, or criminal sentence imposed in months of prison):
- Need a regression model. Problems with OLS:
  - tends to over-fit training data.
  - cannot handle multicollinearity.

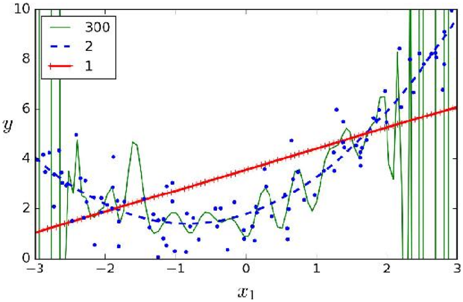



*Figure 4-14. High-degree Polynomial Regression*

- Machine learning models are evaluated by the fit in held-out data (the test set)
  - "Regularization" refers to ML model training methods designed to reduce/prevent over-fitting of the training set
  - (and hopefully better fit in the test set).

## Regularization

- Minimizing the loss $L$ directly usually results in over-fitting. It is standard to add **regularization**:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\theta} \frac{1}{n_D} \sum_{i=1}^{n_D} L(h(\boldsymbol{x}_i; \theta), \boldsymbol{y}_i) + \lambda R(\theta)$$

  - $R(\theta)$ is a "regularization function" or "regularizer", designed to reduce over-fitting.
  - $\lambda$ is a hyperparameter where higher values increase regularization.

---

# Regularization

- Minimizing the loss $L$ directly usually results in over-fitting. It is standard to add **regularization**:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\theta} \frac{1}{n_D} \sum_{i=1}^{n_D} L(h(\boldsymbol{x}_i;\theta), \boldsymbol{y}_i) + \lambda R(\theta)$$

  - $R(\theta)$ is a "regularization function" or "regularizer", designed to reduce over-fitting.
  - $\lambda$ is a hyperparameter where higher values increase regularization.

---

"Ridge" and "Lasso" penalize larger coefficients, shrinking them toward zero:

- Ridge (or L2) penalty:

$$R_2 = \|\theta\|_2^2 = \sum_{j=1}^{n_x} (\theta_j)^2$$

  - also helps select between collinear predictors.
- Lasso (or L1) penalty:

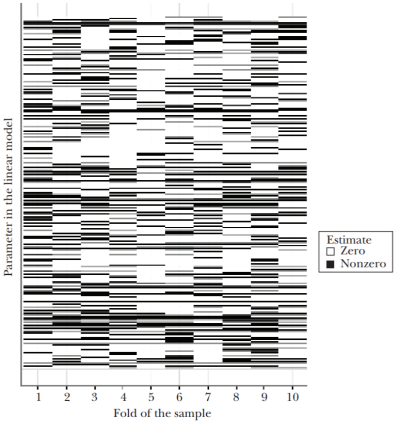
$$R_1 = \|\theta\|_1 = \sum_{j=1}^{n_x} |\theta_j|$$

  - also performs feature selection and outputs a sparse model.

# Does lasso pick the "true" model?

Lasso prediction of house prices with 150 variables – which variables are "selected" (non-zero coefficients) by lasso, in ten models trained on separate data subsamples (Mullainathan and Spiess 2017):

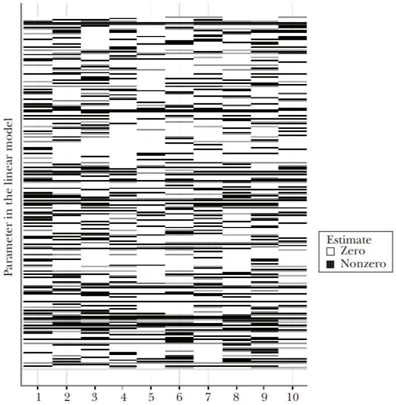
**Selected Coefficients (Nonzero Estimates) across Ten LASSO Regressions**



# Does lasso pick the “true” model?

Lasso prediction of house prices with 150 variables – which variables are “selected” (non-zero coefficients) by lasso, in ten models trained on separate data subsamples (Mullainathan and Spiess 2017):

**Selected Coefficients (Nonzero Estimates) across Ten LASSO Regressions**




- The set of lasso-selected variables changes across folds in the data
- $\rightarrow$ Lasso does not pick the “correct” predictors.
  - It just learns the correct $\hat{h}(X)$
  - when predictors are correlated with each other, they are substitutable.

# Elastic Net = Lasso + Ridge

The Elastic Net cost function is:

$$\begin{aligned} L(\theta) &= \mathrm{MSE}(\theta) + \lambda_1 R_1 + \lambda_2 R_2 \\ &= \mathrm{MSE}(\theta) + \lambda_1 \sum_{j=1}^{n_x} |\theta_j| + \lambda_2 \sum_{j=1}^{n_x} (\theta_j)^2 \end{aligned}$$

- $\lambda_1, \lambda_2$ = strength of L1 (Lasso) penalty and L2 (Ridge) penalty, respectively.

---

# Elastic Net = Lasso + Ridge

The Elastic Net cost function is:

$$\begin{aligned} L(\theta) &= \text{MSE}(\theta) + \lambda_1 R_1 + \lambda_2 R_2 \\ &= \text{MSE}(\theta) + \lambda_1 \sum_{j=1}^{n_x} |\theta_j| + \lambda_2 \sum_{j=1}^{n_x} (\theta_j)^2 \end{aligned}$$

- $\lambda_1, \lambda_2$ = strength of L1 (Lasso) penalty and L2 (Ridge) penalty, respectively.

---

In scikit-learn, e-net penalties are parametrized as "alpha" = total penalty, and "l1_ratio" = proportion of penalty to L1.

```
from sklearn.linear_model import ElasticNet
enet = ElasticNet(alpha=2.0, l1_ratio = .75) # L1 = 1.5, L2 = 0.5
enet.fit(X,y)
```

# Outline

# Binary Outcome $\leftrightarrow$ Binary Classification

- Binary classifiers try to match a boolean outcome $y \in \{0, 1\}$.
  - The standard approach is to apply a transformation (e.g. sigmoid/logit) to normalize $\hat{y} \in [0, 1]$.
  - Prediction rule is 0 for $\hat{y} < .5$ and 1 otherwise.

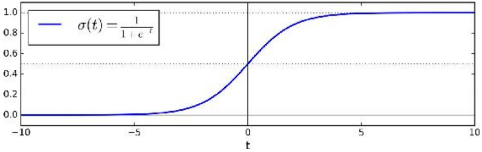
# Binary Outcome $\leftrightarrow$ Binary Classification

- Binary classifiers try to match a boolean outcome $y \in \{0, 1\}$.
  - The standard approach is to apply a transformation (e.g. sigmoid/logit) to normalize $\hat{y} \in [0, 1]$.
  - Prediction rule is 0 for $\hat{y} < .5$ and 1 otherwise.
- The binary cross-entropy (or log loss) is:

$$L(\theta) = \underbrace{-\frac{1}{n_D}}_{\text{negative}} \sum_{i=1}^{n_D} [\underbrace{y_i}_{y_i=1} \underbrace{\log(\hat{y}_i)}_{\text{log prob} y_i=1} + \underbrace{(1 - y_i)}_{y_i=0} \underbrace{\log(1 - \hat{y}_i)}_{\text{log prob} y_i=0}]$$

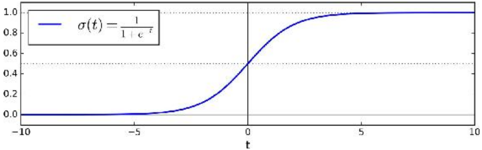
- In **logistic regression** we use a sigmoid transformation:

$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$



- In **logistic regression** we use a sigmoid transformation:

$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$



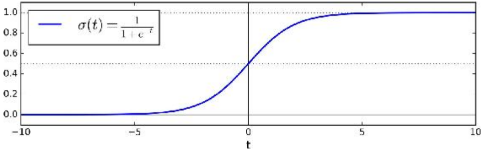

- Plugging into the binary-cross entropy loss gives the logistic regression cost objective:

$$\min_{\theta} \sum_{i=1}^{n_D} -y_i \log(\text{sigmoid}(\boldsymbol{x}_i \cdot \theta)) - [1 - y_i] \log(1 - \text{sigmoid}(\boldsymbol{x}_i \cdot \theta))$$

  - does not have a closed form solution, but it is convex (guaranteeing that gradient descent will find the global minimum).

- In **logistic regression** we use a sigmoid transformation:

$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$




- Plugging into the binary-cross entropy loss gives the logistic regression cost objective:

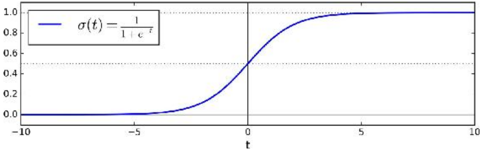
$$\min_{\theta} \sum_{i=1}^{n_D} -y_i \log(\text{sigmoid}(\boldsymbol{x}_i \cdot \theta)) - [1 - y_i] \log(1 - \text{sigmoid}(\boldsymbol{x}_i \cdot \theta))$$

  - does not have a closed form solution, but it is convex (guaranteeing that gradient descent will find the global minimum).
- The gradient for one data point is

$$\frac{\partial L(\theta)}{\partial \theta_j} = \underbrace{(\text{sigmoid}(\boldsymbol{x}_i \cdot \theta) - y_i)}_{\text{error for obs } i} \underbrace{x_i^j}_{\text{input } j}$$

- In **logistic regression** we use a sigmoid transformation:

$$\hat{y} = \text{sigmoid}(\boldsymbol{x} \cdot \theta) = \frac{1}{1 + \exp(-\boldsymbol{x} \cdot \theta)}$$




- Plugging into the binary-cross entropy loss gives the logistic regression cost objective:

$$\min_{\theta} \sum_{i=1}^{n_D} -y_i \log(\text{sigmoid}(\boldsymbol{x}_i \cdot \theta)) - [1 - y_i] \log(1 - \text{sigmoid}(\boldsymbol{x}_i \cdot \theta))$$

  - does not have a closed form solution, but it is convex (guaranteeing that gradient descent will find the global minimum).
- The gradient for one data point is

$$\frac{\partial L(\theta)}{\partial \theta_j} = \underbrace{(\text{sigmoid}(\boldsymbol{x}_i \cdot \theta) - y_i)}_{\text{error for obs } i} \underbrace{x_i^j}_{\text{input } j}$$

- Like linear regression, logistic regression can be regularized with L1 or L2 penalties.

```
from sklearn.linear_model import LogisticRegression
logit = LogisticRegression(penalty='l2', C = 2.0) # lambda = 1/2
logit.fit(X,y)
```

A **Confusion Matrix** is a nice way to visualize classifier performance:

| | | **Predicted Class** | |
|---|---|---|---|
| | | Negative | Positive |
| **True Class** | Negative | **# True Negatives** | # False Positives |
| | Positive | # False Negatives | **# True Positives** |

- Cell values give counts in the test set.

A **Confusion Matrix** is a nice way to visualize classifier performance:

| | | **Predicted Class** | |
|---|---|---|---|
| | | Negative | Positive |
| **True Class** | Negative | **# True Negatives** | # False Positives |
| | Positive | # False Negatives | **# True Positives** |

- Cell values give counts in the test set.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{False Positives} + \text{False Negatives} + \text{True Negatives}}$$

A **Confusion Matrix** is a nice way to visualize classifier performance:

| | | **Predicted Class** | |
|---|---|---|---|
| | | Negative | Positive |
| **True Class** | Negative | **# True Negatives** | # False Positives |
| | Positive | # False Negatives | **# True Positives** |

- Cell values give counts in the test set.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{False Positives} + \text{False Negatives} + \text{True Negatives}}$$

$$\text{Precision (for positive class)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- Precision decreases with false positives. "When I guess this outcome, I tend to guesses correctly."

A **Confusion Matrix** is a nice way to visualize classifier performance:

| | | **Predicted Class** | |
|---|---|---|---|
| | | Negative | Positive |
| **True Class** | Negative | **# True Negatives** | # False Positives |
| | Positive | # False Negatives | **# True Positives** |

- Cell values give counts in the test set.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{False Positives} + \text{False Negatives} + \text{True Negatives}}$$

$$\text{Precision (for positive class)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

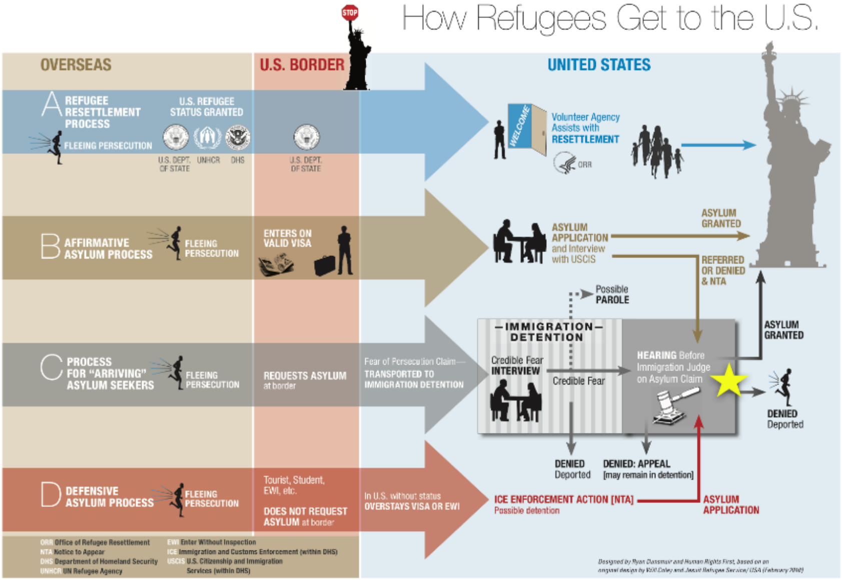
- Precision decreases with false positives. “When I guess this outcome, I tend to guesses correctly.”

$$\text{Recall (for positive class)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- Recall decreases with false negatives. “When this outcome occurs, I don’t miss it.”

# Outline

# Asylum in the U.S.




Source: rcusa.org.

# Dunn, Sagun, Sirin, and Chen (2017): Asylum Courts

- Data:
  - universe of asylum court cases, 1981-2013
  - 492,903 decisions, 336 courts, 441 judges

# Dunn, Sagun, Sirin, and Chen (2017): Asylum Courts

- Data:
  - universe of asylum court cases, 1981-2013
  - 492,903 decisions, 336 courts, 441 judges
- High stakes: denial of asylum results in deportation.
- Average grant rate: 35%.
- What type of ML problem is this?

# Predicting U.S. Asylum Court Decisions

| | | Predicted | |
|---|---|---|---|
| | | Denied | Granted |
| True | Denied | 195,223 | 65,798 |
| | Granted | 73,269 | 104,406 |

**Accuracy = 68.3%, F1 = 0.60**

- predictions made using logistic regression with L2 regularization, penalty selected by cross-validation grid search.

# Judge Identity is Most Predictive Factor

| Model | Accuracy | ROC AUC |
|---|---|---|
| Judge ID | 0.71 | 0.74 |
| Judge ID & Nationality | 0.76 | 0.82 |
| Judge ID & Opening Date | 0.73 | 0.77 |
| Judge ID & Nationality & Opening Date | 0.78 | 0.84 |
| Full model at case completion | 0.82 | 0.88 |

- Predictions from random forest classifier, with parameters selected by cross-validated grid search.
  - Training/test split 482K/120K.

# Judge Variation in Predictability

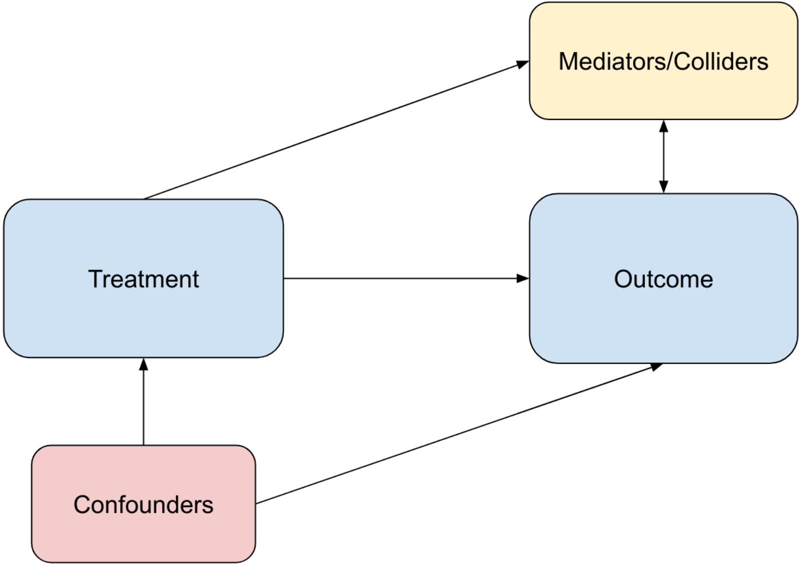
- Some judges are highly predictable, always granting or rejecting.
  - suggests they use heuristics or stereotypes rather than considering cases carefully.

# Judge Variation in Predictability

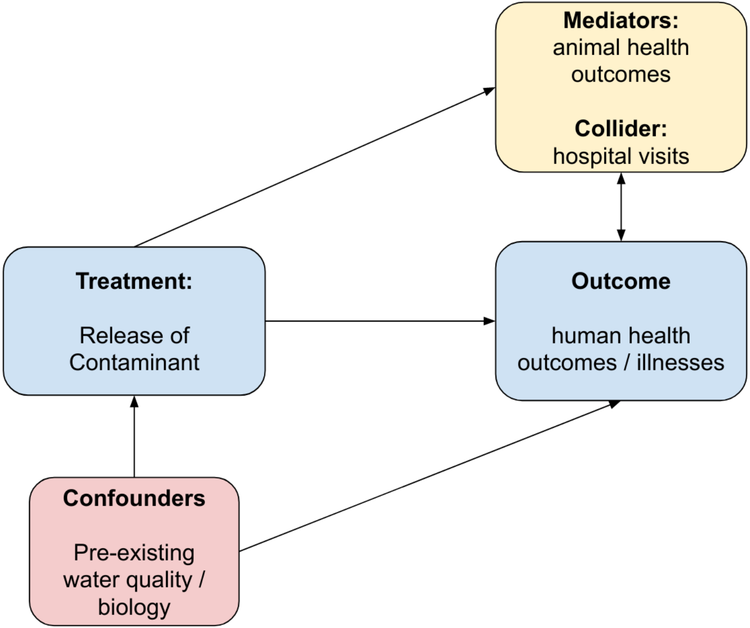
- Some judges are highly predictable, always granting or rejecting.
  - suggests they use heuristics or stereotypes rather than considering cases carefully.
- There is significant variation in predictability by judge, conditional on grant rate.
  - suggests disagreement about circumstances contributing to asylum decision.

# Outline

# Causal Graphs: Review



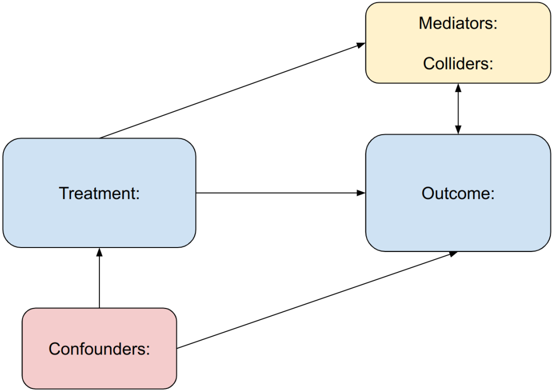
# Causal Graph Example: Pollution of a River



- Activity: Think of an example causal inference question:
    - a research question from your field
    - a policy you are interested in
    - a mystery you are fascinated by

- Activity: Think of an example causal inference question:
  - a research question from your field
  - a policy you are interested in
  - a mystery you are fascinated by

Causal graph template
(also at `http://bit.ly/BRJ-W5A2a`):




- Draw this template on a piece of paper (recommended), or save a copy of the template
- Fill in on paper or electronically.
- Share with neighbors.