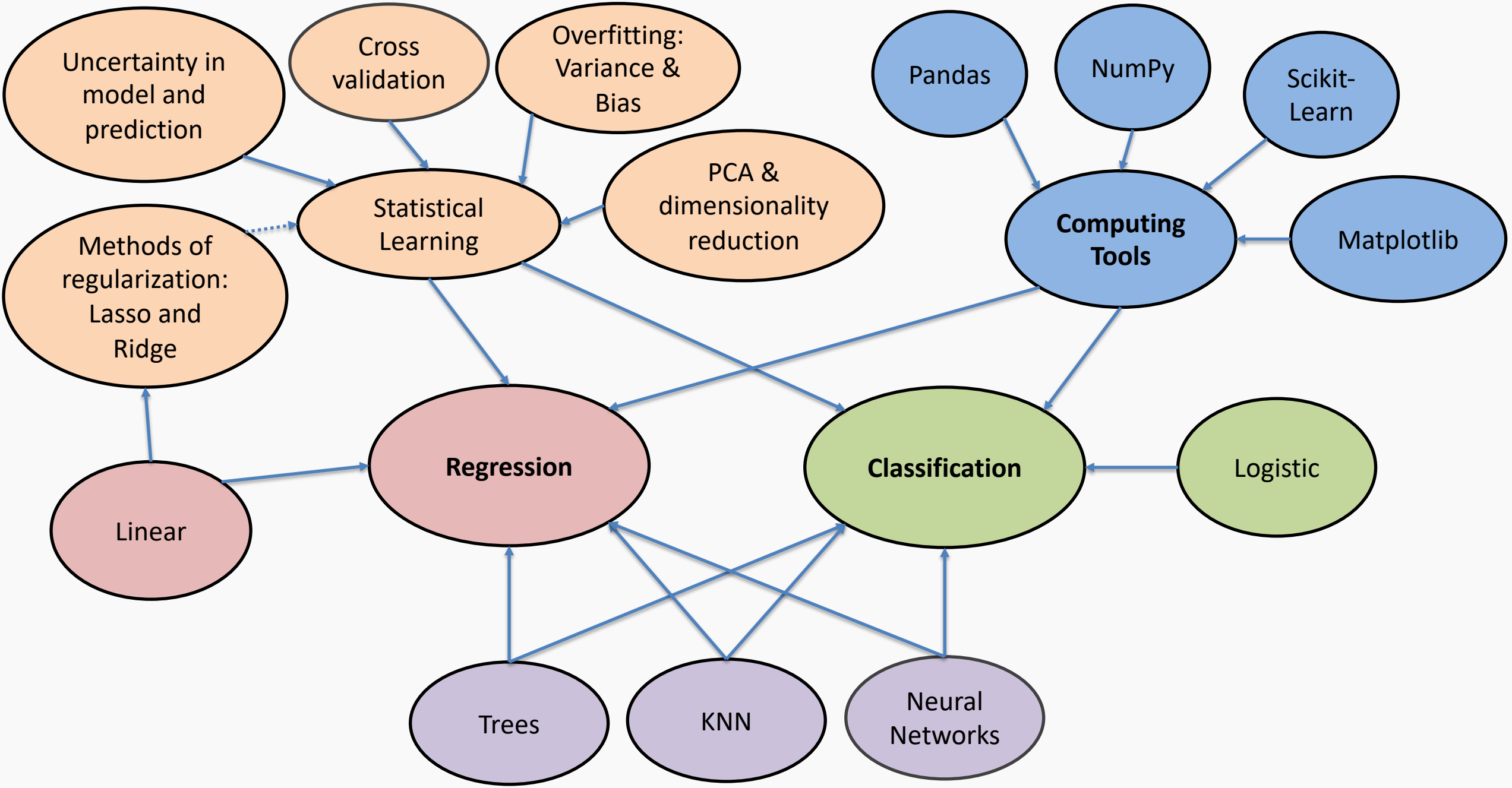
Lecture 36: Review

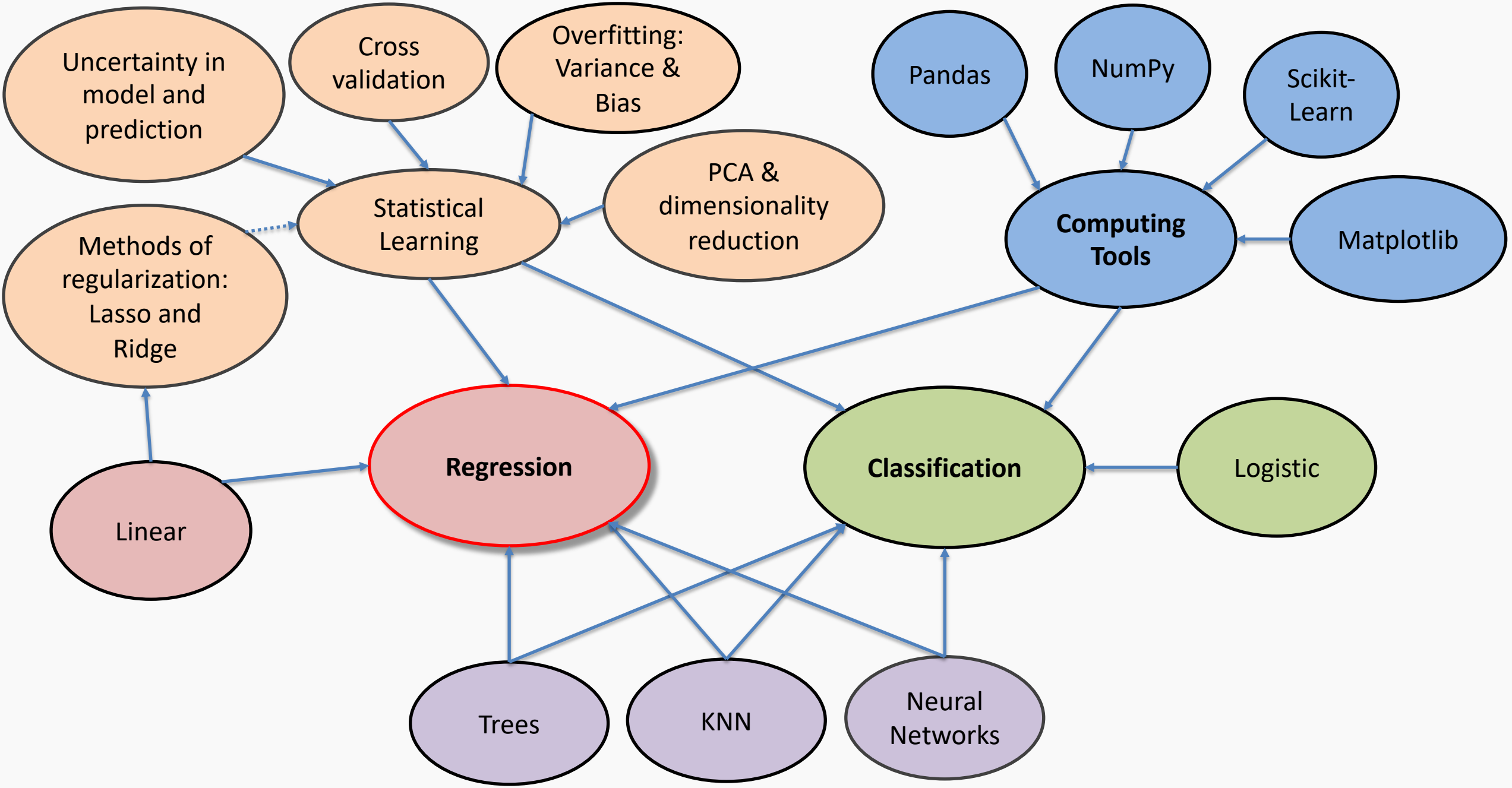
CS109A Introduction to Data Science

Pavlos Protopapas, Kevin Rader and Chris Tanner

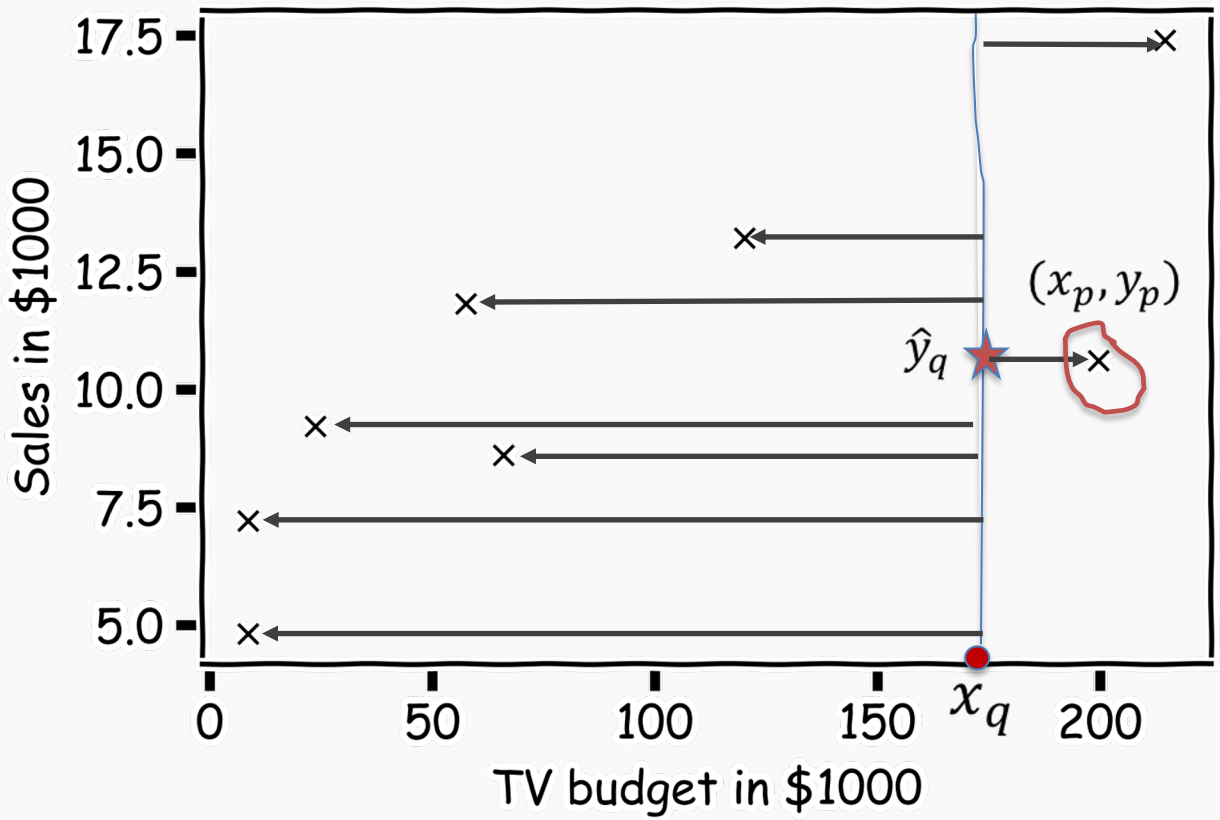
Lectures

- ⊙ Lecture 1: Introduction
- Lecture 2: Data
- ✓ Lecture 3: Data Sources, Parsing, and EDA
- Lecture 4: EDA + PANDAS
- ✓ ☿ Lecture 5: Introduction to Regression
- ♀ Lecture 6: Multi and Poly Regression
- ♁ Lecture 7: Model Selection
- ✓ Lecture 8: Probability in Regression
- ✓ ♂ Lecture 9: Inference in Linear Regression
- ♃ Lecture 10: Hypothesis Testing and Predictive CI
- ♄ Lecture 11: Regularization
- ✓ ♅ Lecture 12: Estimation of the Regularization Coefficients using CV and com...
- Lecture 13: Models, Data, Debugging, oh my!
- Lecture 14: Visualization
- or Lecture 15: Logistic Regression
- ✓ Lecture 16: Review and Preview
- Lecture 17: Logistic Regression II
- ✓ Lecture 18: Multiclass Logistic Regression
- Lecture 19: Missing Data and Imputation


Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks


Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks

Simple Prediction Model (KNN)



What is $\hat{y}_q$ at some x_q ?

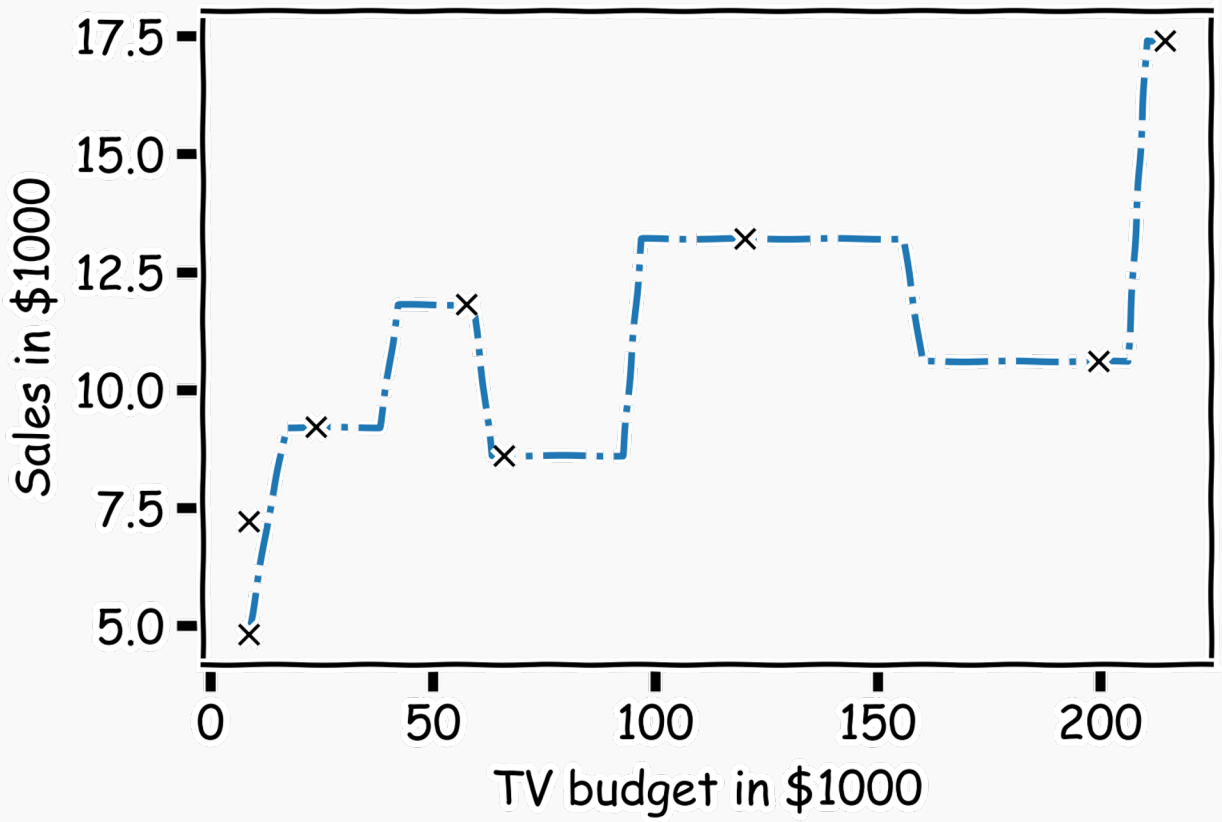
Find distances to all other points $D(x_q, x_i)$

Find the nearest neighbor, (x_p, y_p)

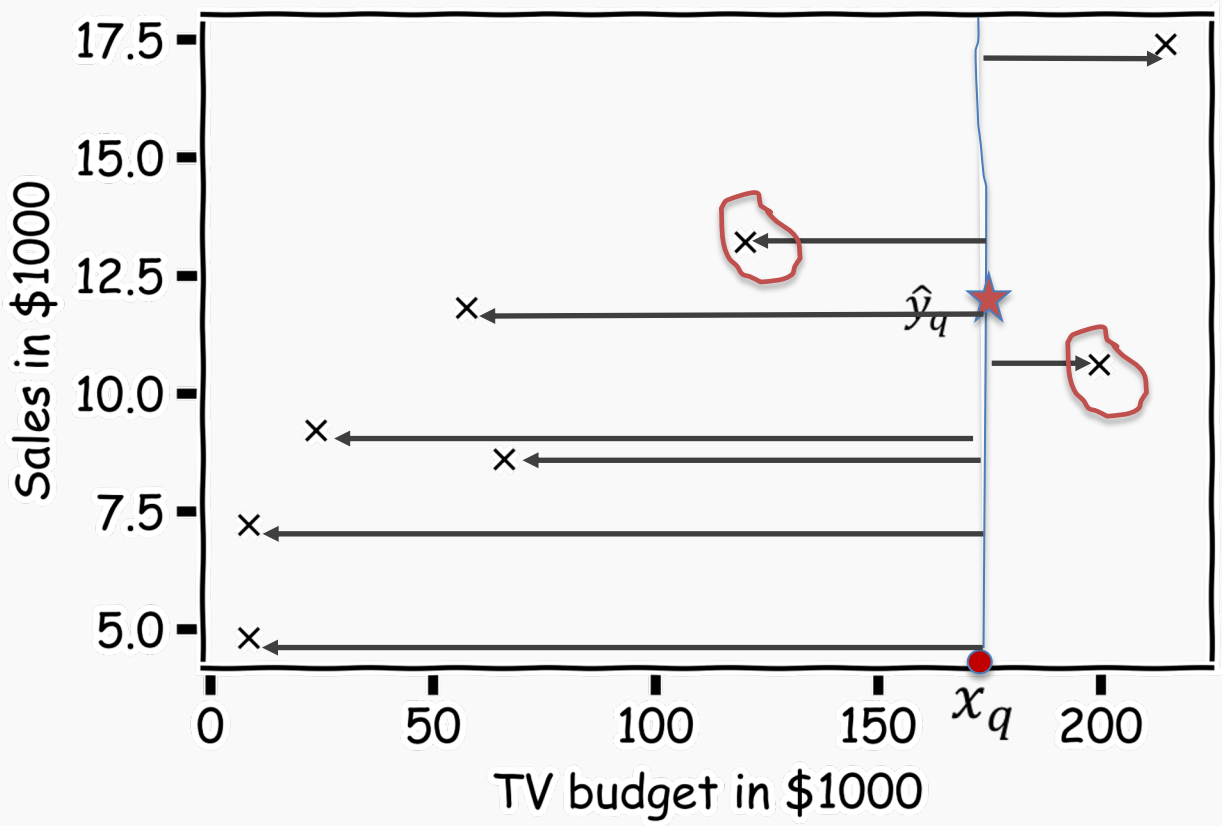
Predict $\hat{y}_q = y_p$

Simple Prediction Model

Do the same for "all" $x's$



Extend the Prediction Model



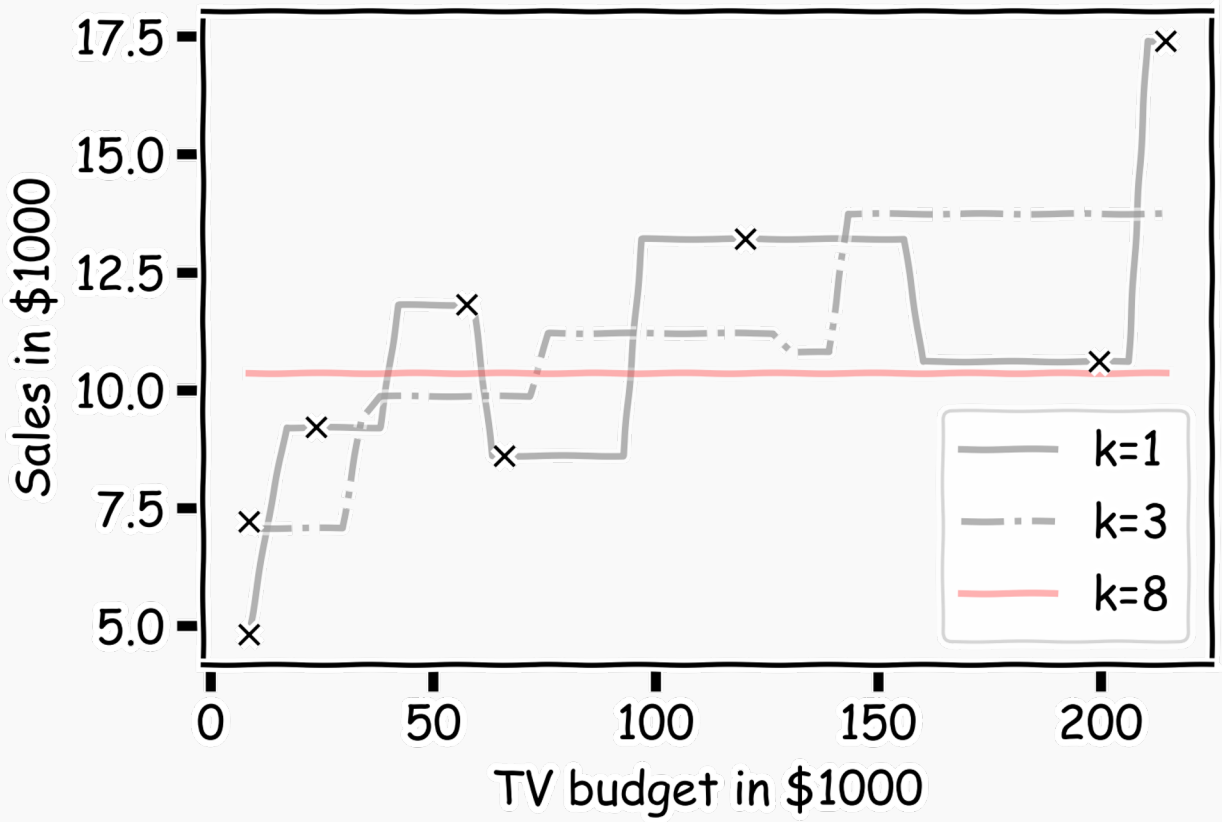
What is $\hat{y}_q$ at some x_q?

Find distances to all other points $D(x_q, x_i)$

Find the k-nearest neighbors, $x_{q_1}, \ldots, x_{q_k}$

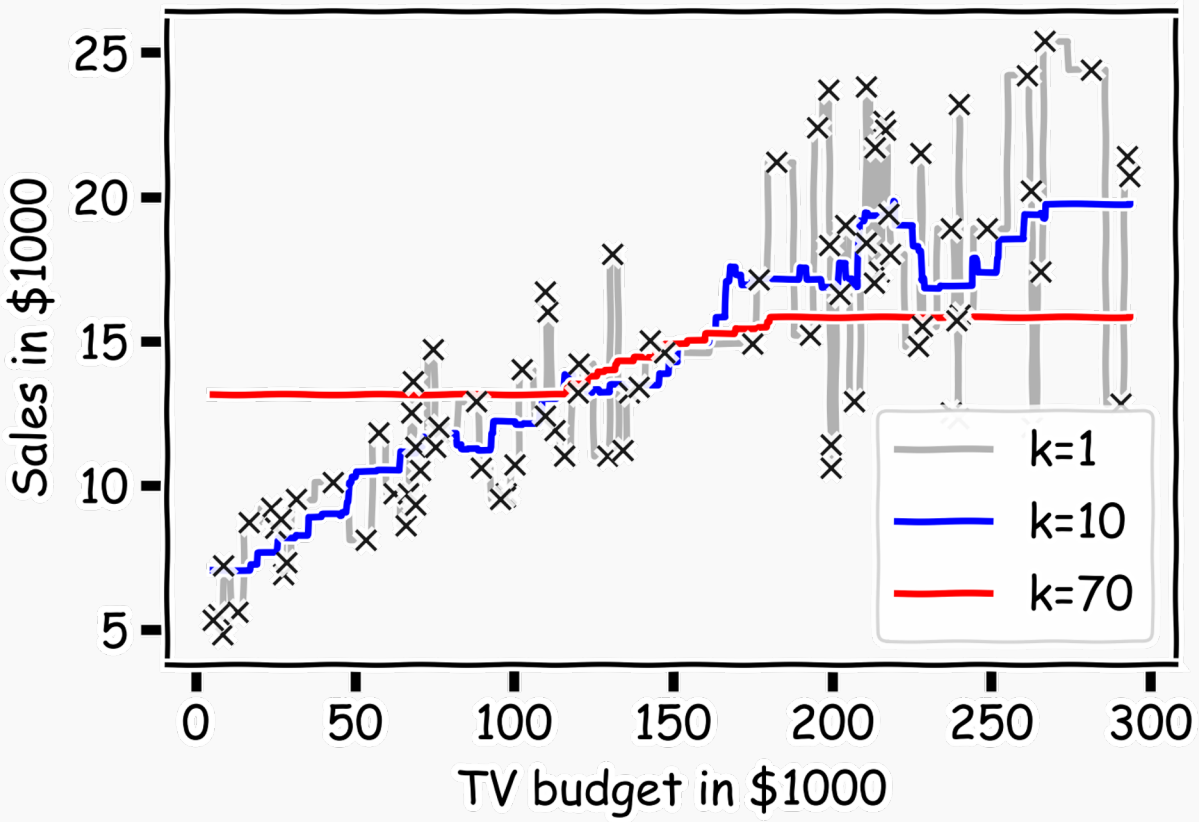
Predict $\widehat{y_q} = \frac{1}{k}\sum_i^k y_{q_i}$

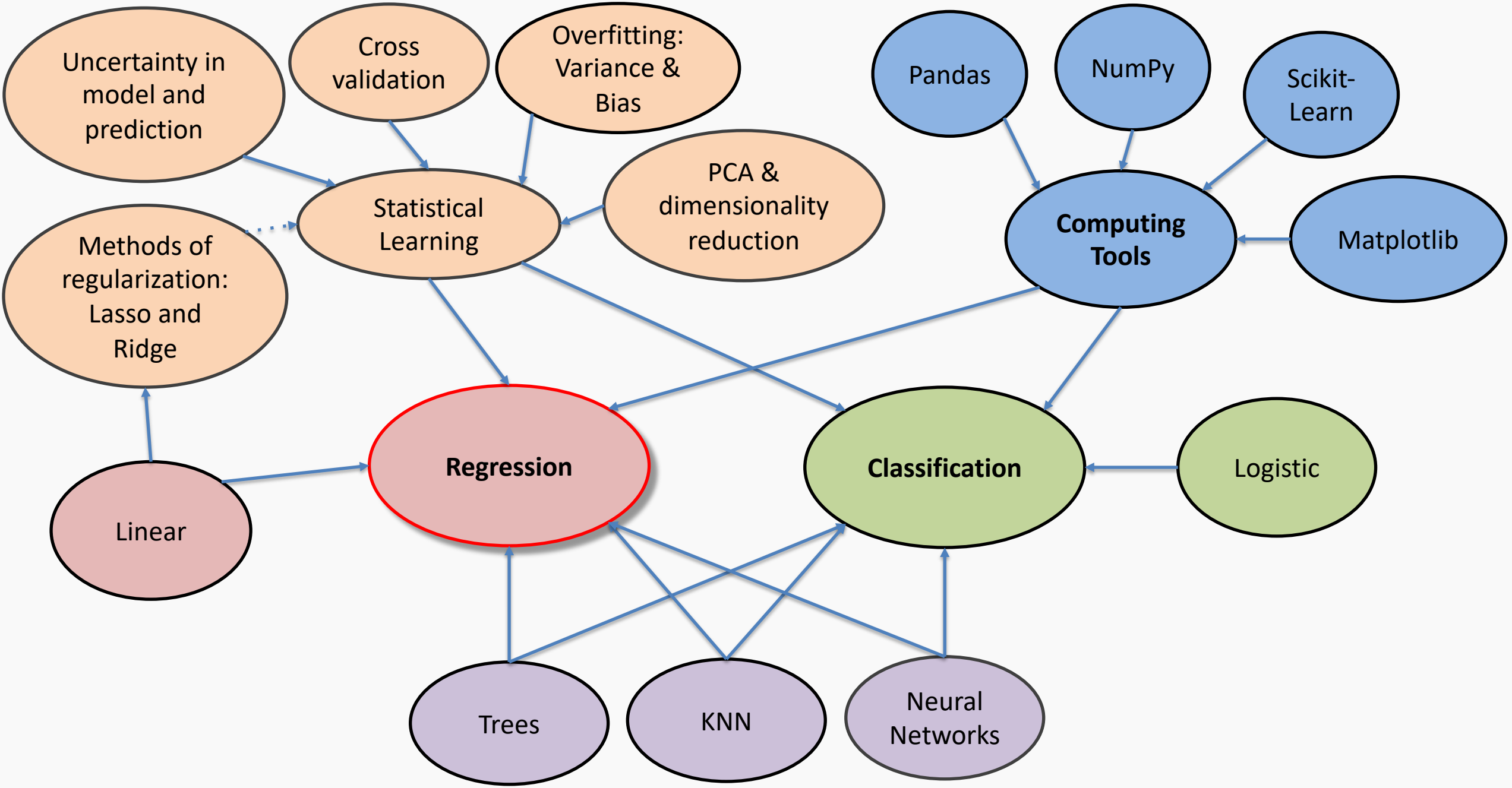
Simple Prediction Models

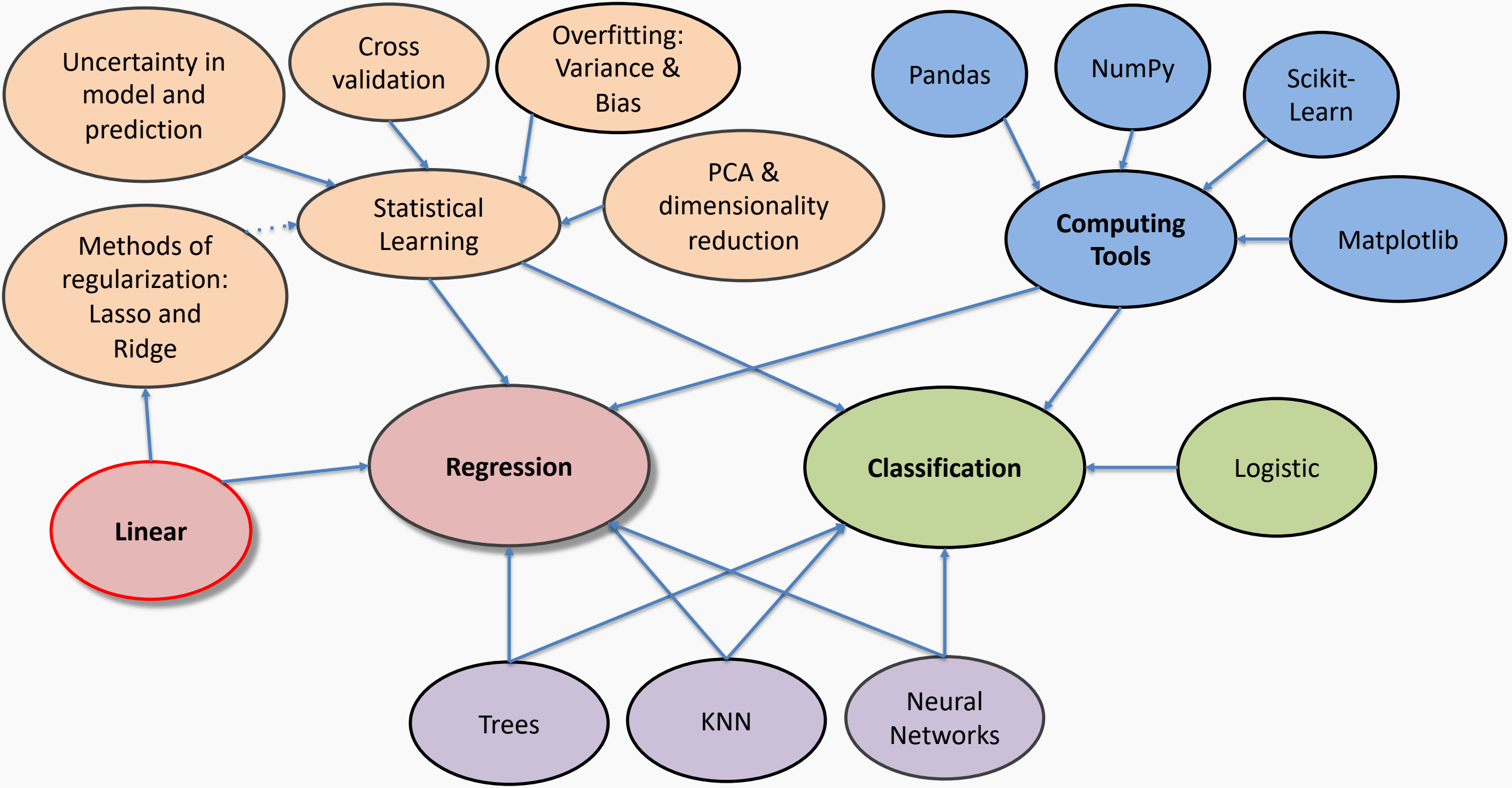


Simple Prediction Models

We can try different k-models on more data

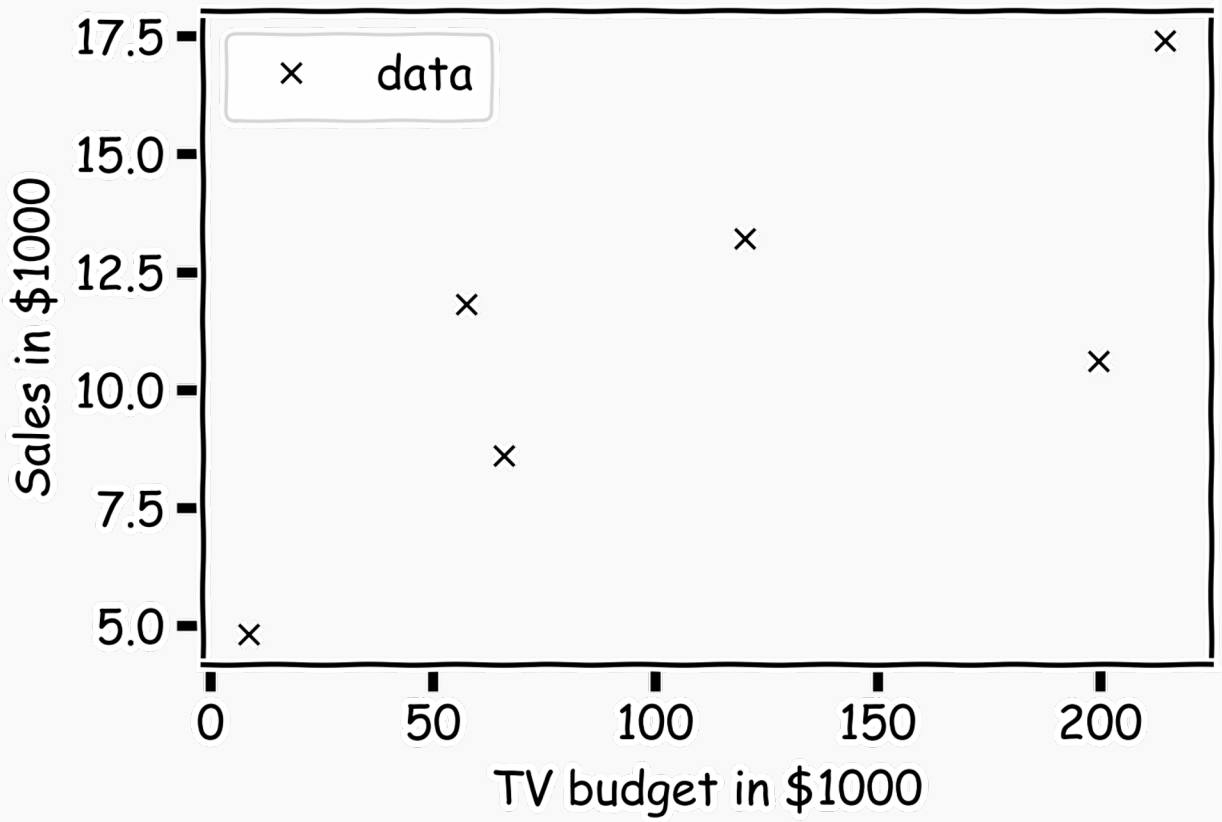



Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks


Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks

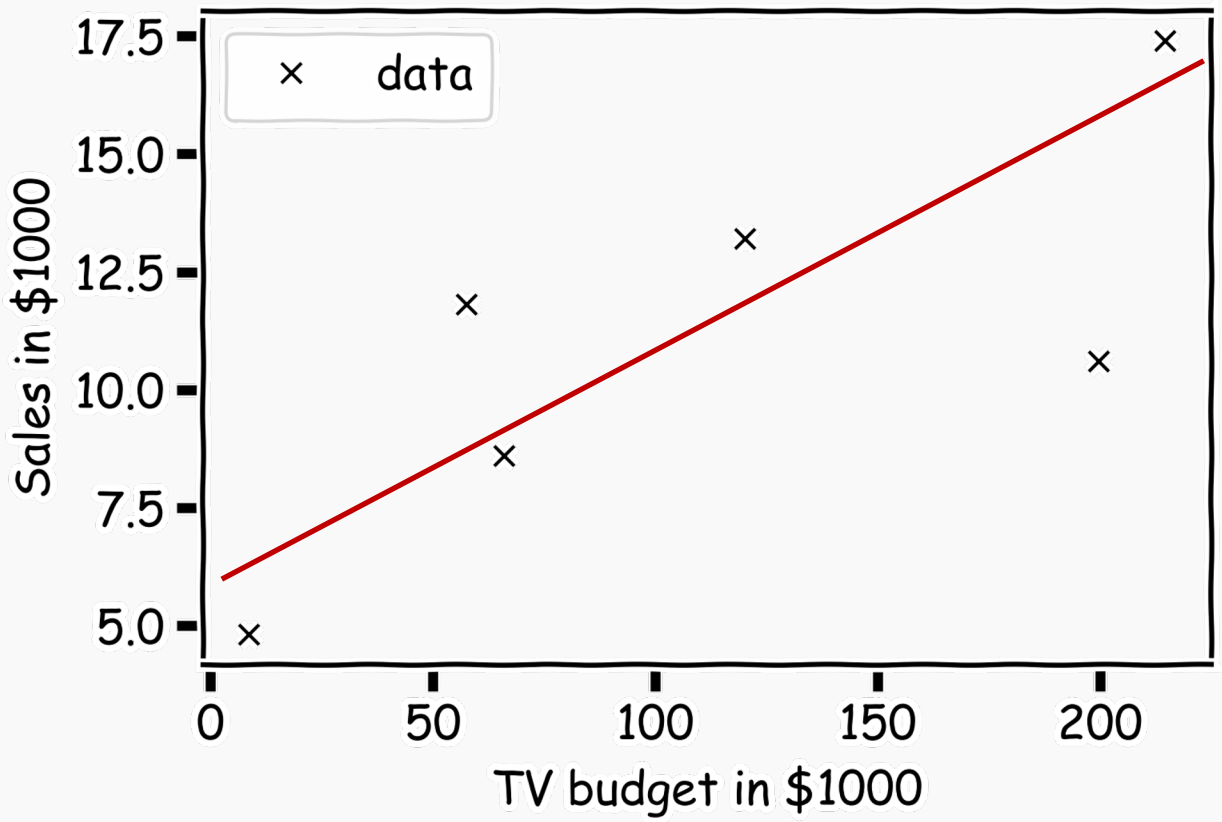
Estimate of the regression coefficients

For a given data set



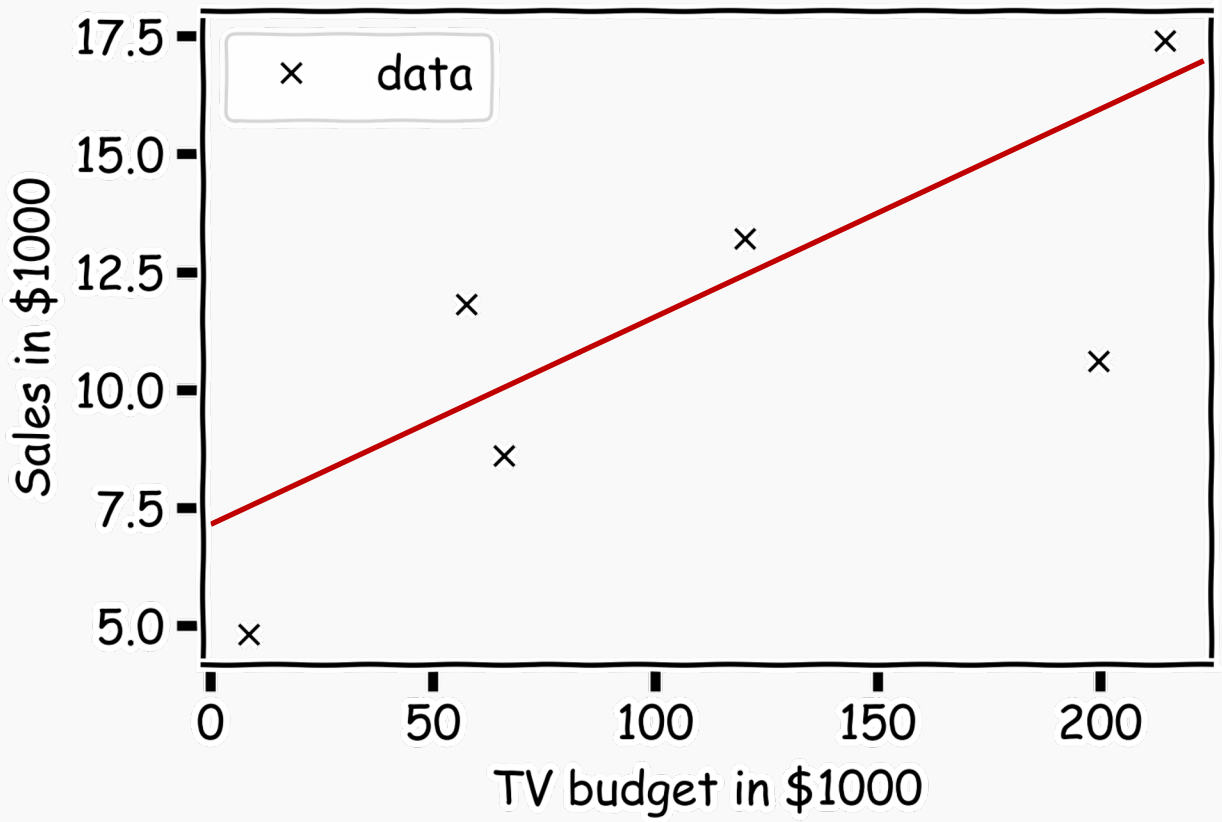
Estimate of the regression coefficients (cont)

Is this line good?



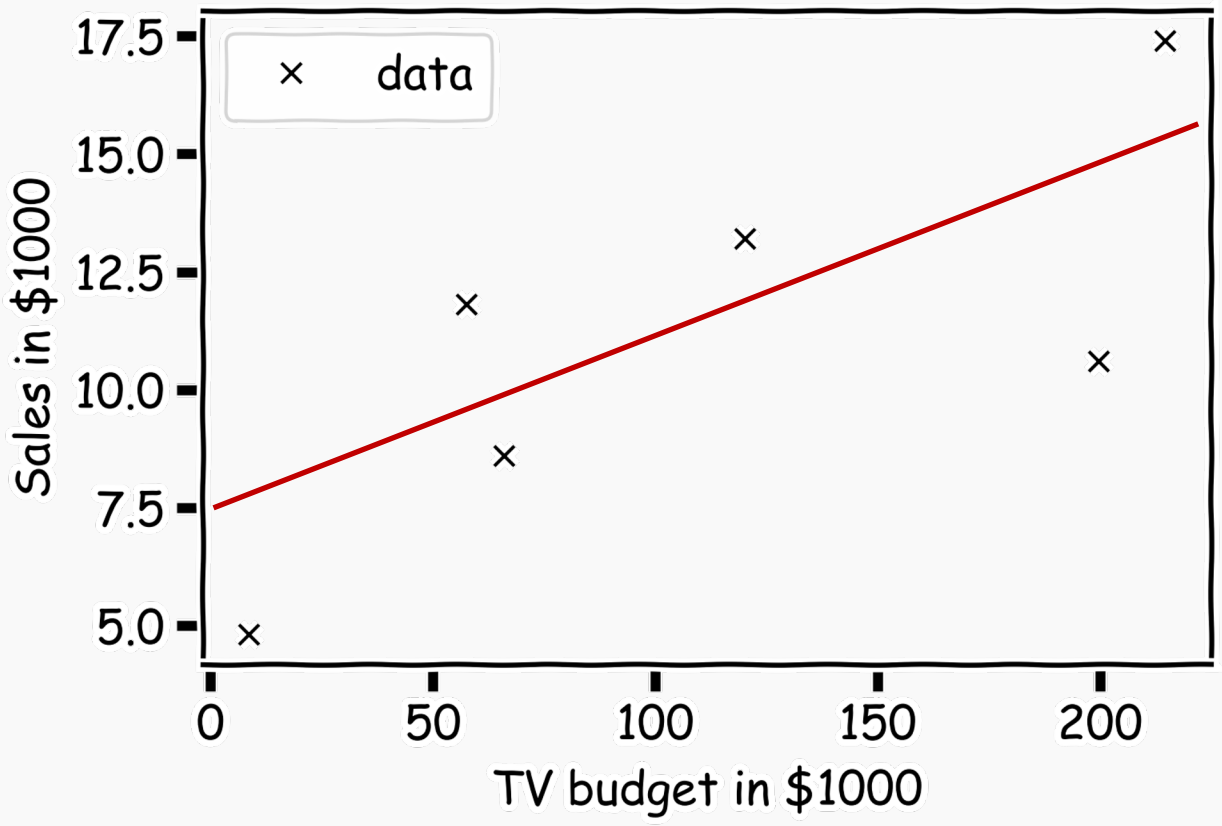
Estimate of the regression coefficients (cont)

Maybe this one?



Estimate of the regression coefficients (cont)

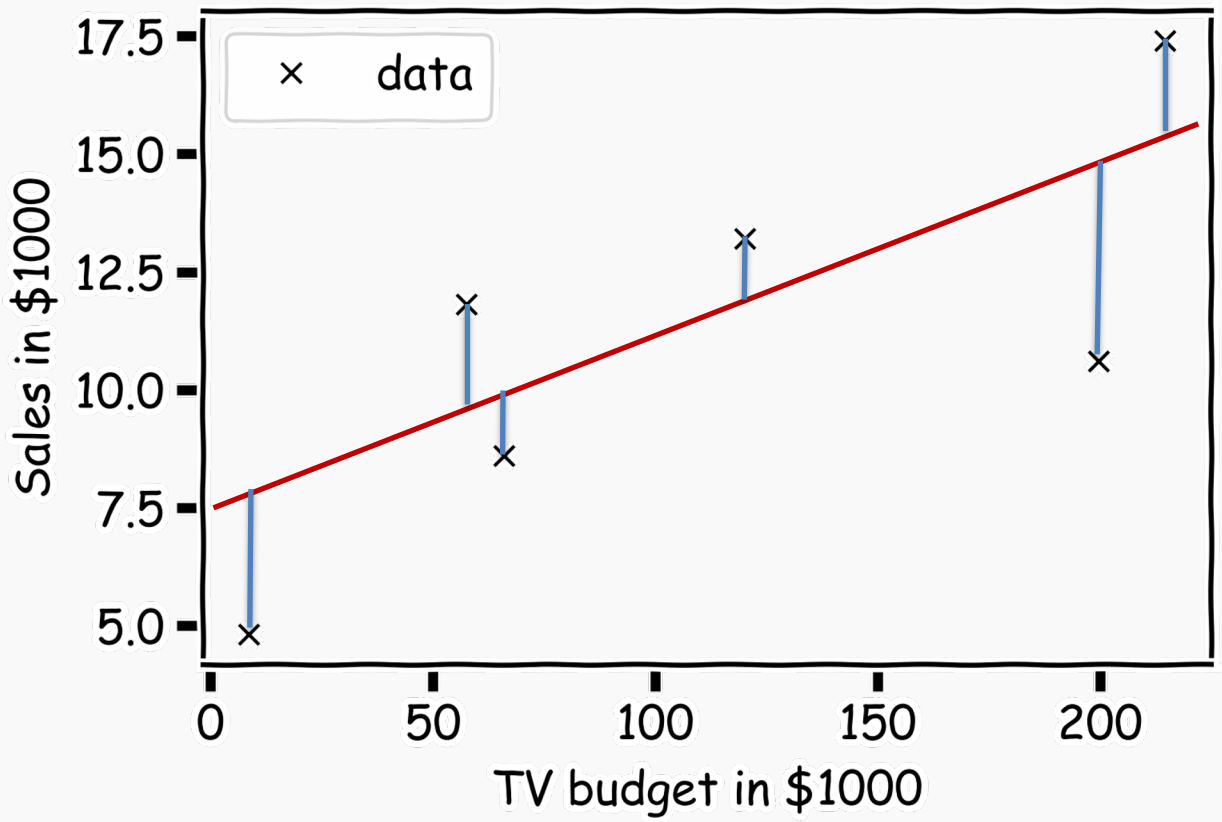
Or this one?



Estimate of the regression coefficients (cont)

Question: Which line is the best?

First calculate the residuals



Estimate of the regression coefficients (cont)

Again we use MSE as our **loss function**,

$$L(\beta_0, \beta_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}[y_i - (\beta_1 X + \beta_0)]^2 .$$

We choose $\hat{\beta}_1$ and $\hat{\beta}_0$ in order to minimize the predictive errors made by our model, i.e. minimize our loss function.

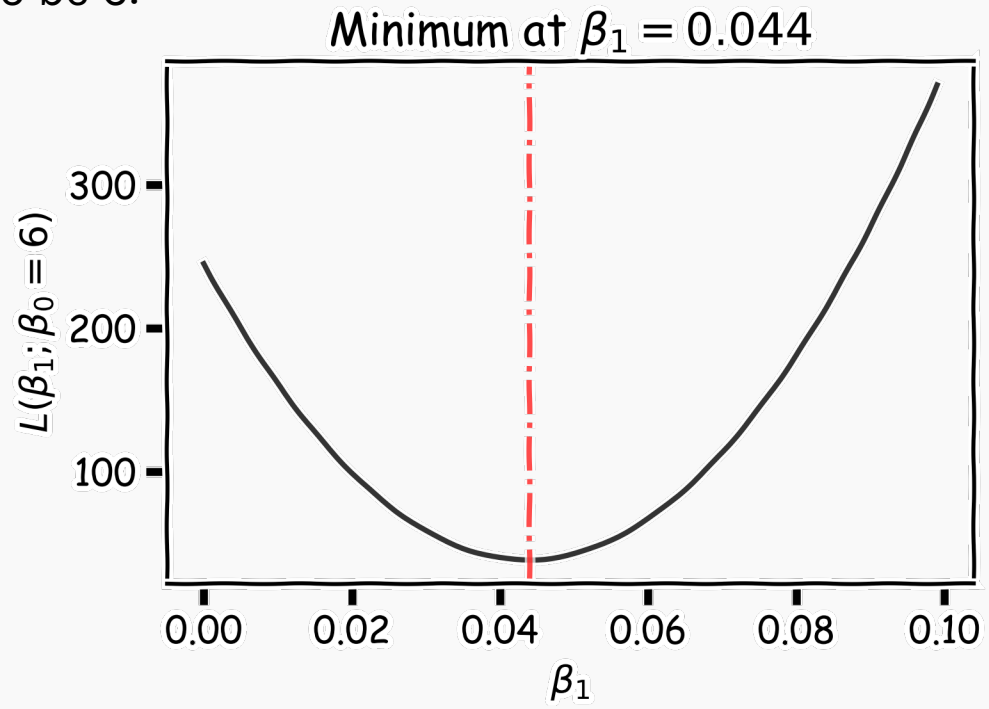
Then the optimal values for $\hat{\beta}_0$ and $\hat{\beta}_1$ should be:

$$\widehat{\beta}_0, \widehat{\beta}_1 = \underset{\beta_0, \beta_1}{\operatorname{argmin}} L(\beta_0, \beta_1).$$

Estimate of the regression coefficients: brute force

A way to estimate is to calculate the loss function for every possible beta_0 and beta_1. Then select the betas where the loss function is at the minimum.

E.g. the loss function for different beta_1s when beta_0 is fixed to be 6:



Estimate of the regression coefficients: exact method

Take the partial derivatives of L with respect to β_0 and β_1, set to zero, and find the solution to that equation. This procedure will give us explicit formulae for $\hat{\beta}_0$ and $\hat{\beta}_1$:

$$\hat{\beta}_1 = \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sum_i (x_i - \overline{x})^2}$$

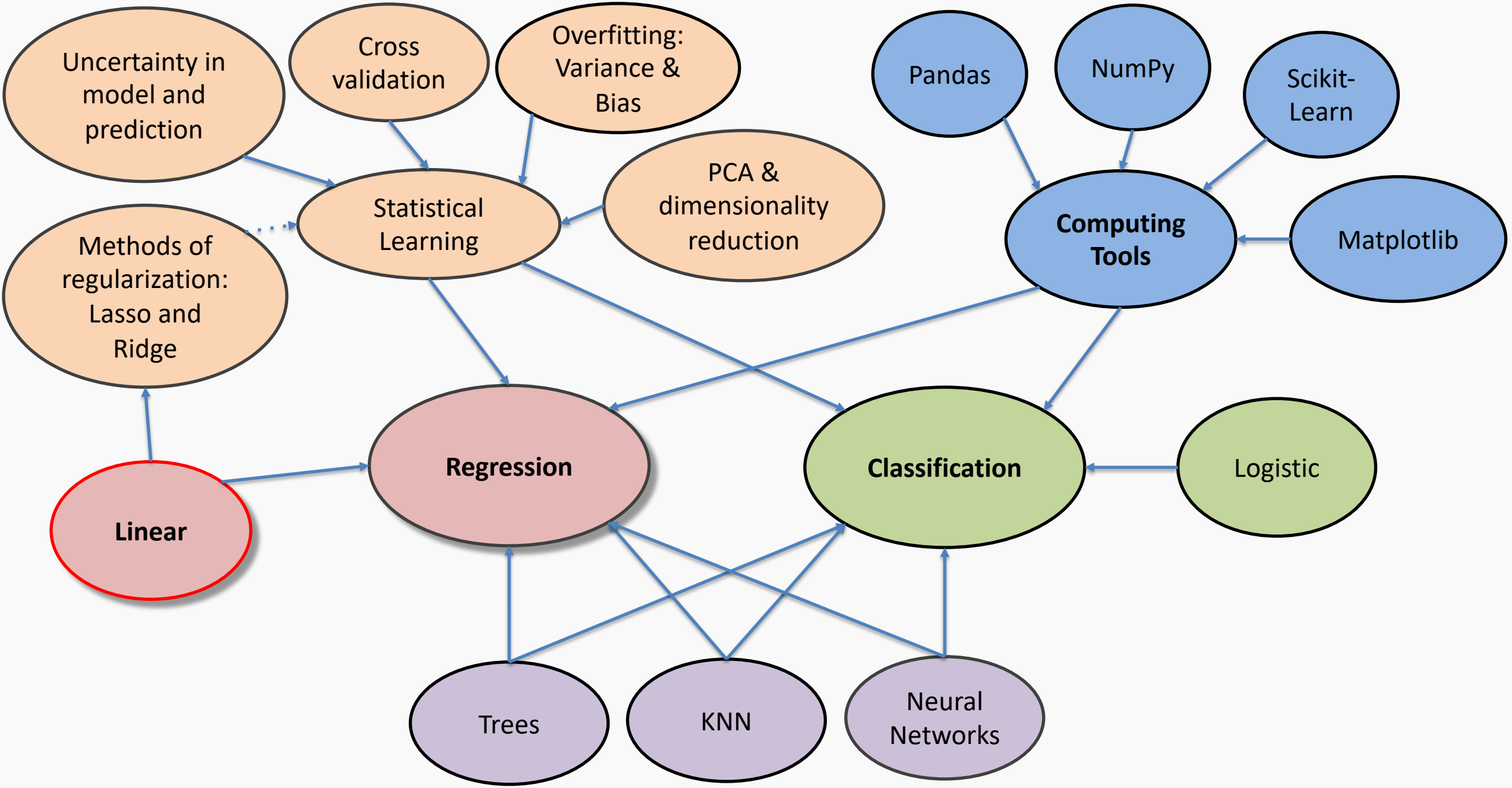
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

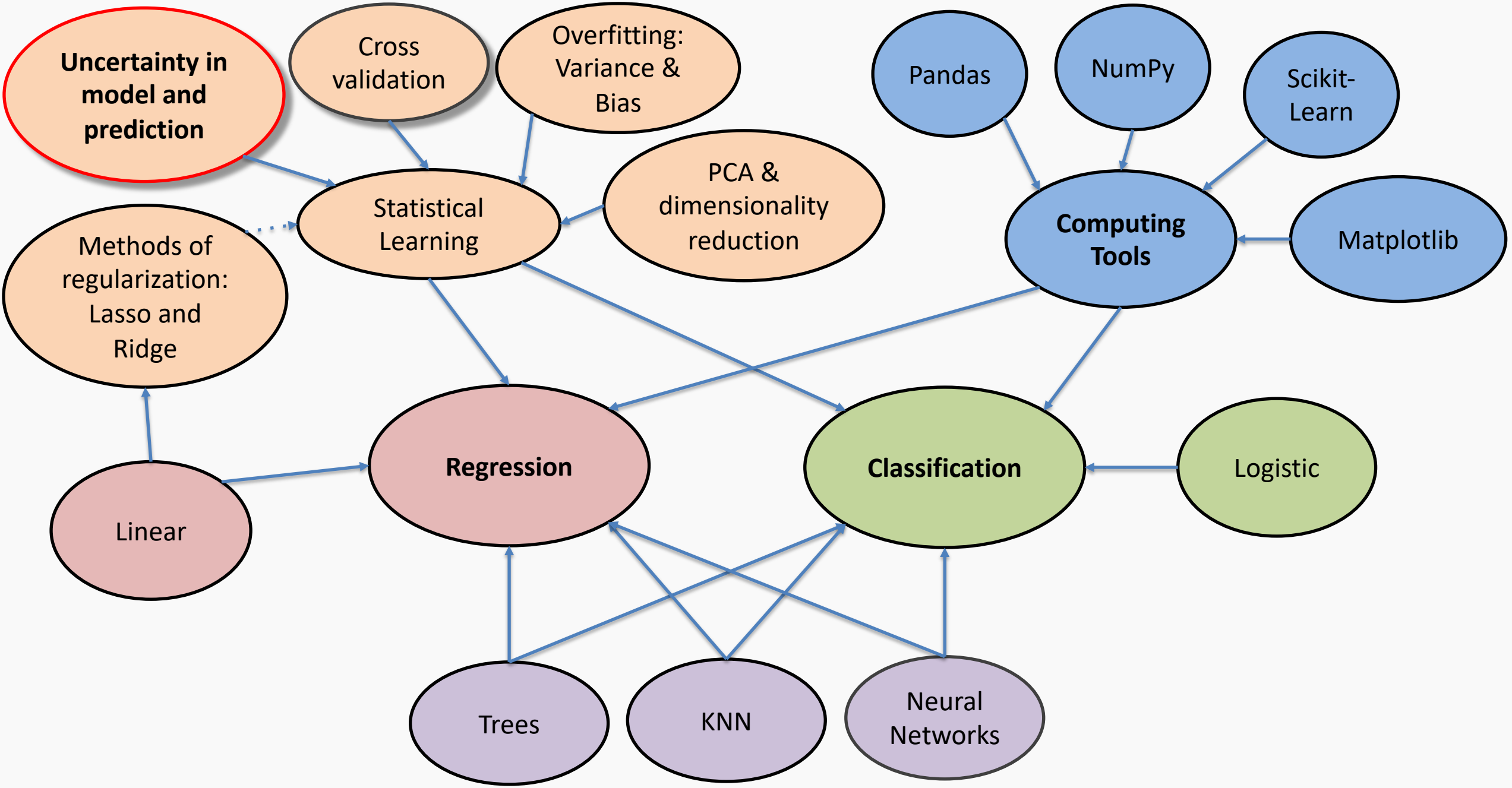
where $\bar{y}$ and $\bar{x}$ are sample means.

The line:

$$\widehat{Y} = \widehat{\beta}_1 X + \widehat{\beta}_0$$

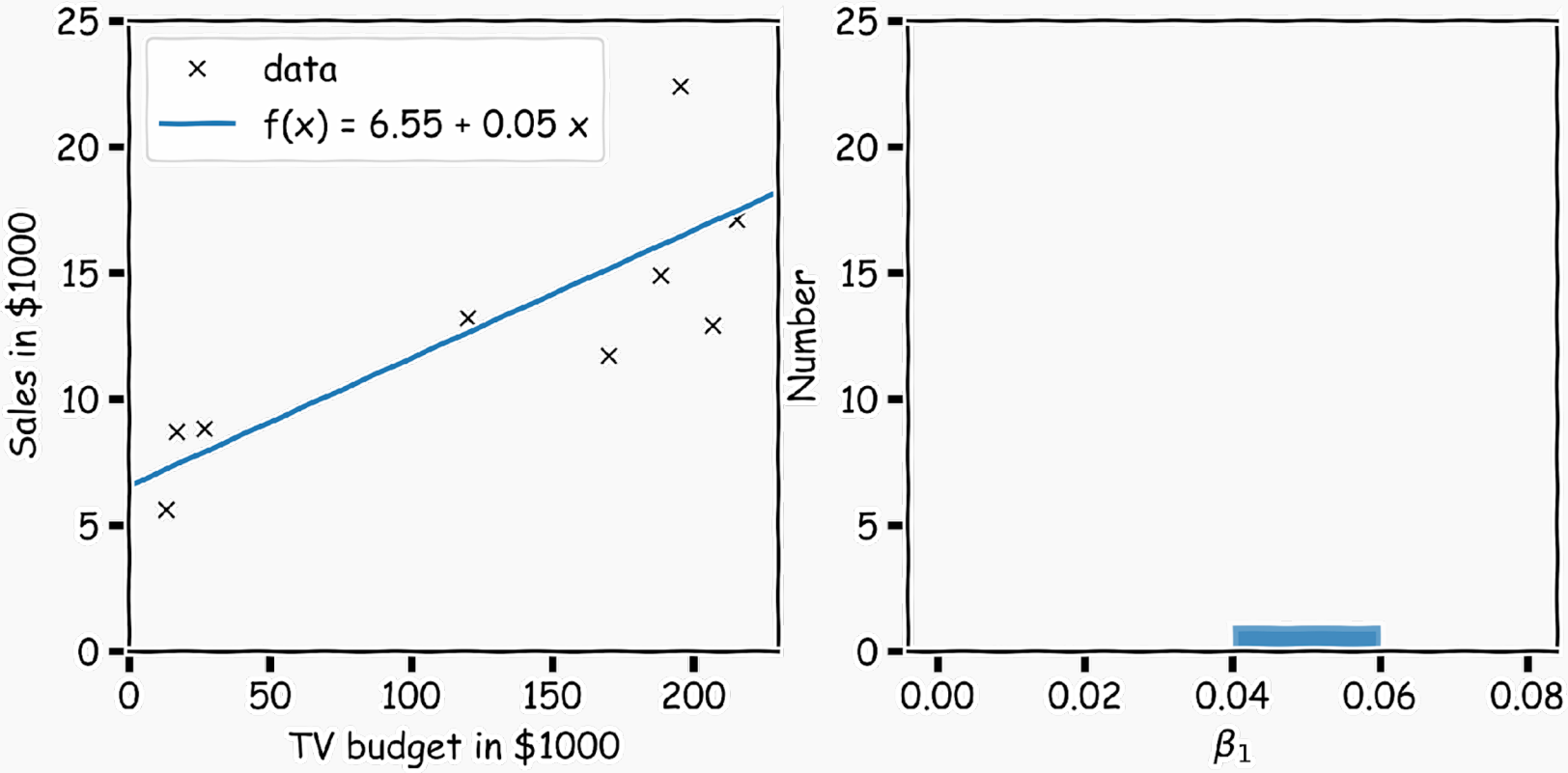
is called the **regression line**.


Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks


Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks

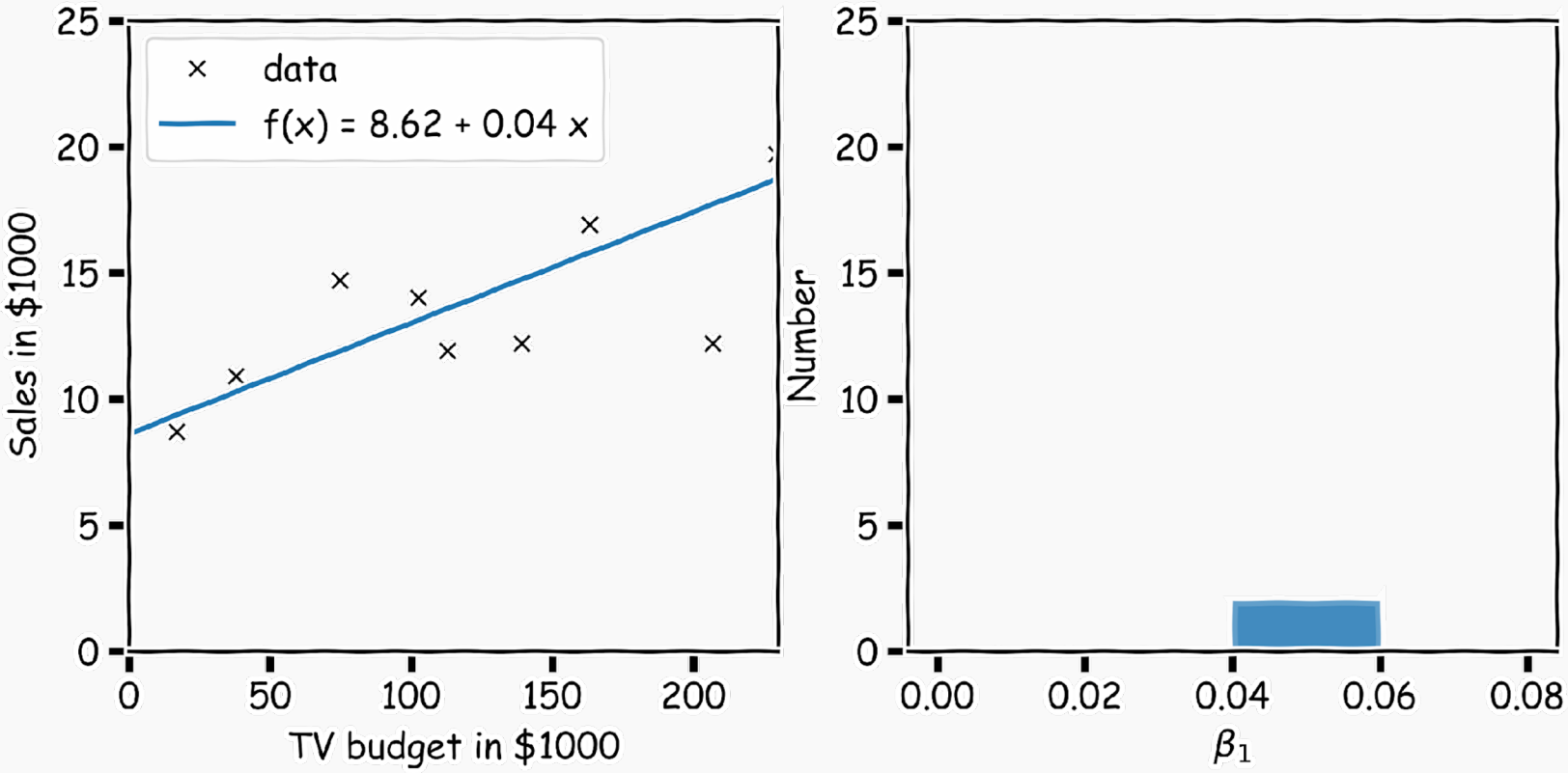
Confidence intervals for the predictors estimates (cont)

In our magical realisms, we can now sample multiple times



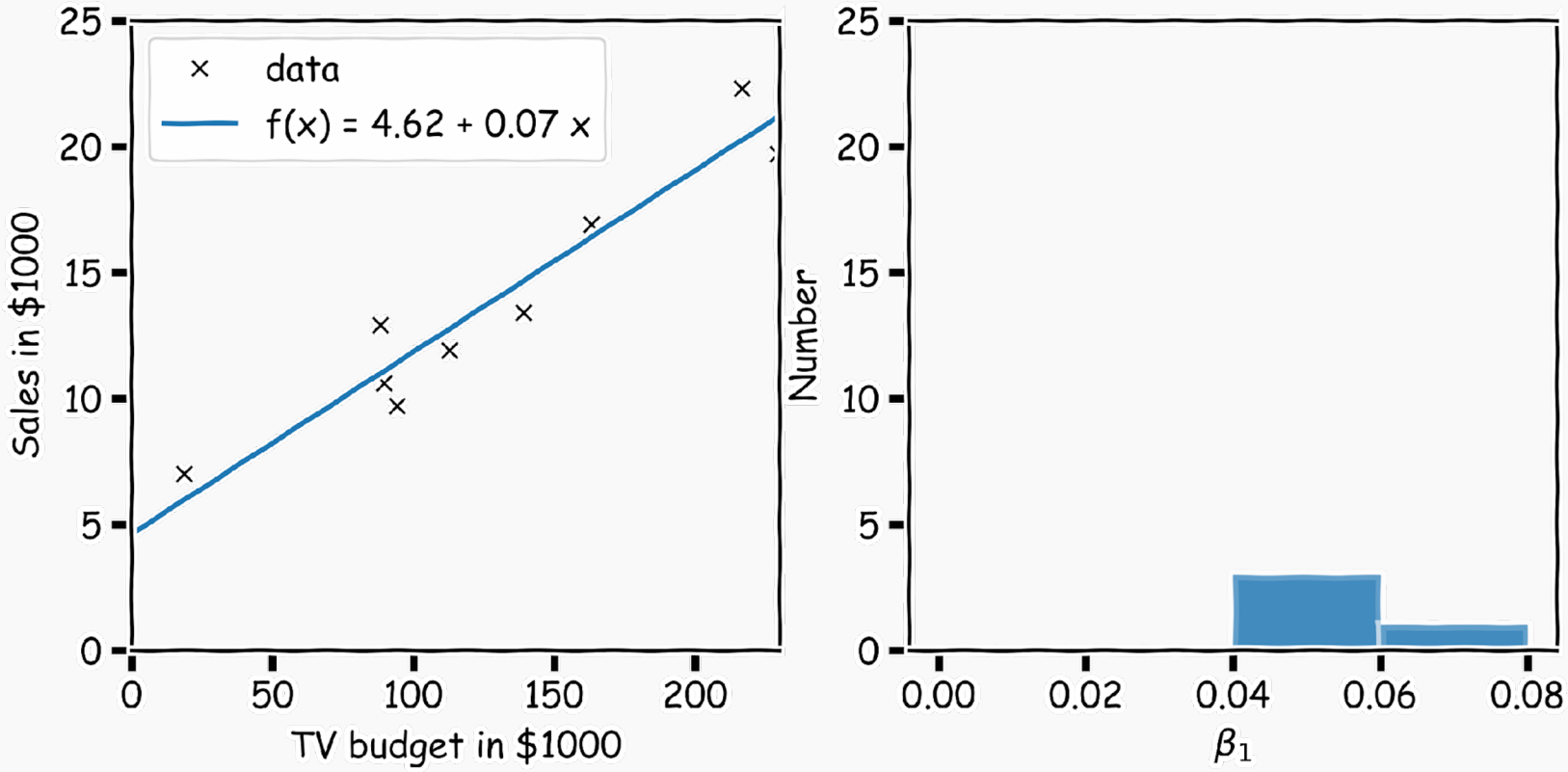
Confidence intervals for the predictors estimates (cont)

Another sample



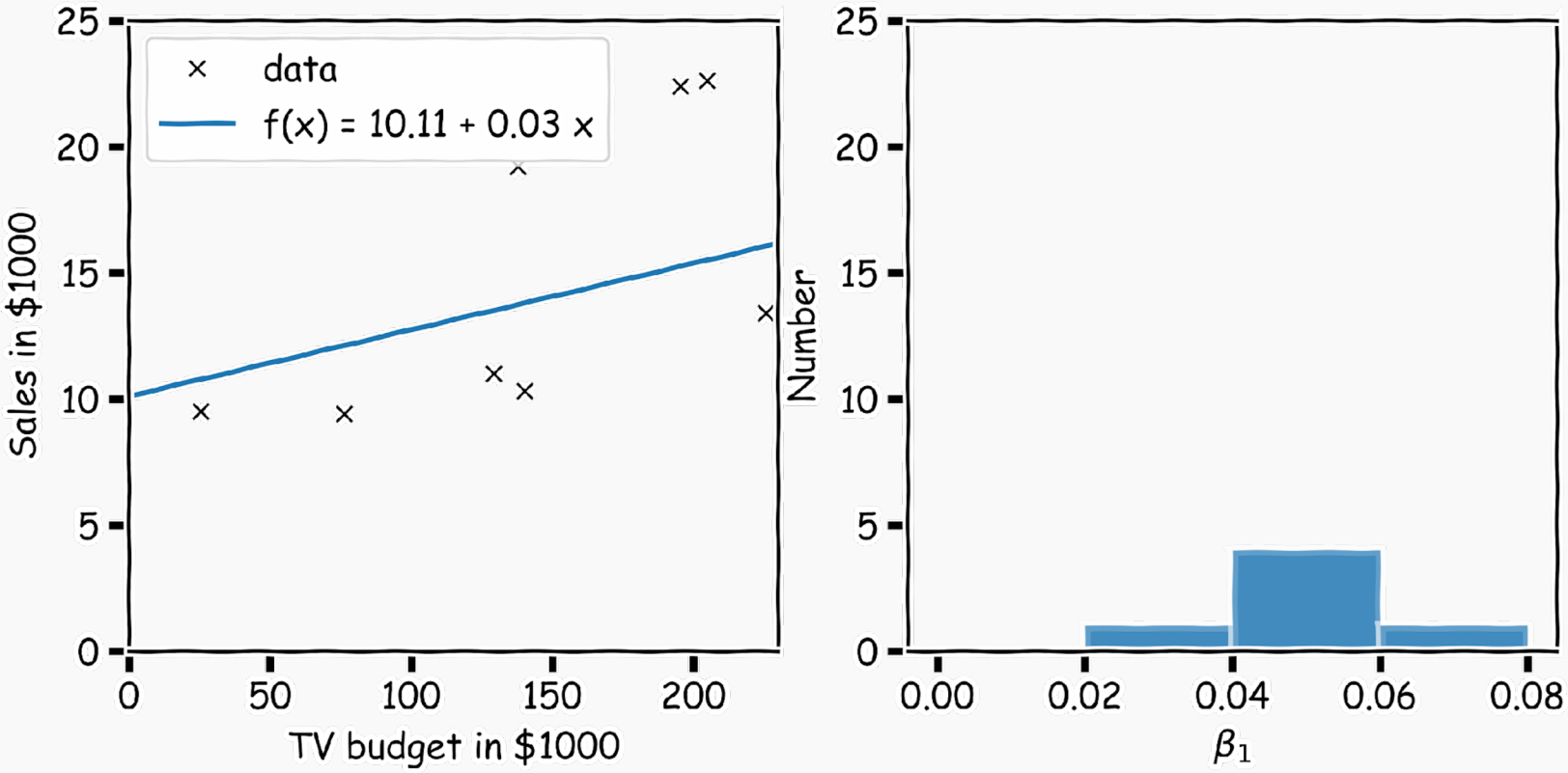
Confidence intervals for the predictors estimates (cont)

Another sample



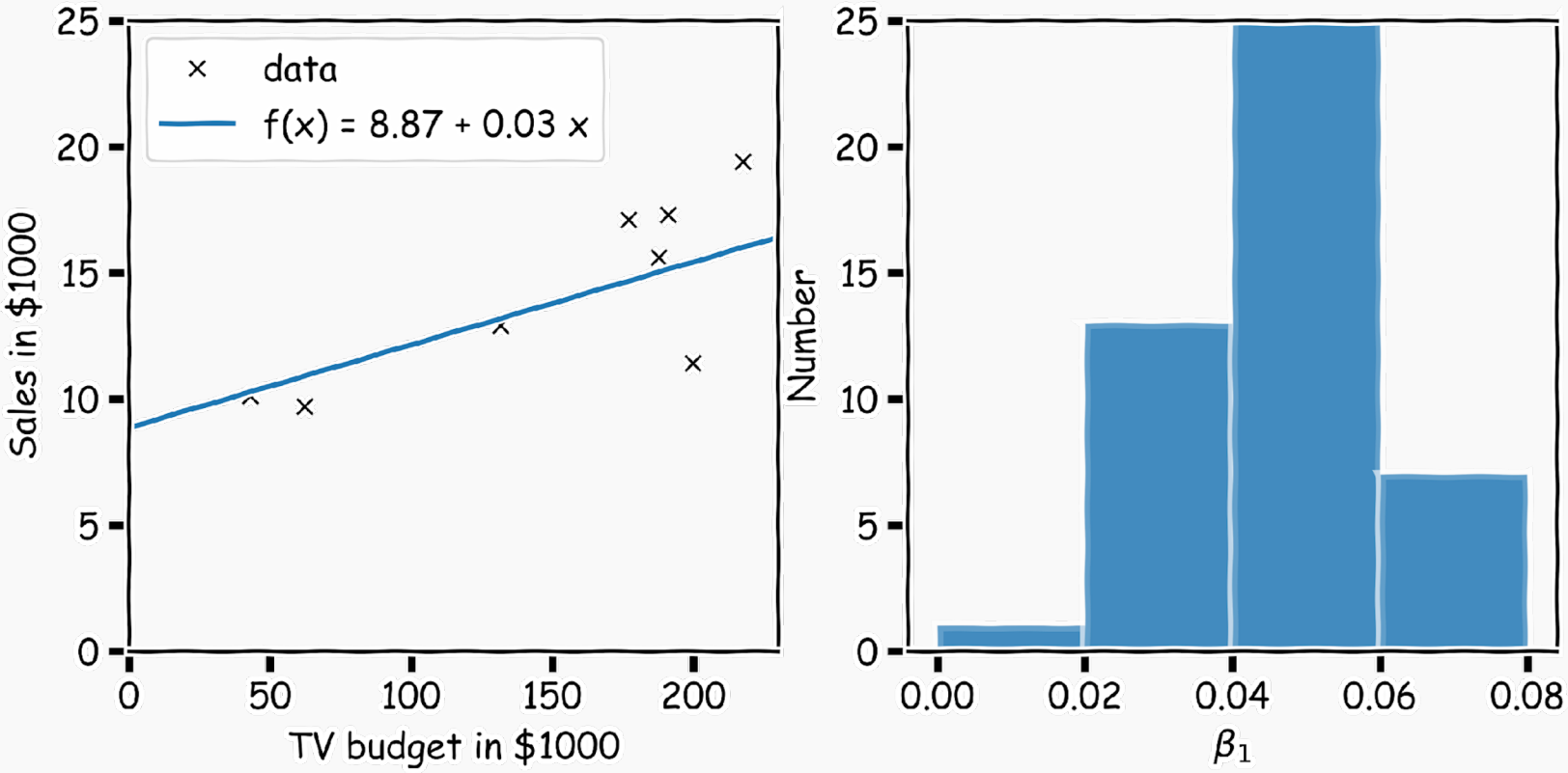
Confidence intervals for the predictors estimates (cont)

And another sample



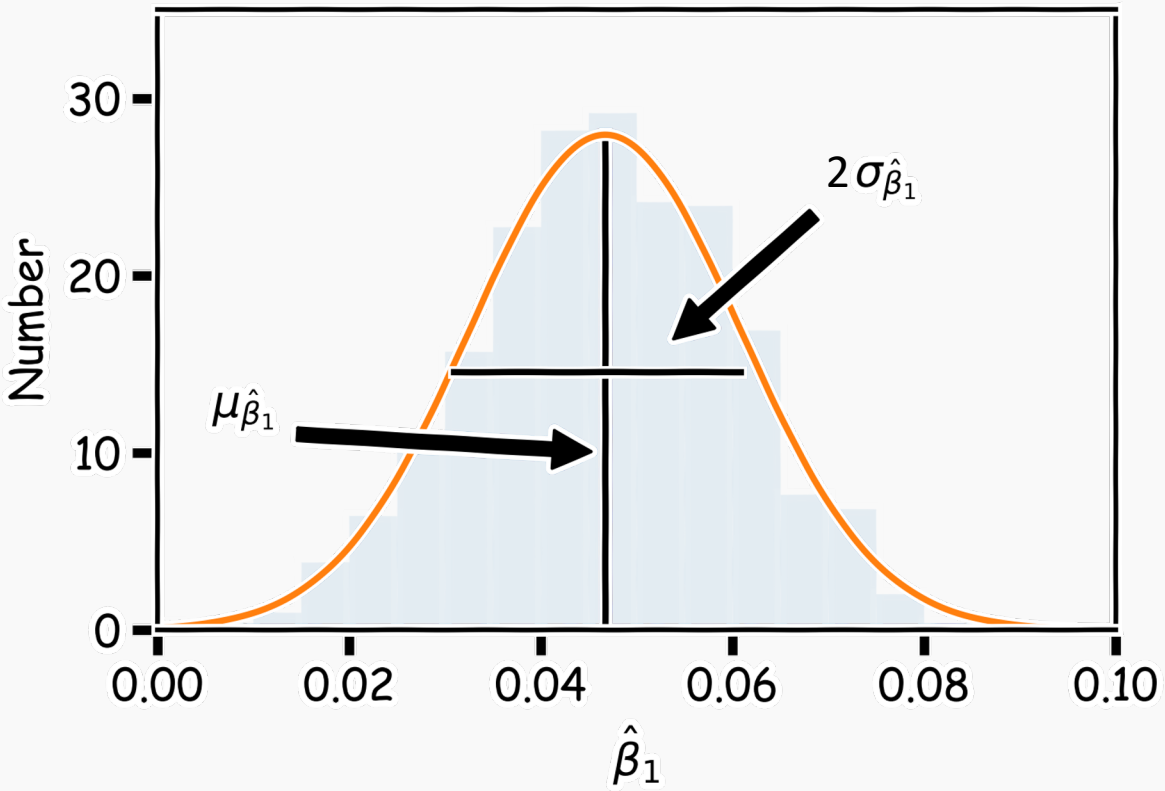
Confidence intervals for the predictors estimates (cont)

Repeat this for 100 times



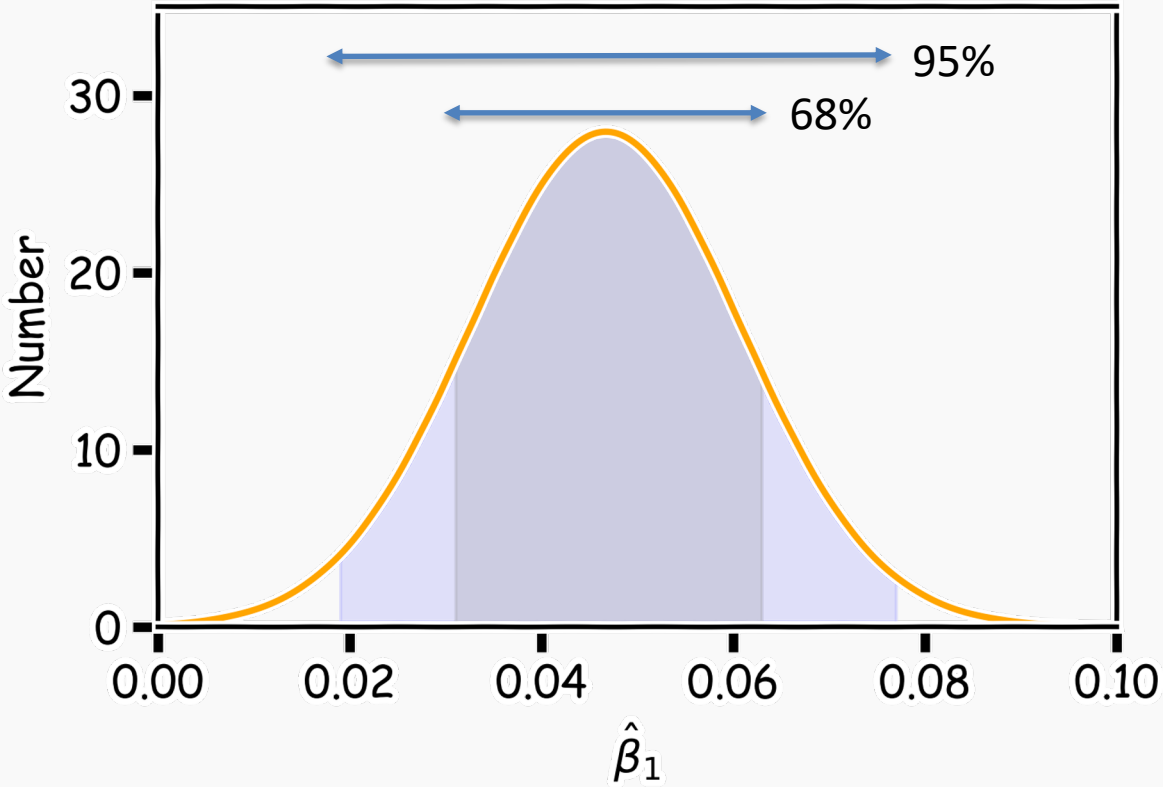
Confidence intervals for the predictors estimates (cont)

We can now estimate the mean and standard deviation of all the estimates $\hat{\beta}_1$.



Confidence intervals for the predictors estimates (cont)

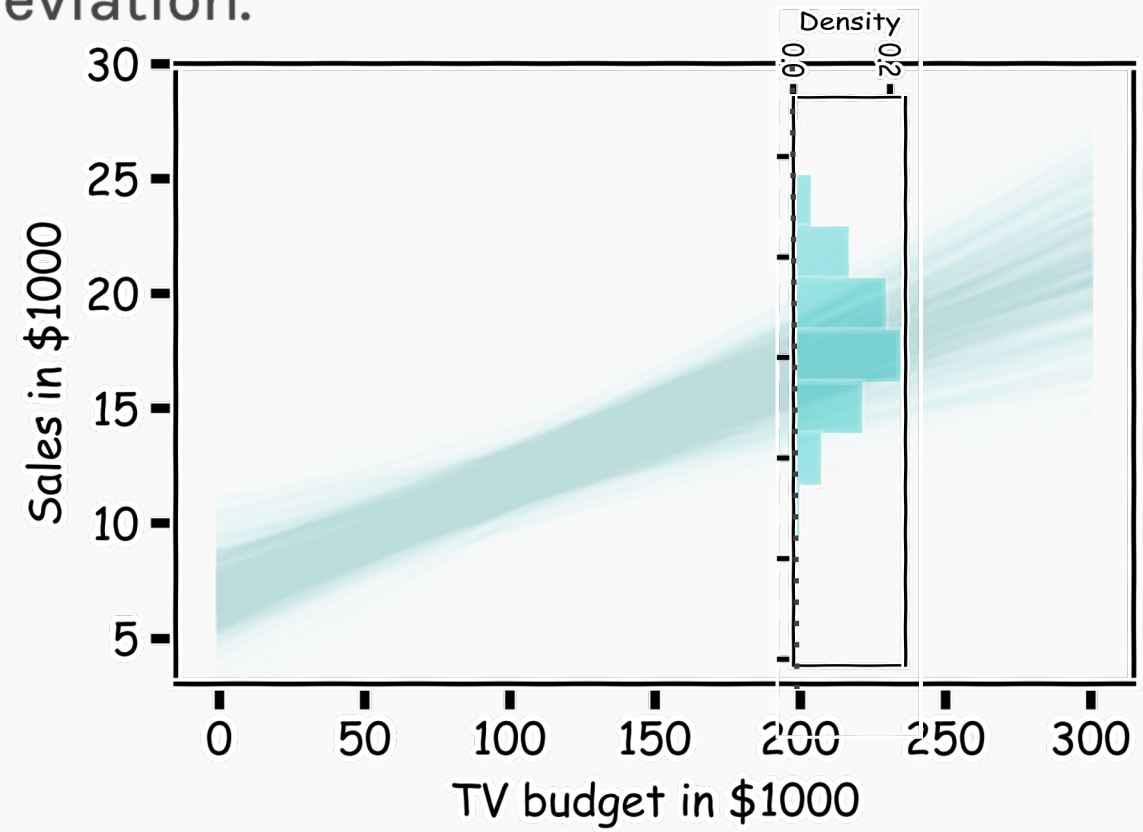
Finally we can calculate the confidence intervals, which are the ranges of values such that the true value of is contained in this interval with n percent probability.



How well do we know $\hat{f}$?

Below we show all regression lines for a thousand of such bootstrapped samples.

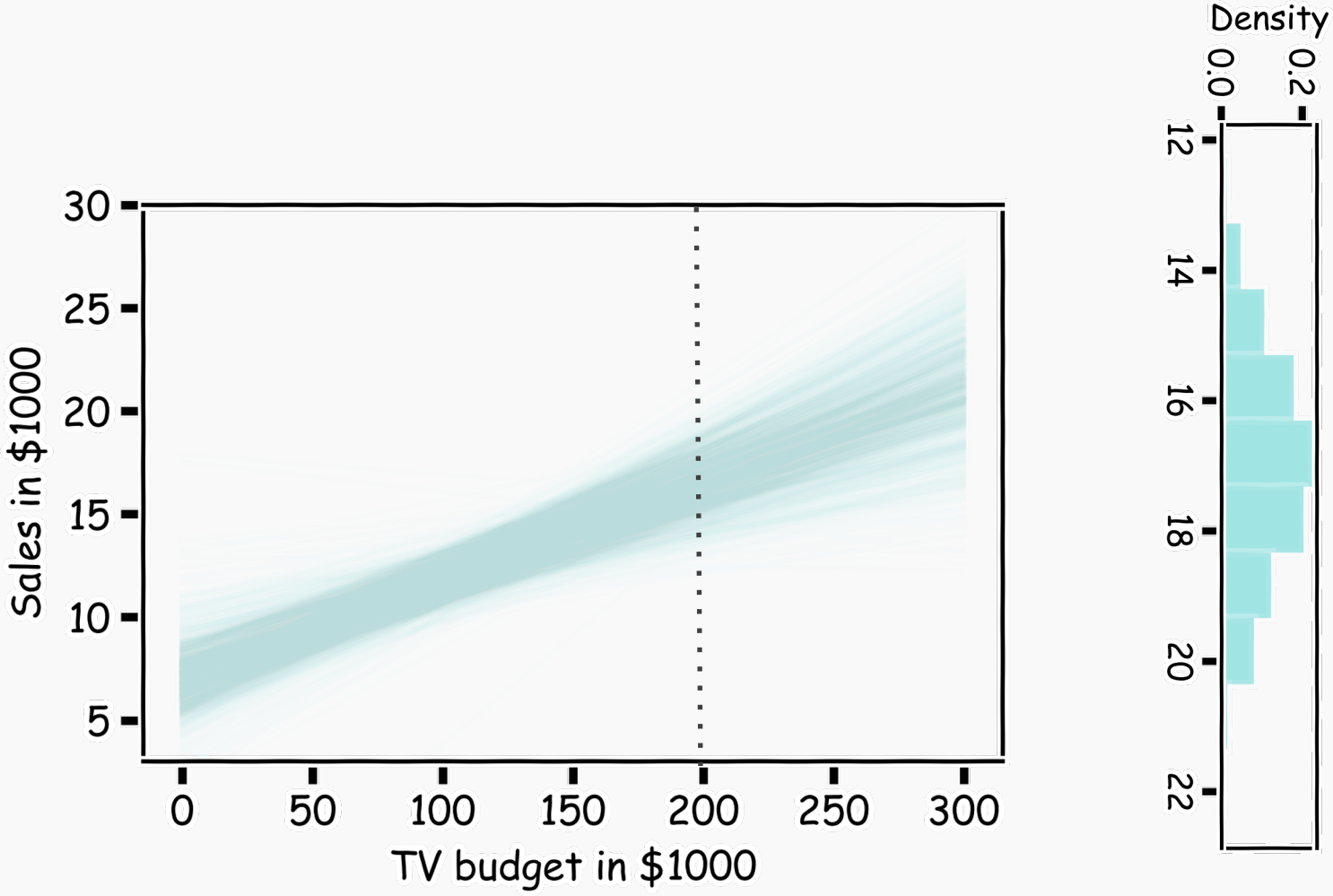
For a given x, we examine the distribution of $\hat{f}$, and determine the mean and standard deviation.



How well do we know $\hat{f}$?

Below we show all regression lines for a thousand of such sub-samples.

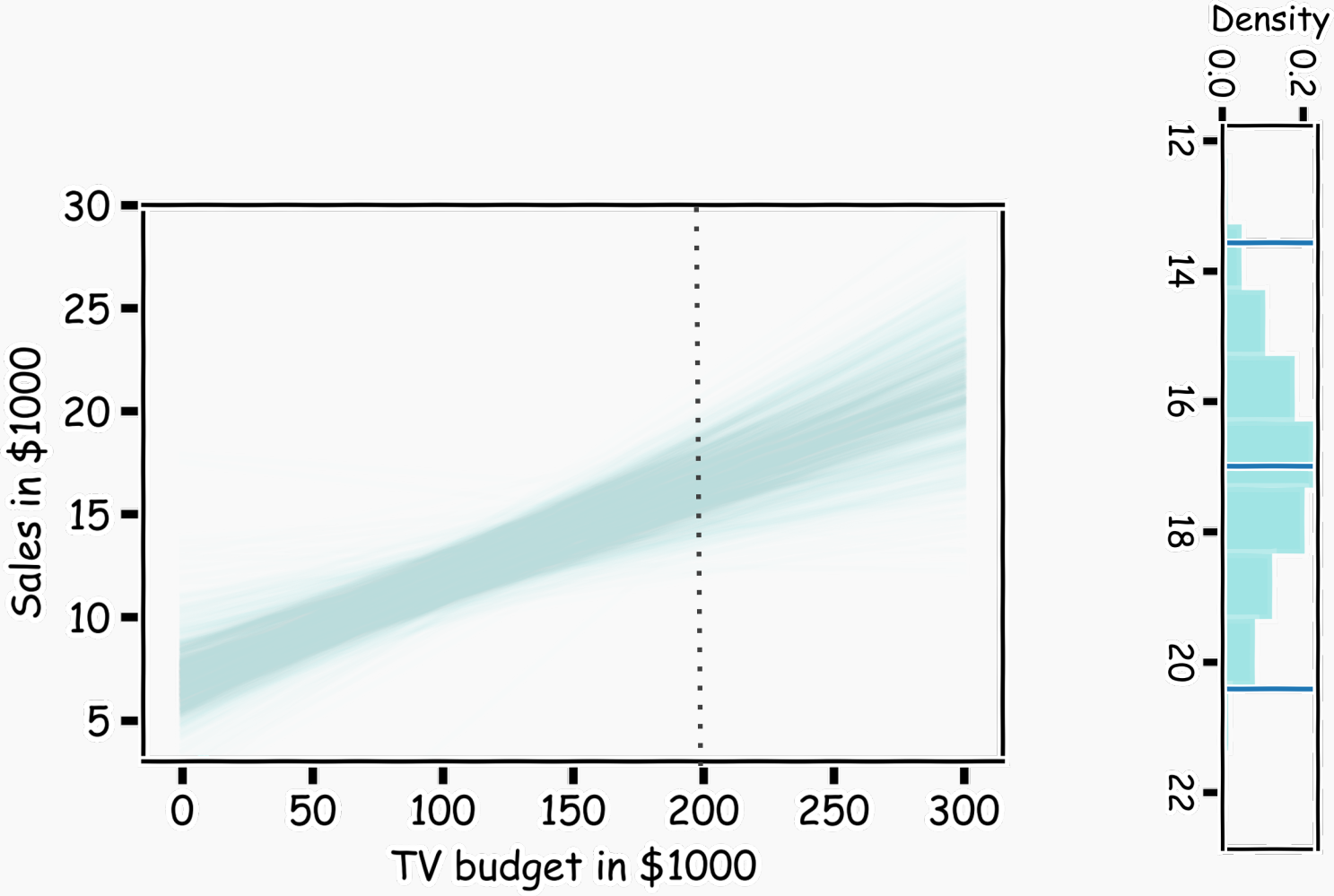
For each one of those "realizations", we could fit a model and estimate $\hat{\beta}_0$ and $\hat{\beta}_1$.



How well do we know $\hat{f}$?

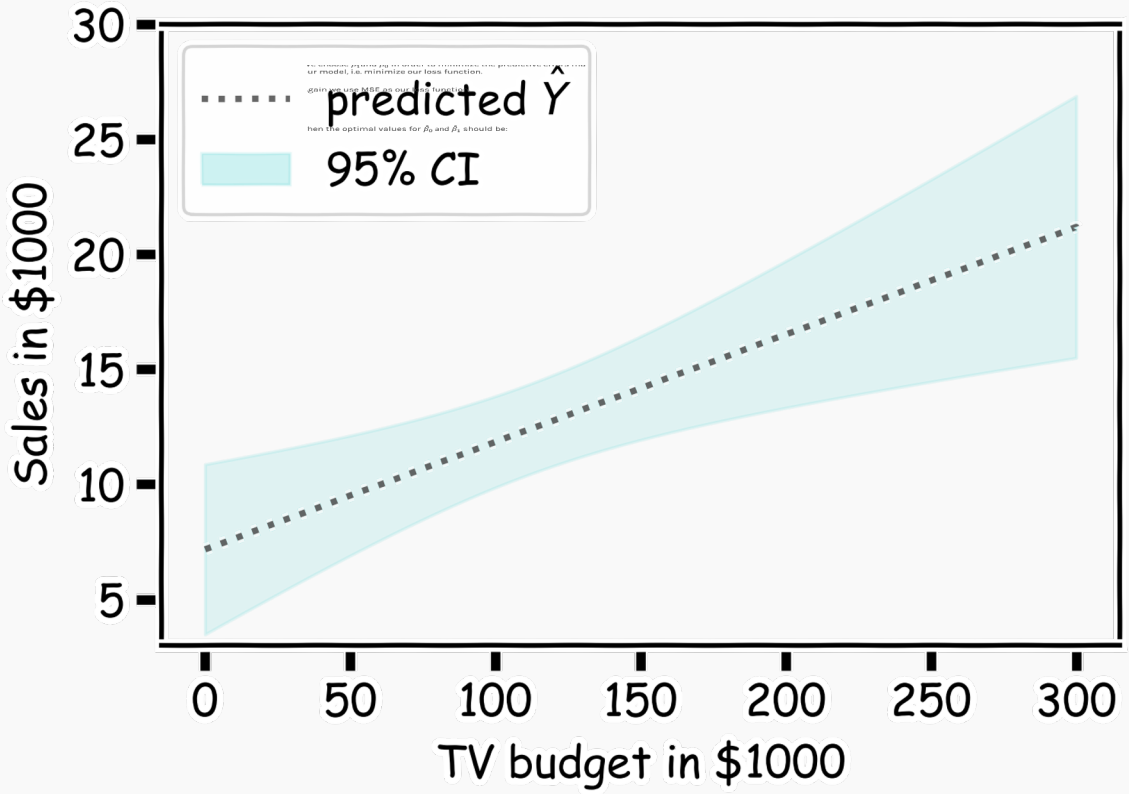
Below we show all regression lines for a thousand of such sub-samples.

For each one of those "realizations", we could fit a model and estimate $\hat{\beta}_0$ and $\hat{\beta}_1$.

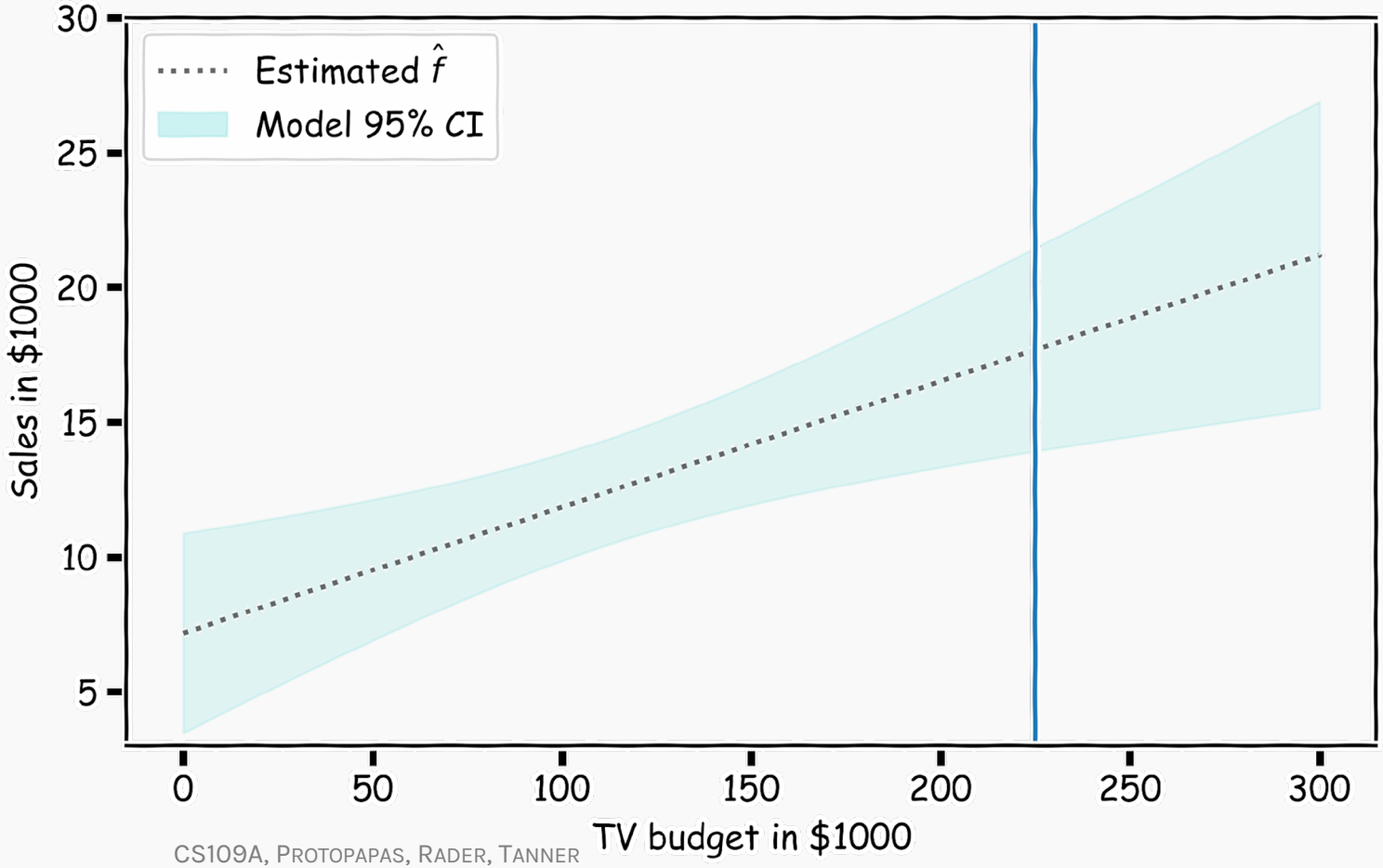


How well do we know $\hat{f}$?

For every x, we calculate the mean of the models, $\hat{f}$ (shown with dotted line) and the 95% CI of those models (shaded area).

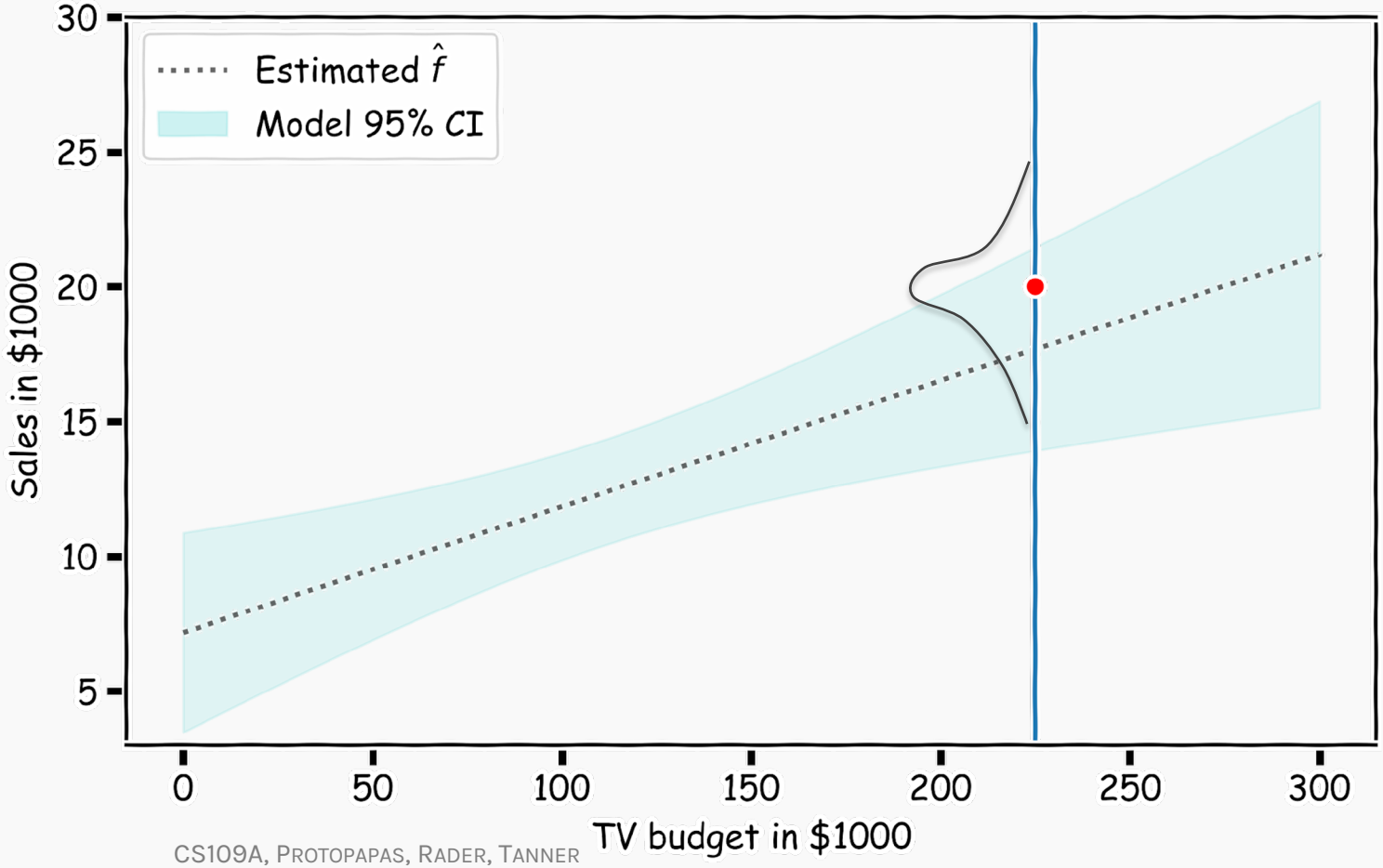


Confidence in predicting $\hat{y}$



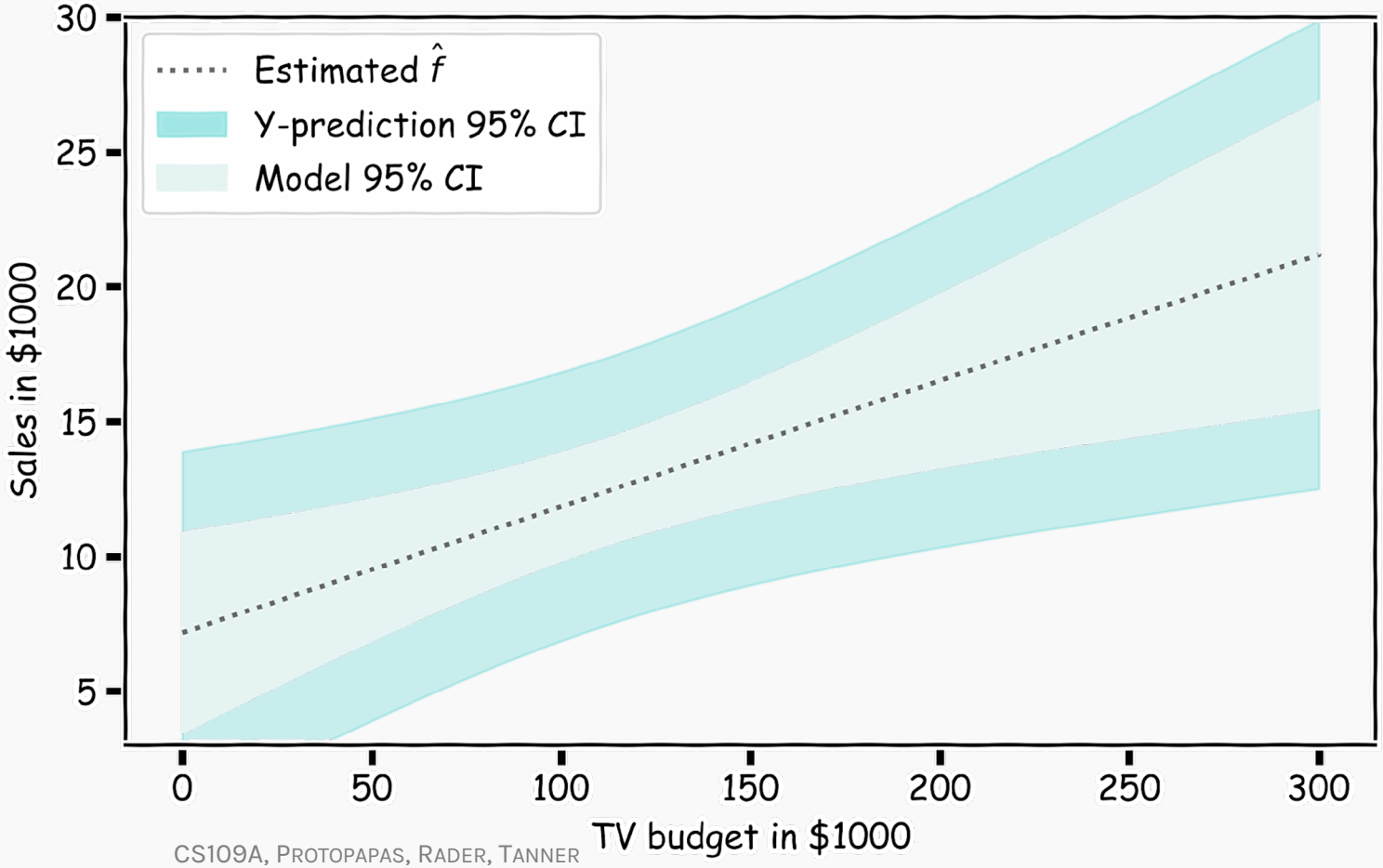
Confidence in predicting $\hat{y}$

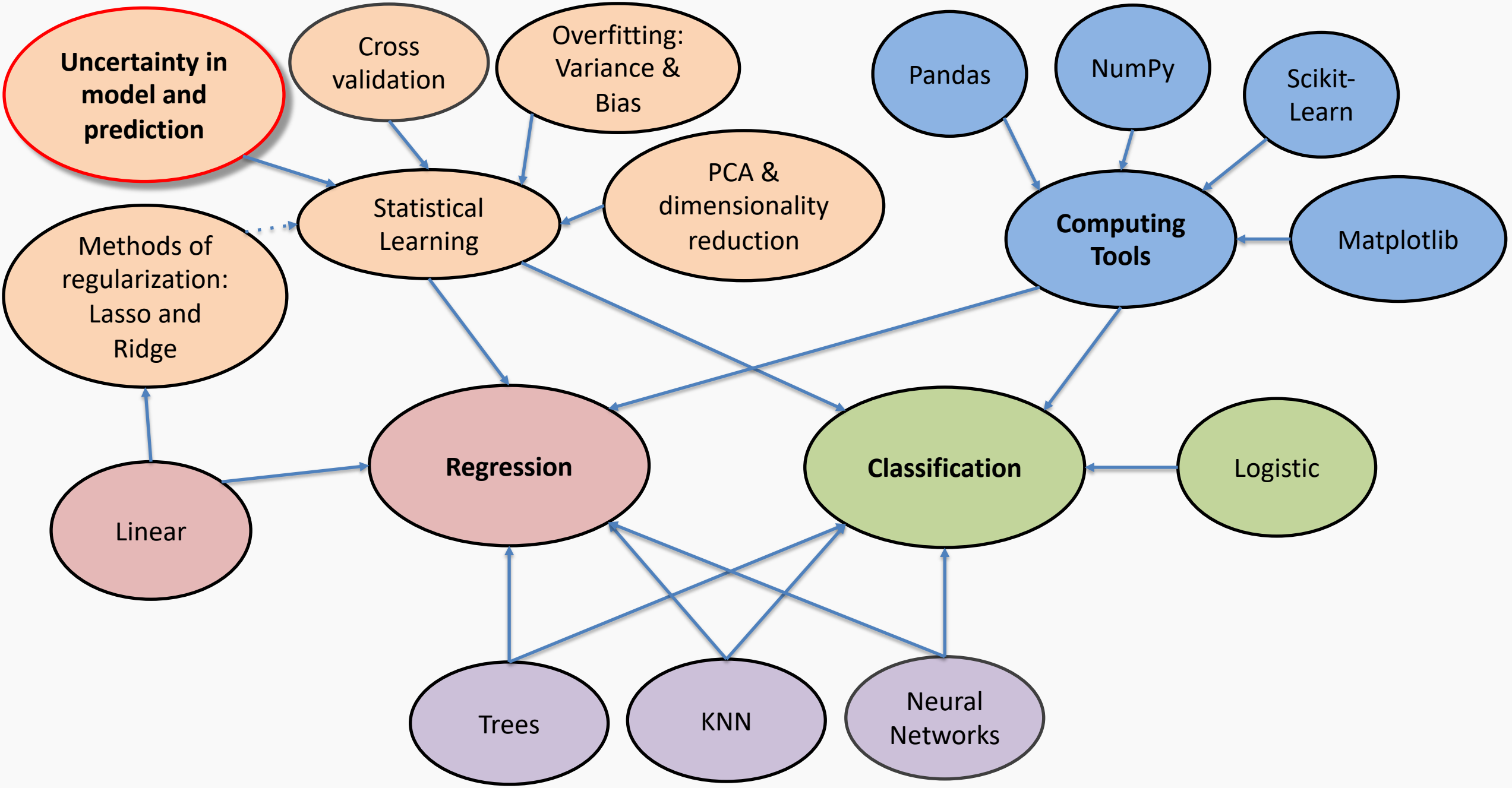
- for a given x, we have a distribution of models
- for each of these the prediction for

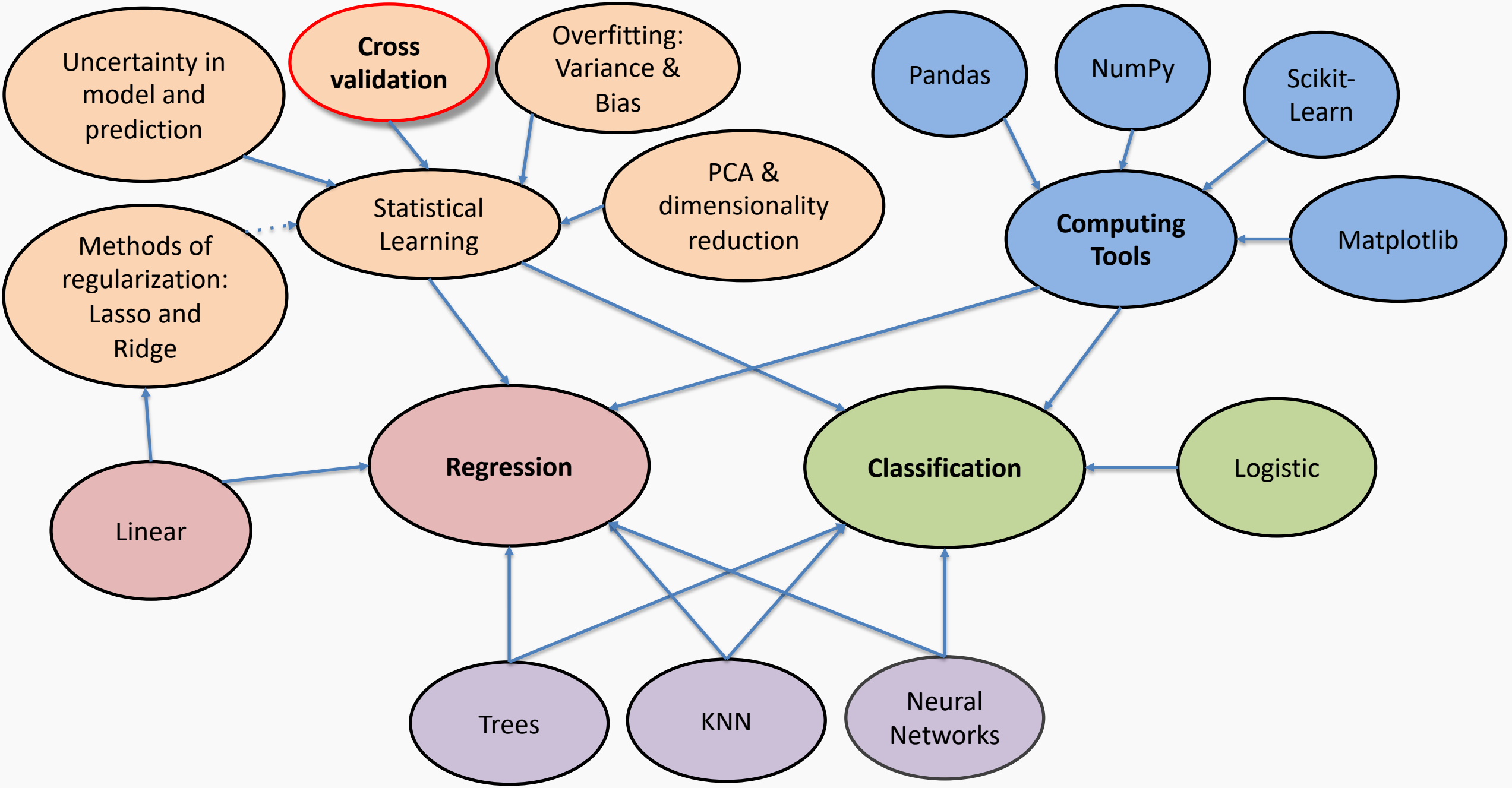


Confidence in predicting $\hat{y}$

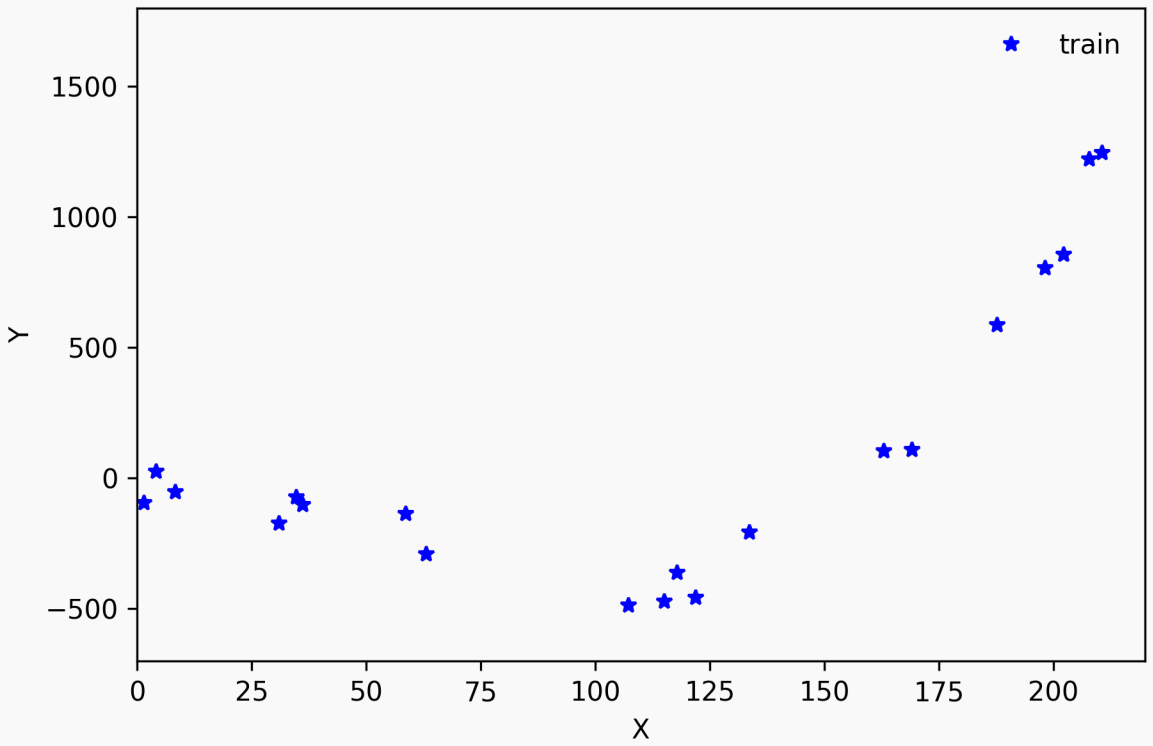
- for a given x, we have a distribution of models $f(x)$
- for each of these $f(x)$, the prediction for $y \sim N(f, \sigma_\epsilon)$



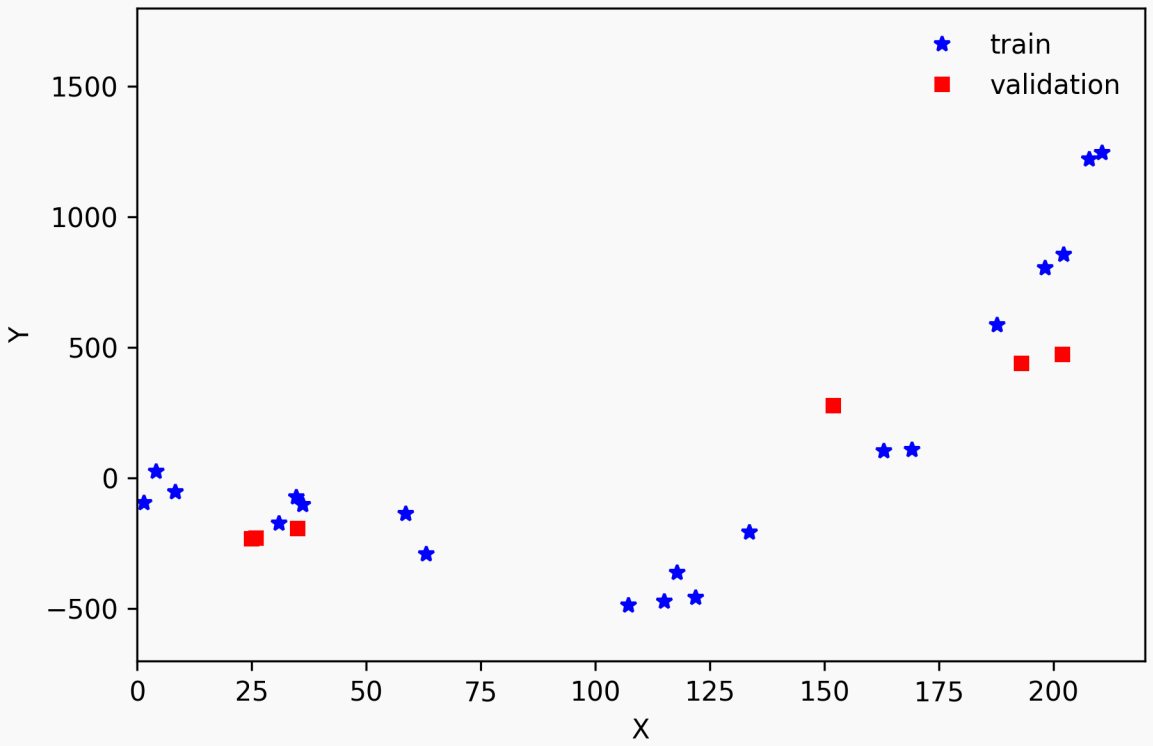

Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks


Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks

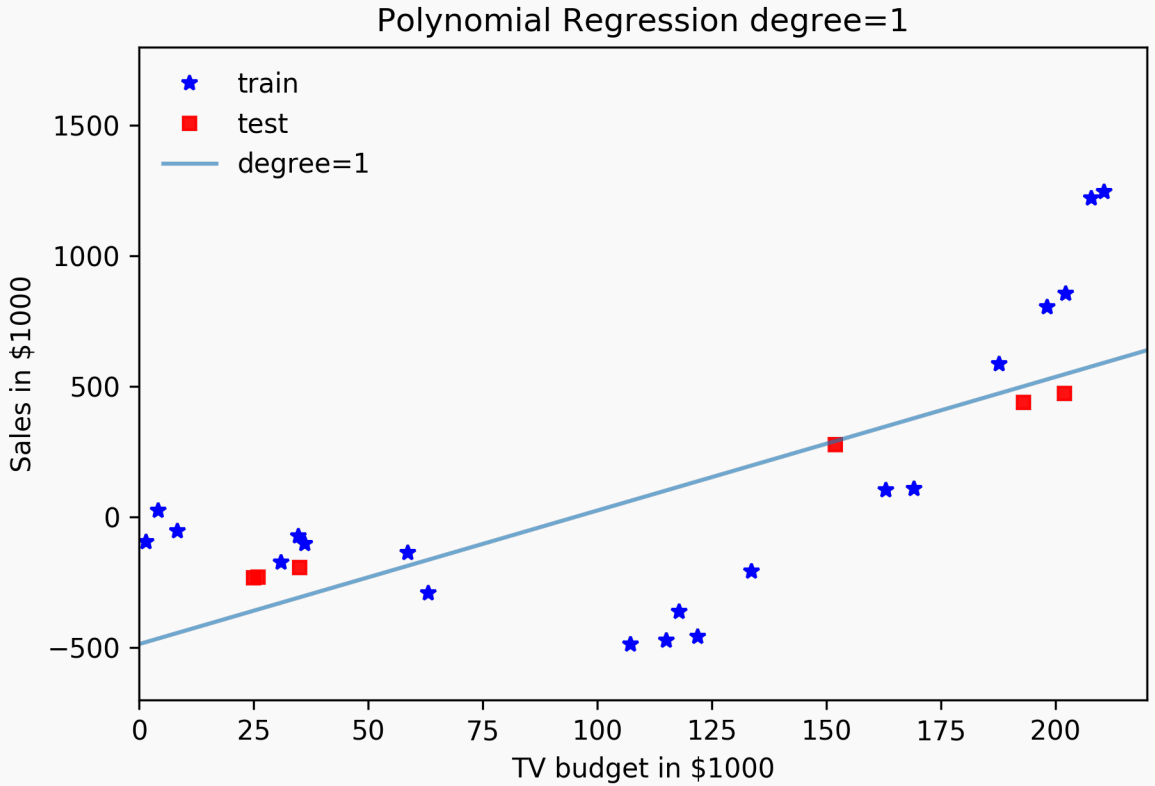
Cross Validation



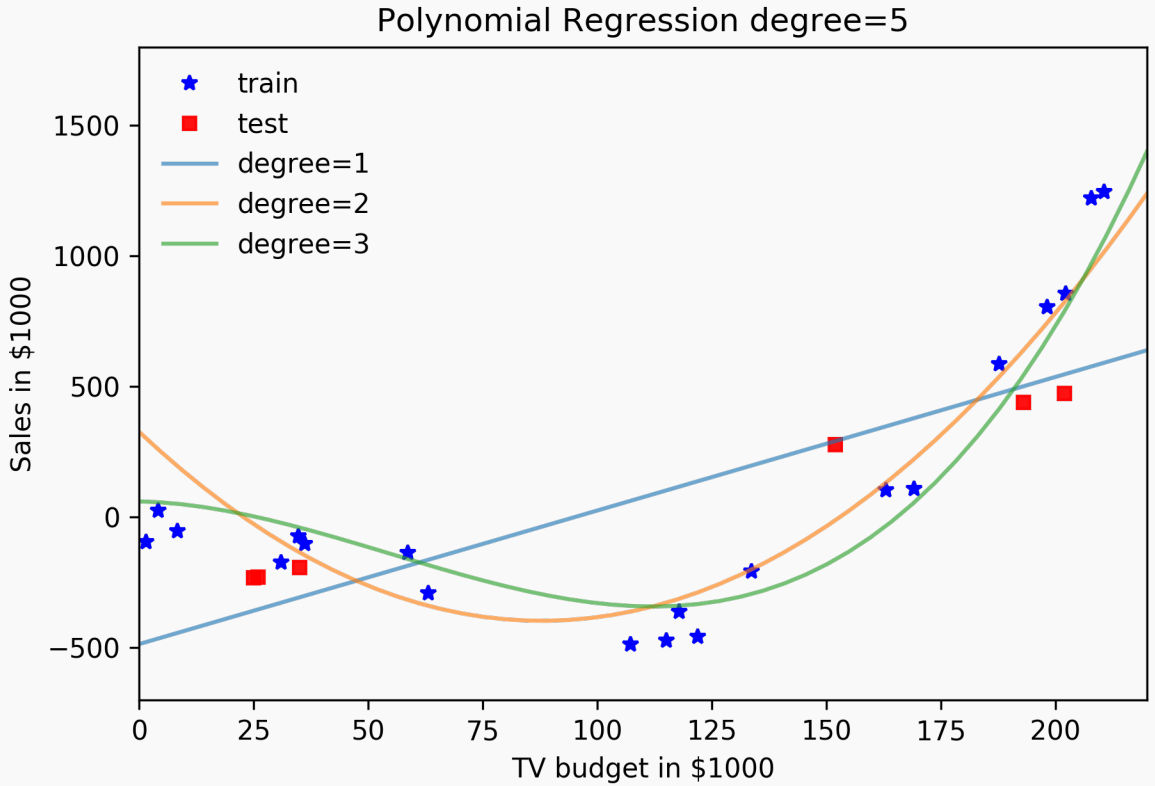
Cross Validation



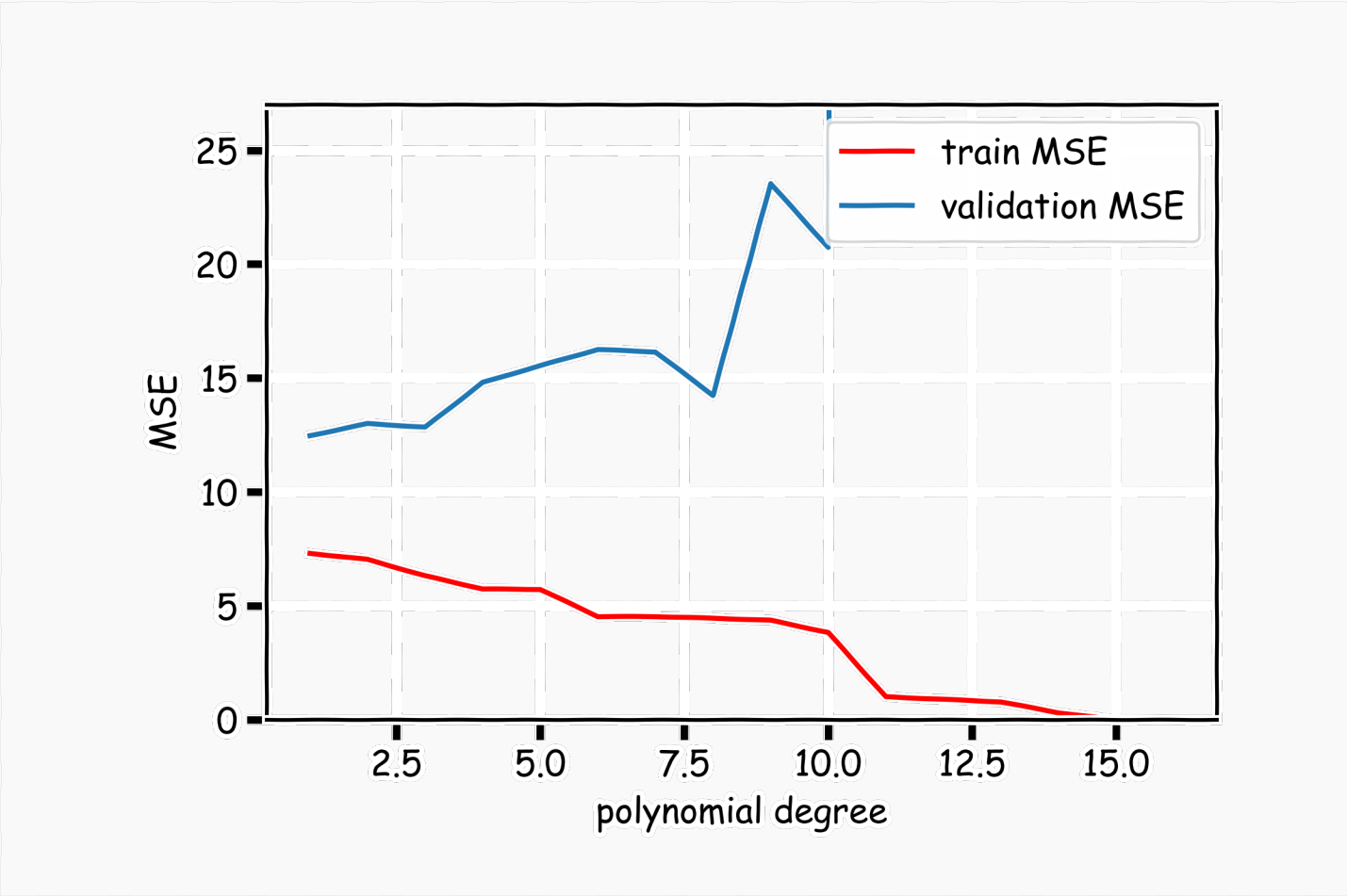
Cross Validation



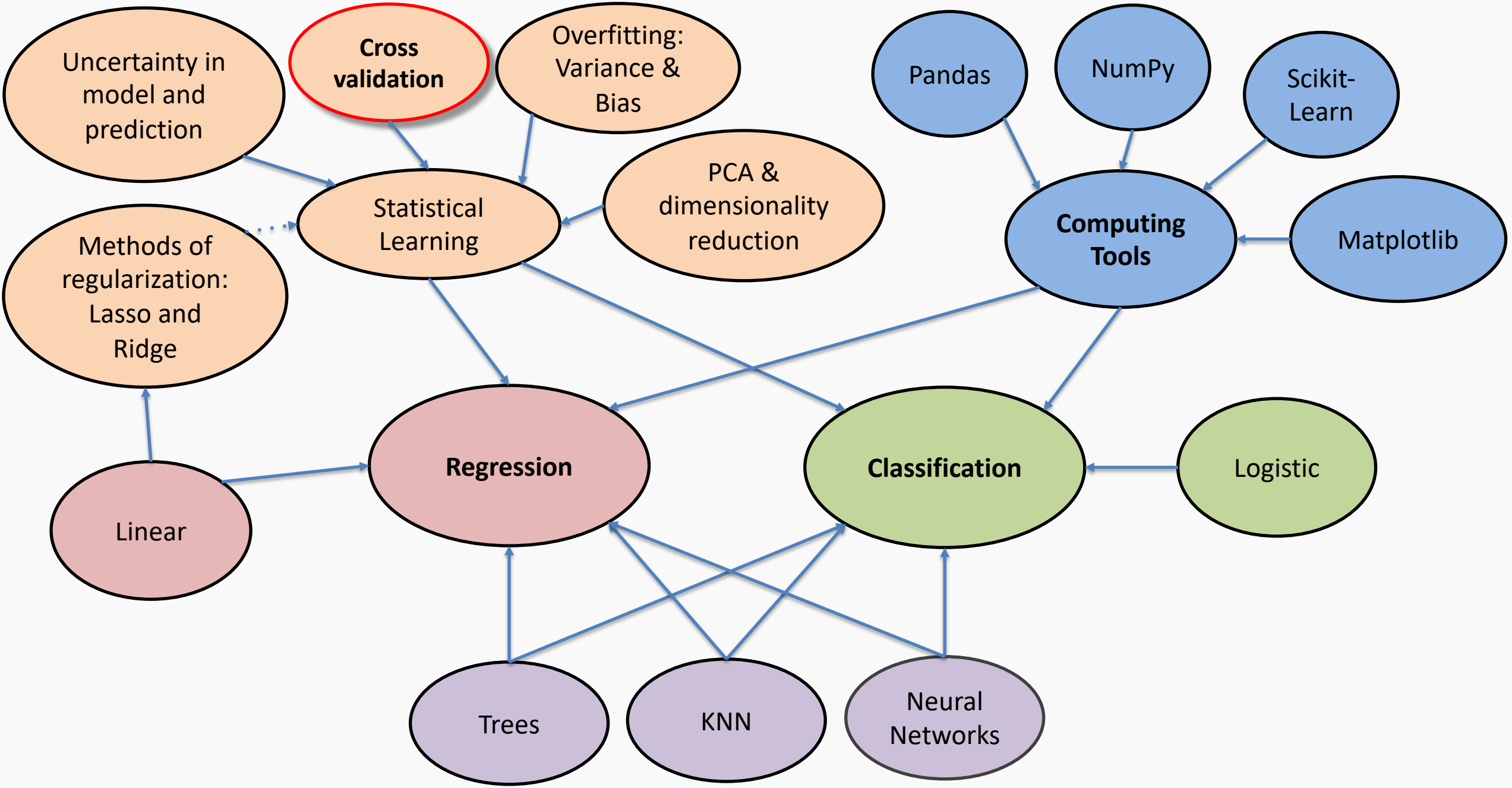
Cross Validation

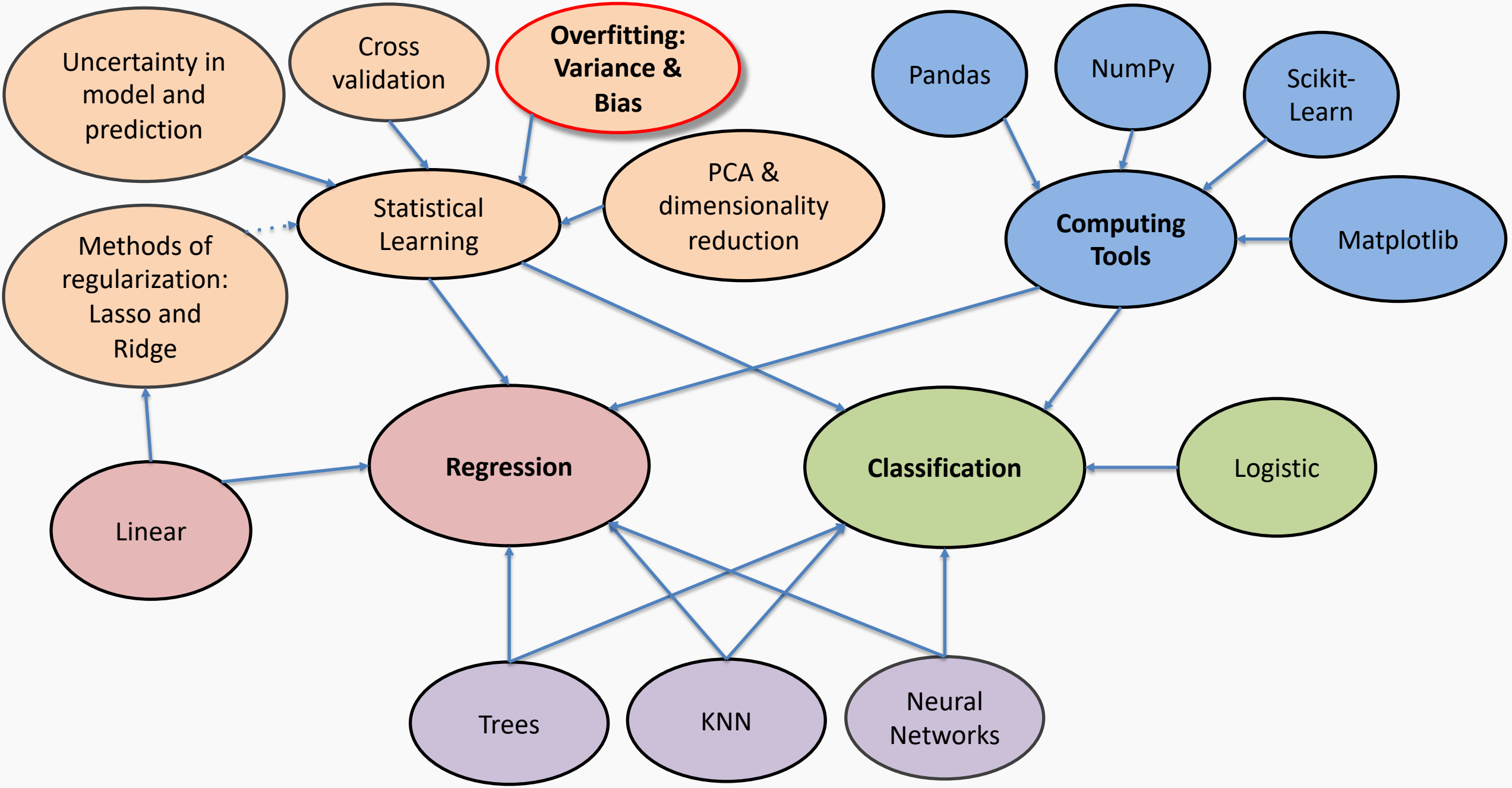


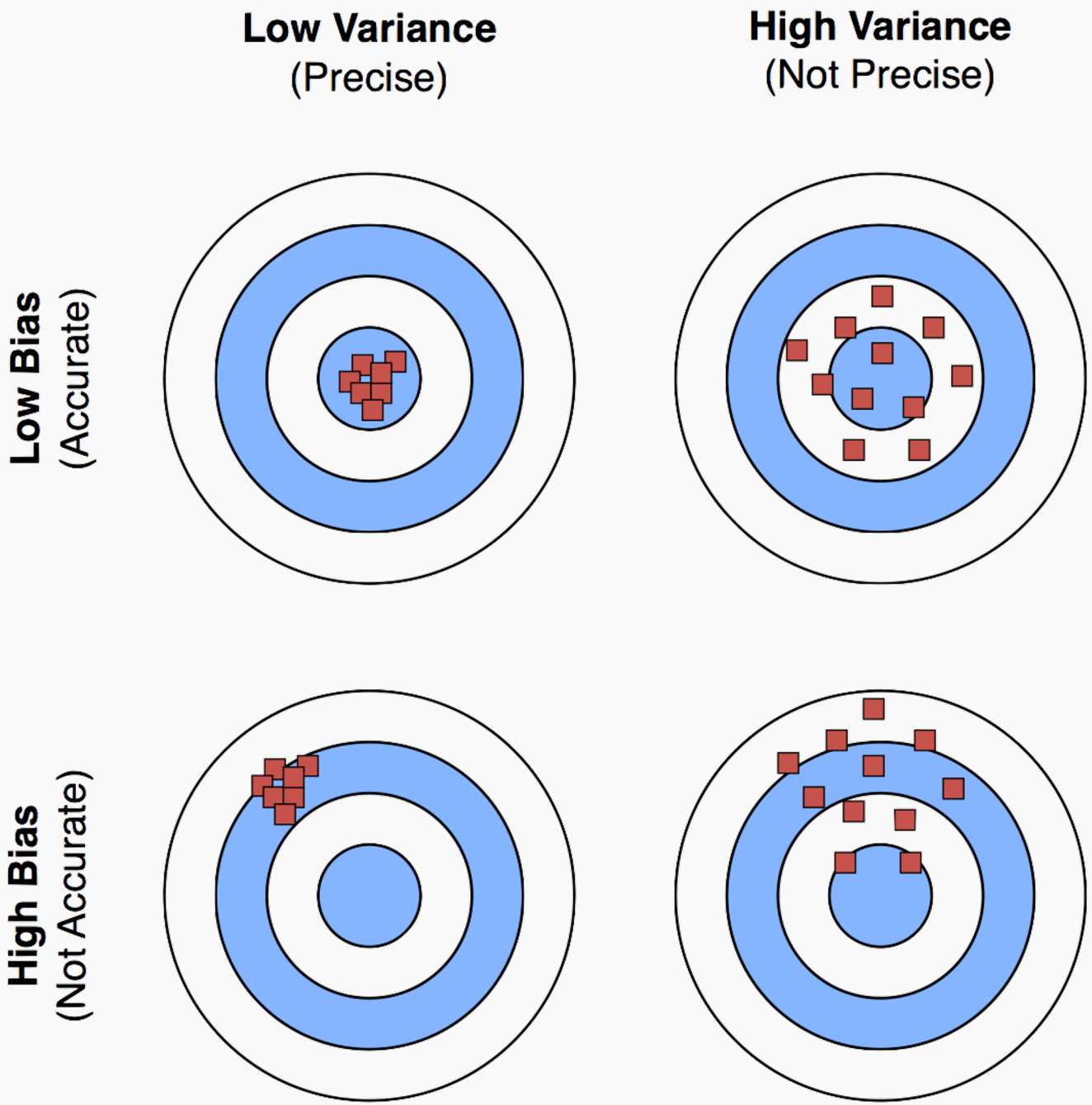
Validation

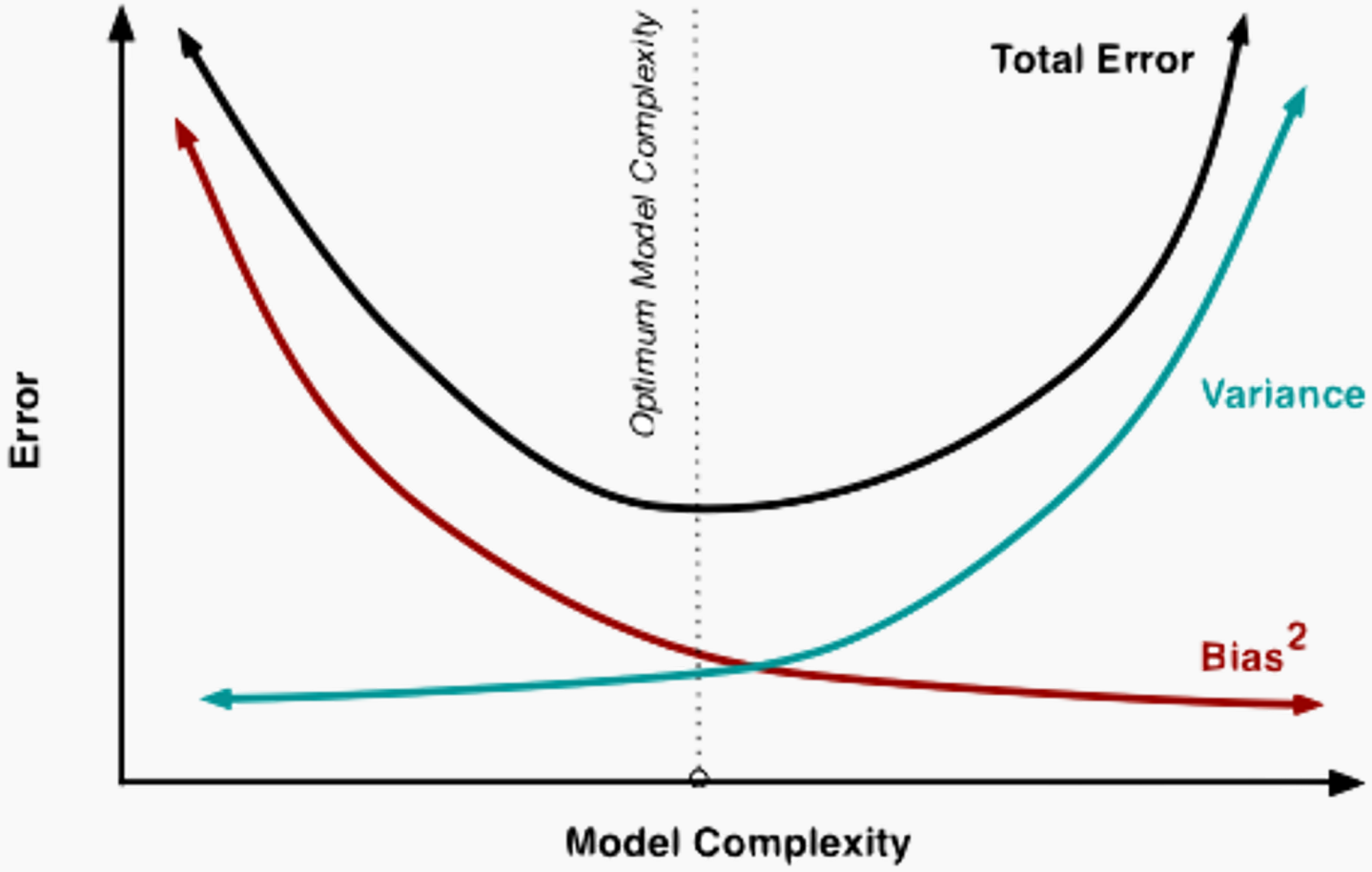

train MSE
validation MSE
MSE
25
20
15
10
5
0
2.5
5.0
7.5
10.0
12.5
15.0
polynomial degree

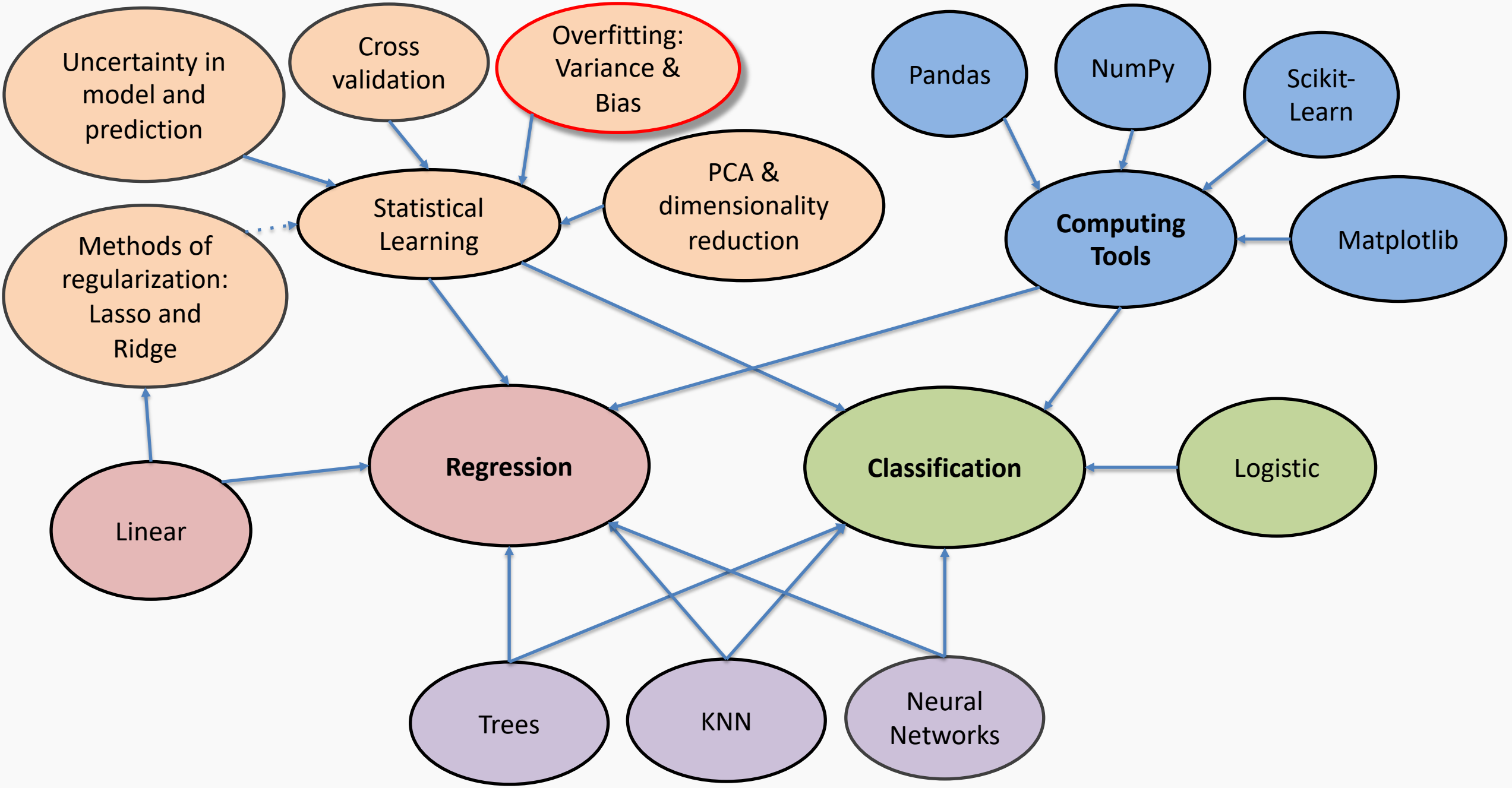
Cross Validation

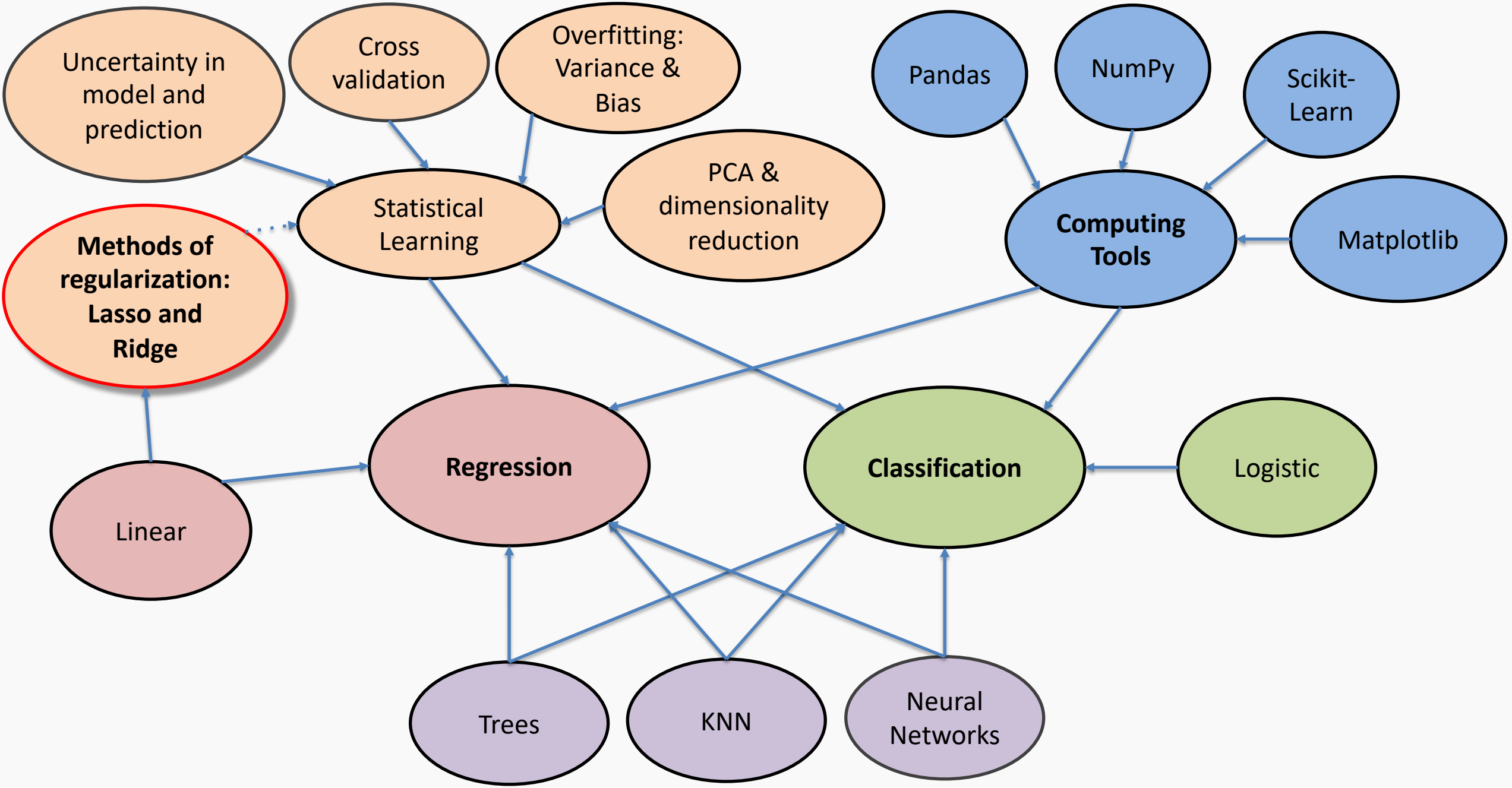

Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
PCA & dimensionality reduction
Statistical Learning
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks


Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks

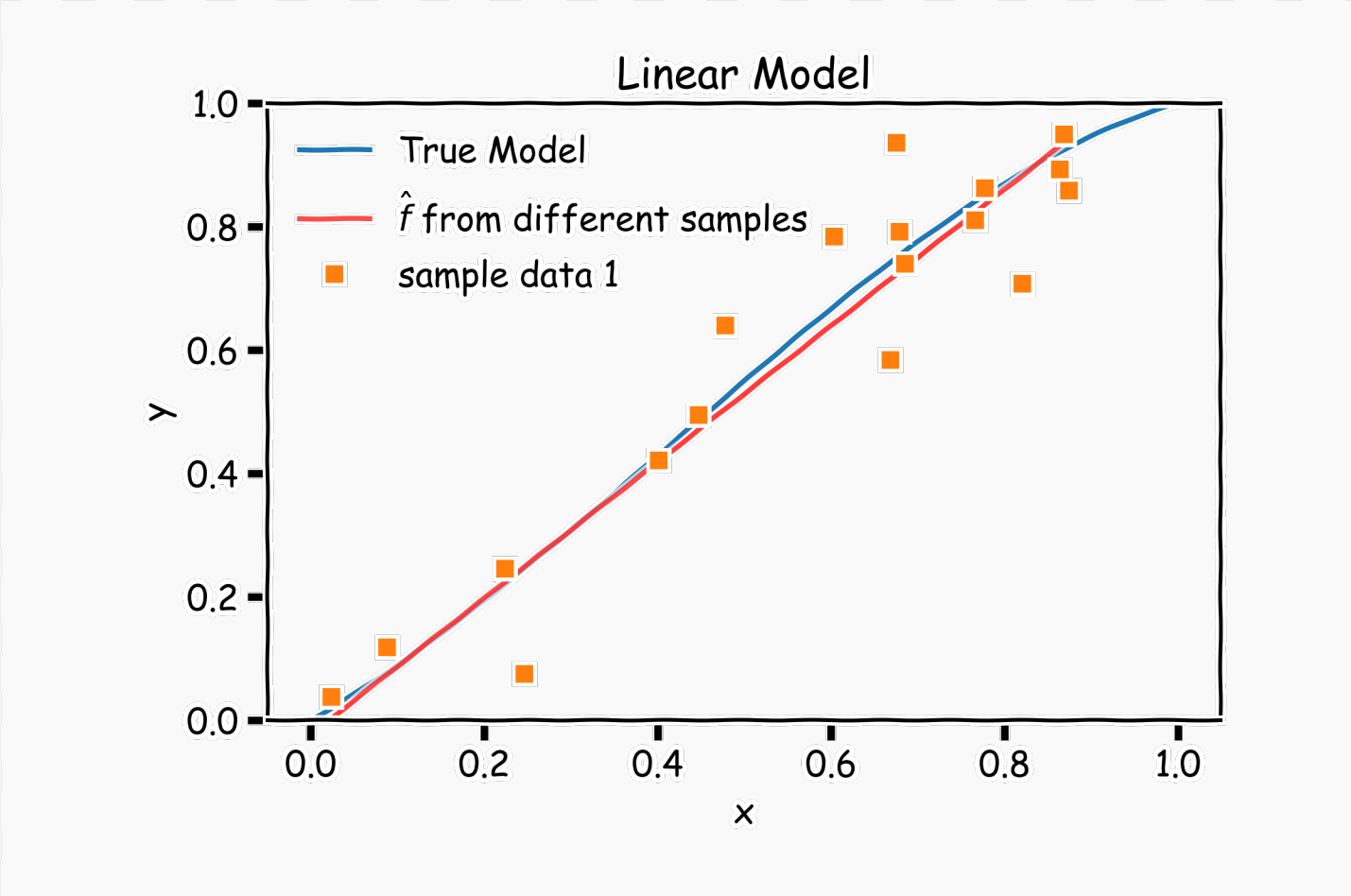

Low Variance
(Precise)
High Variance
(Not Precise)
Low Bias
(Accurate)
High Bias
(Not Accurate)


Total Error
Optimum Model Complexity
Variance
Bias^2
Error
Model Complexity

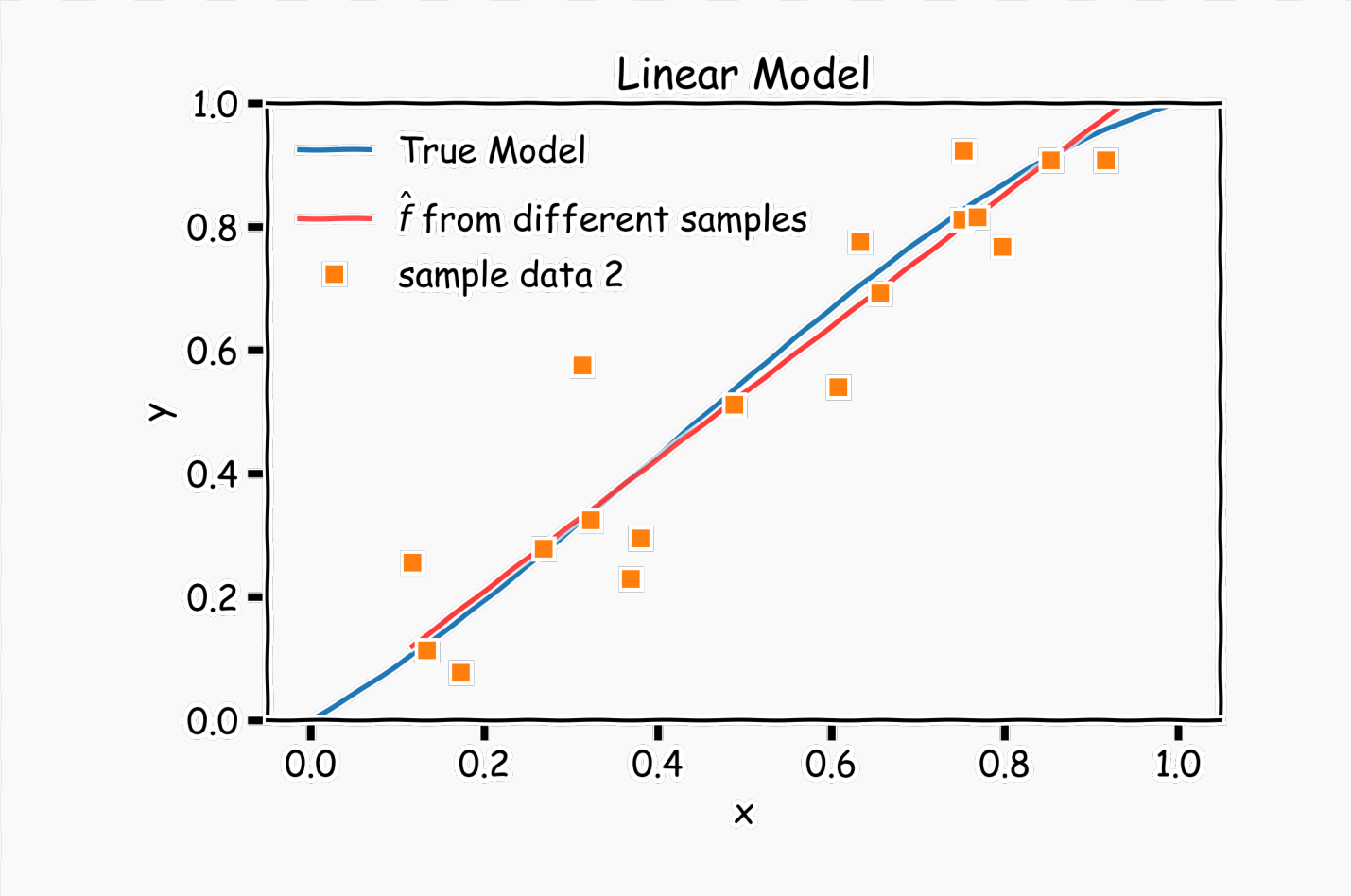

Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks


Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks

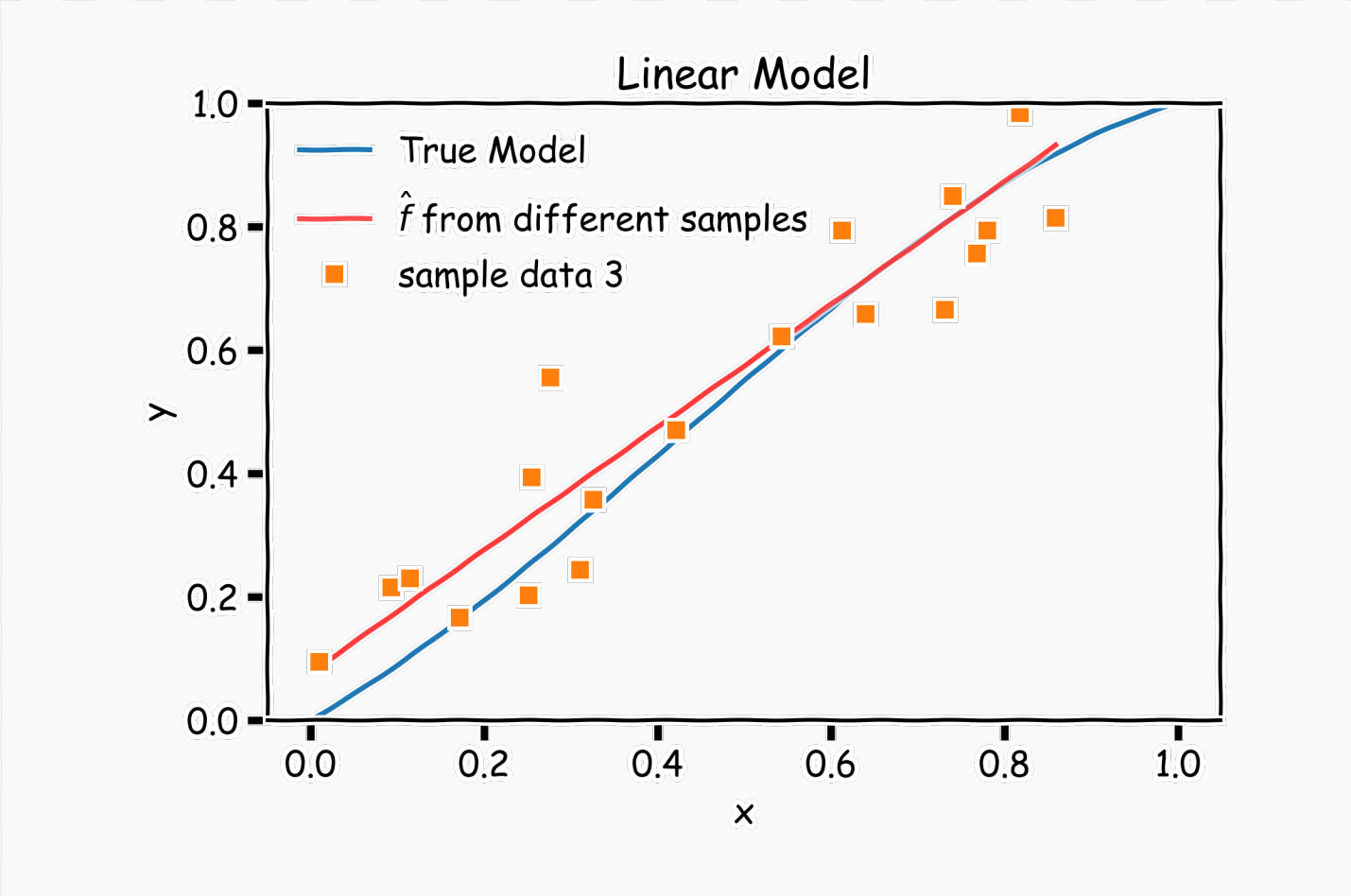
Bias vs Variance



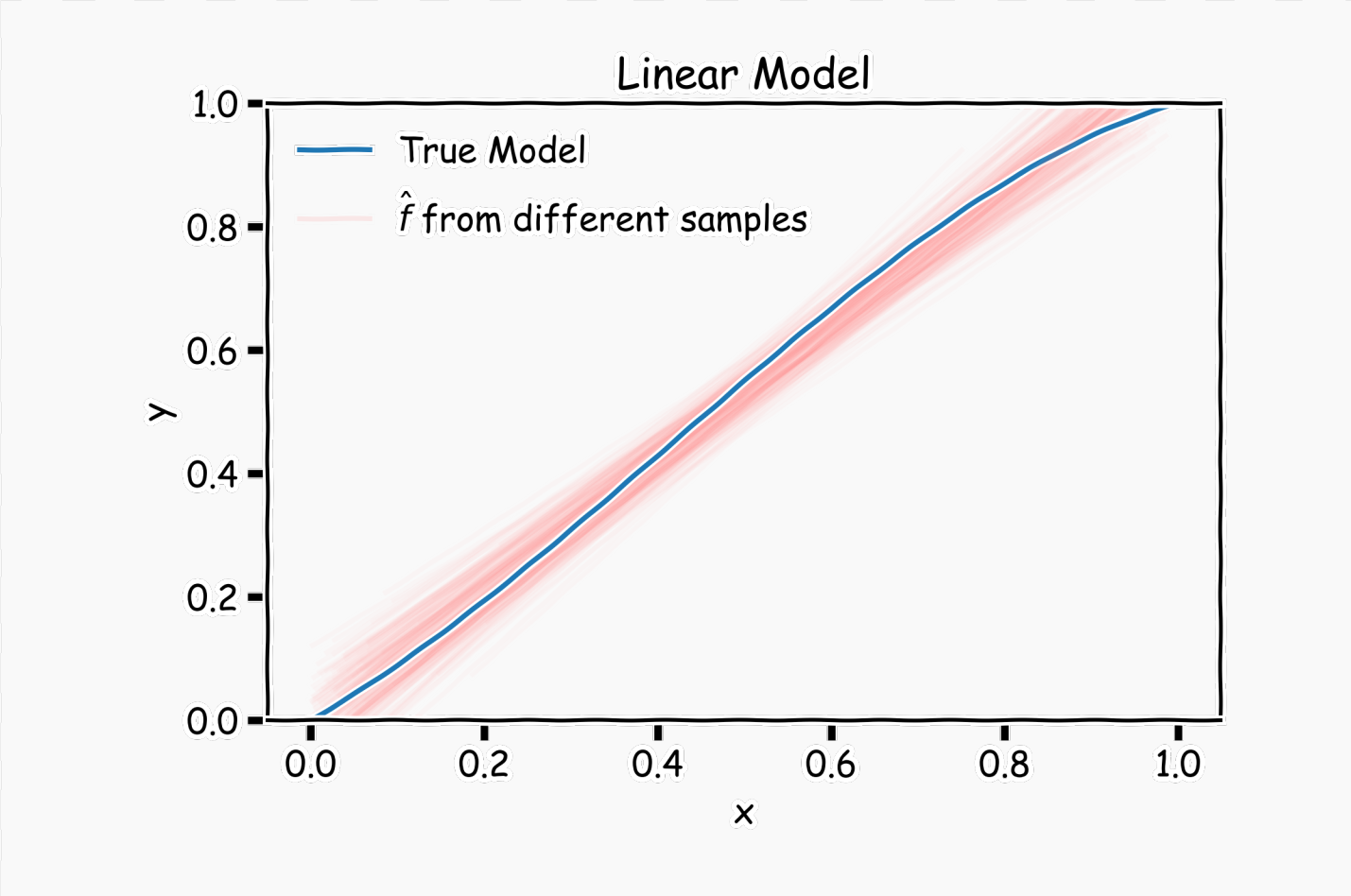
Bias vs Variance



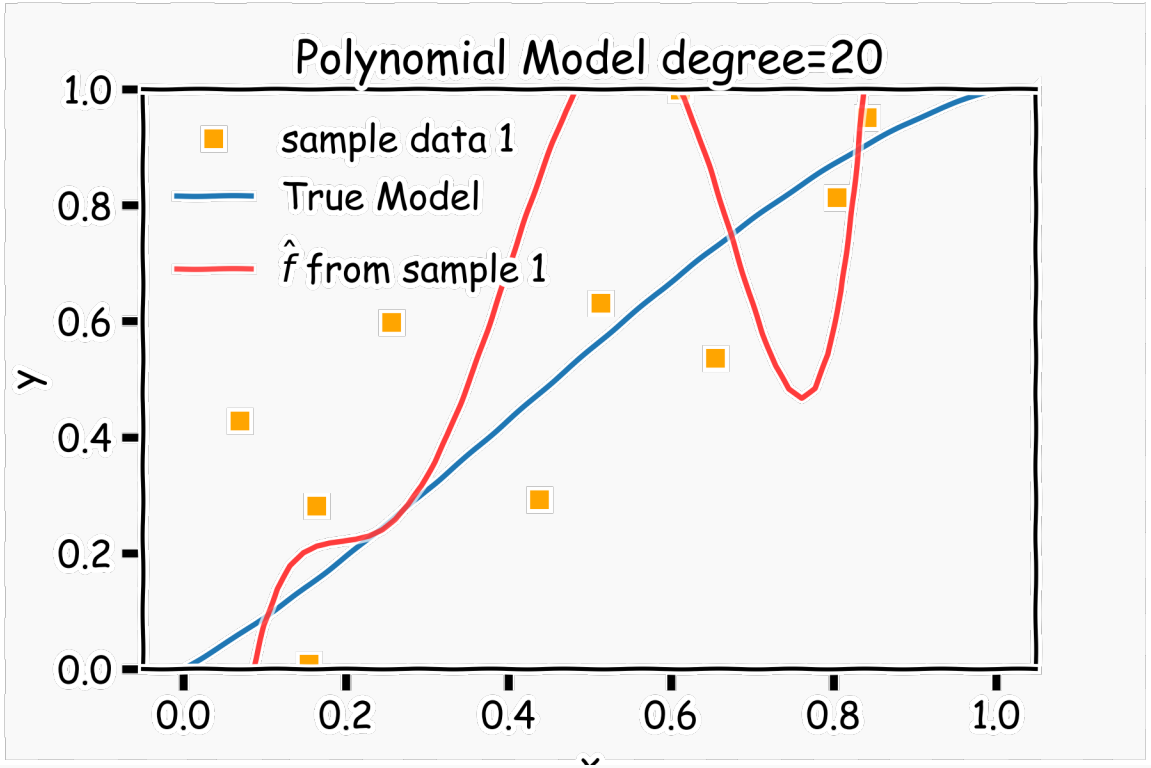
Bias vs Variance



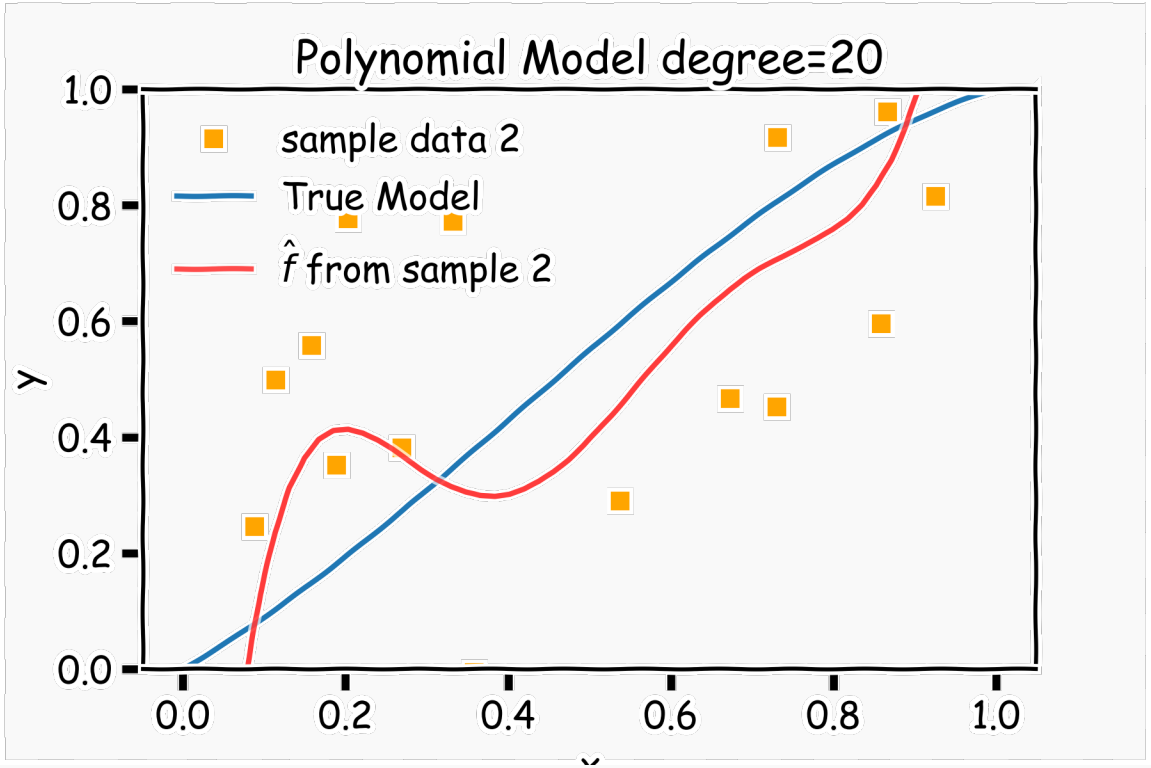
Linear models: 20 data points per line 2000 simulations



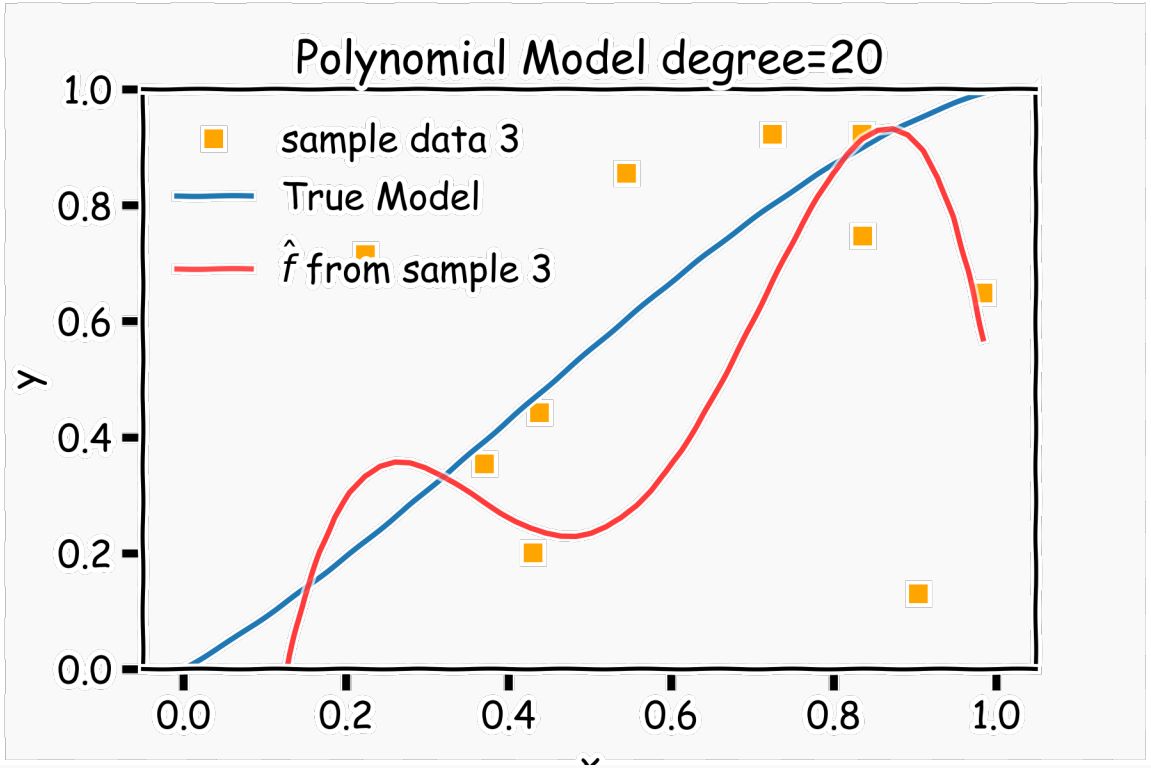
Bias vs Variance



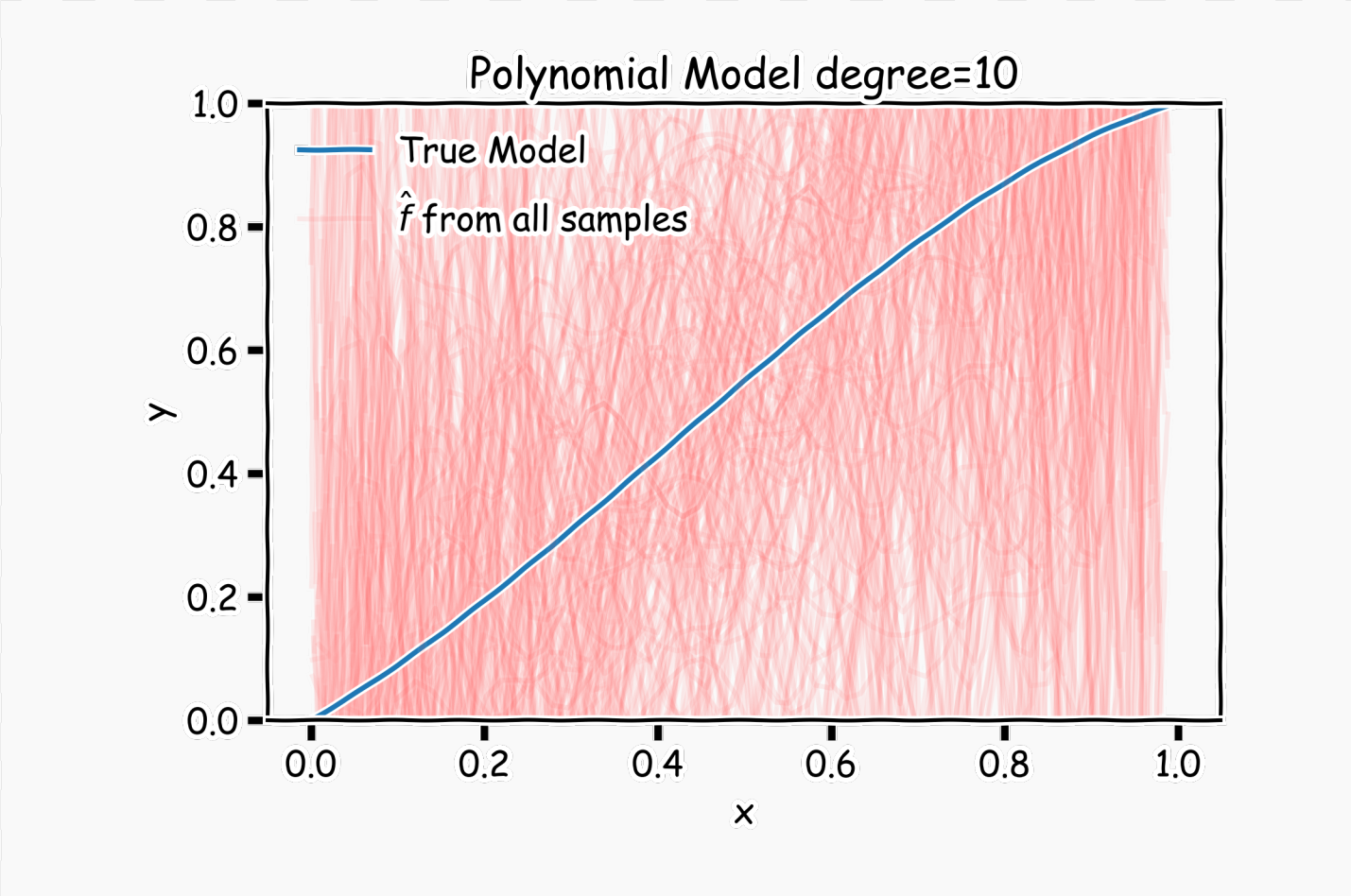
Bias vs Variance



Bias vs Variance



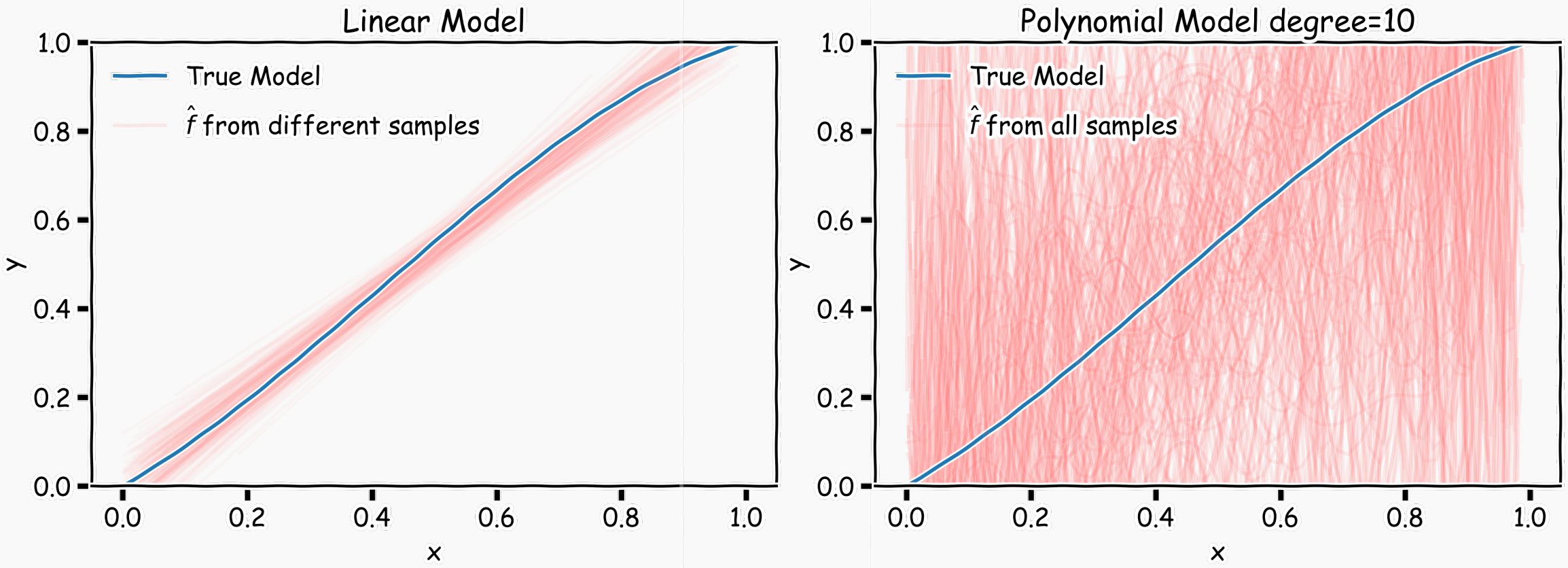
Poly 10 degree models : 20 data points per line 2000 simulations



Bias vs Variance

Left: 2000 best fit straight lines, each fitted on a different 20 point training set.

Right: Best-fit models using degree 10 polynomial



Regularization: An Overview

The idea of regularization revolves around modifying the loss function L; in particular, we add a regularization term that penalizes some specified properties of the model parameters

$$L_{reg}(\beta) = L(\beta) + \lambda R(\beta),$$

LASSO Regression

Since we wish to discourage extreme values in model parameter, we need to choose a regularization term that penalizes parameter magnitudes. For our loss function, we will again use MSE.

Together our regularized loss function is:

$$L_{LASSO}(\beta) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \boldsymbol{\beta}^\top \boldsymbol{x}_i|^2 + \lambda \sum_{j=1}^{J}|\beta_j|.$$

Note that $\sum_{j=1}^{J}|\beta_j|$ is the $\boldsymbol{l_1}$ norm of the vector $\boldsymbol{\beta}$

$$\sum_{j=1}^{J}|\beta_j| = \|\boldsymbol{\beta}\|_1$$

Ridge Regression

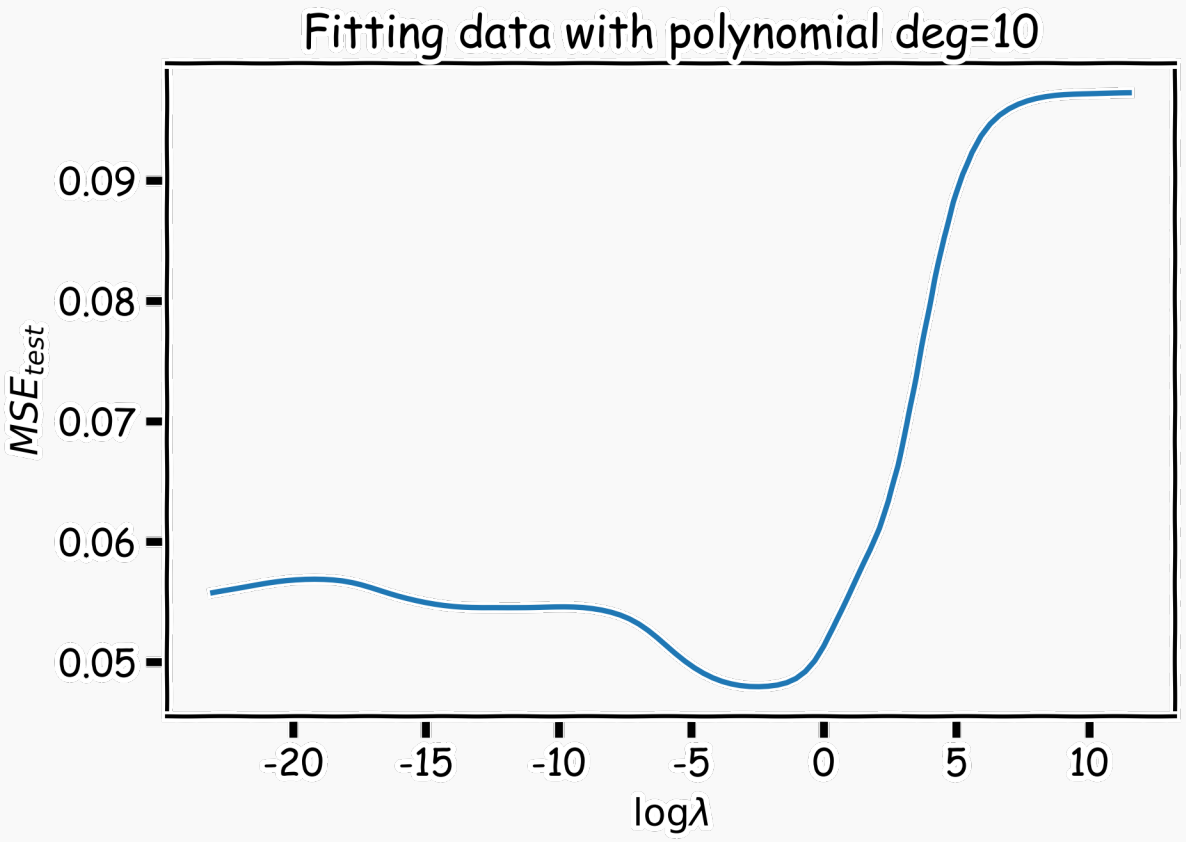
Alternatively, we can choose a regularization term that penalizes the squares of the parameter magnitudes. Then, our regularized loss function is:

$$L_{Ridge}(\beta) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \boldsymbol{\beta}^\top \boldsymbol{x}_i|^2 + \lambda \sum_{j=1}^{J}\beta_j^2.$$

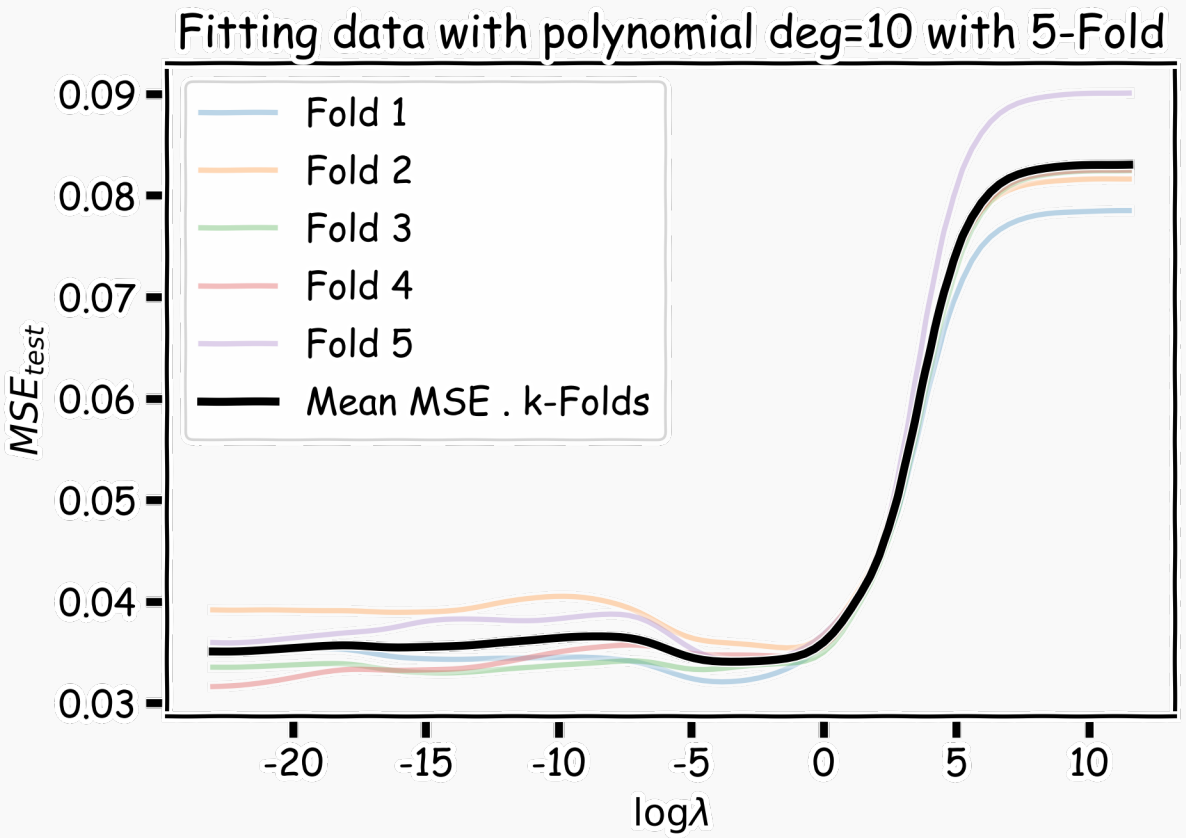
Note that $\sum_{j=1}^{J}|\beta_j|^2$ is the $\boldsymbol{l_2}$ norm of the vector $\boldsymbol{\beta}$

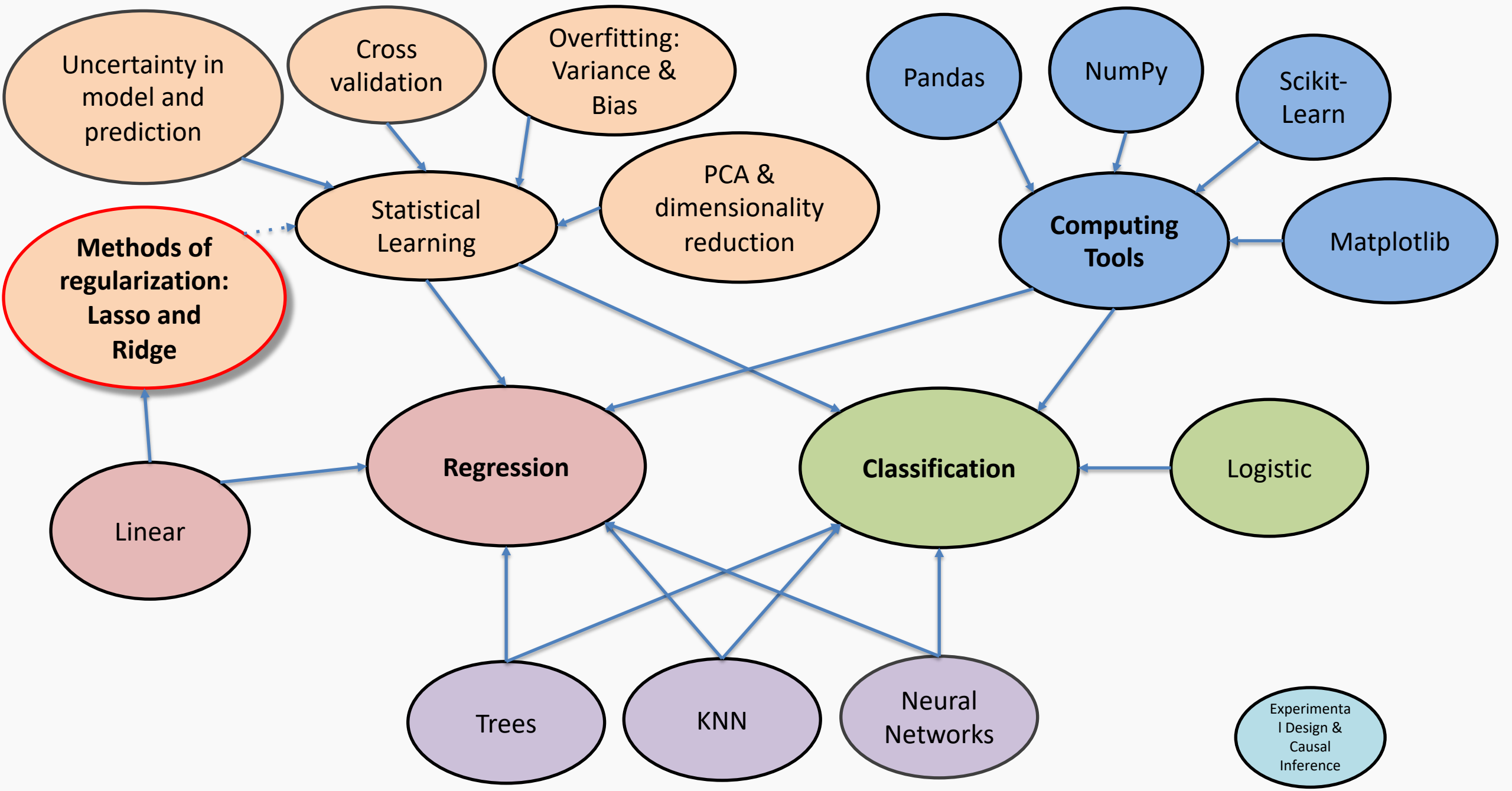
$$\sum_{j=1}^{J}\beta_j^2 = \|\boldsymbol{\beta}\|_2^2$$

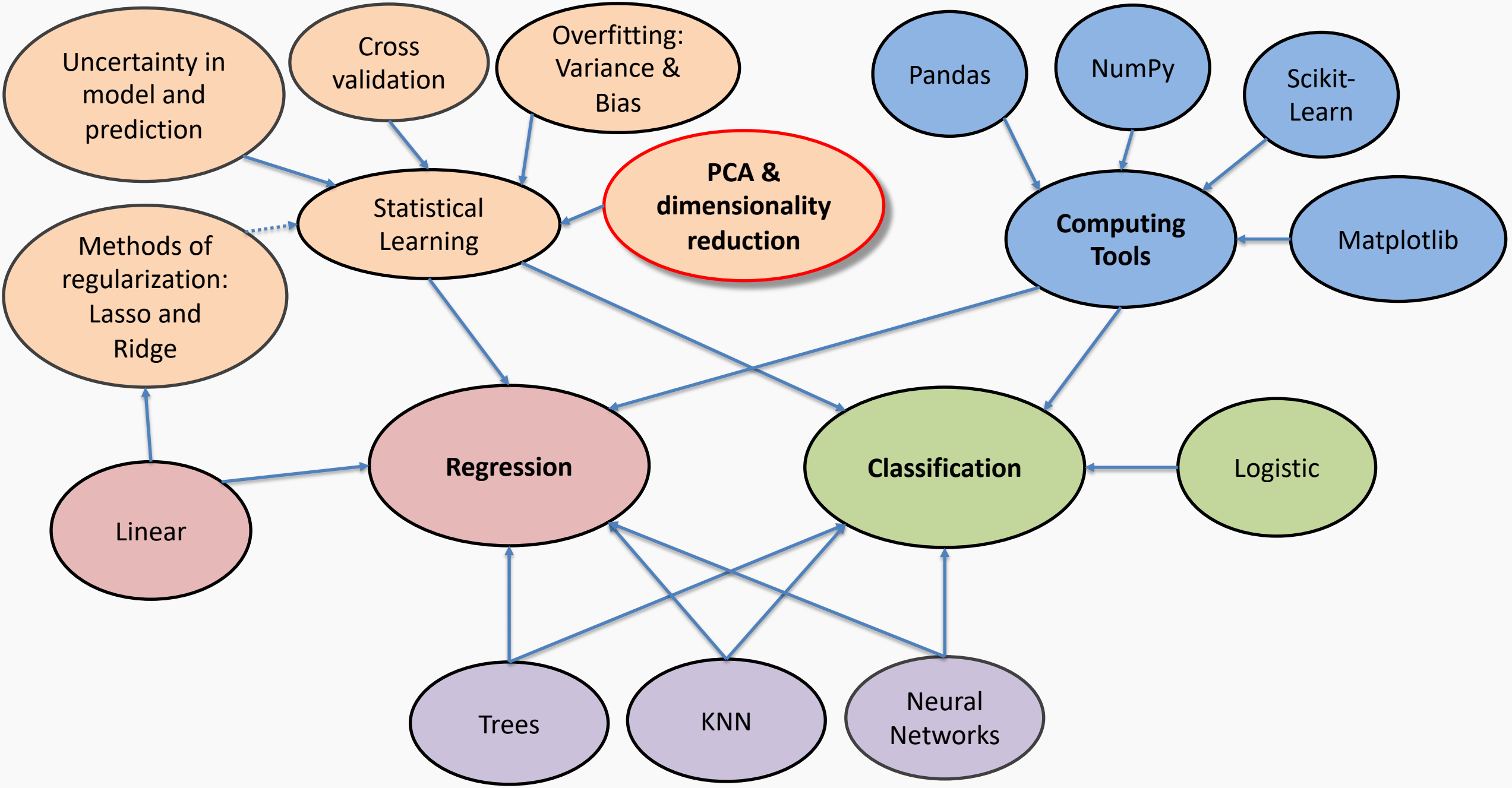
Ridge regularization with **validation** only: step by step



Ridge regularization with **validation** only: step by step



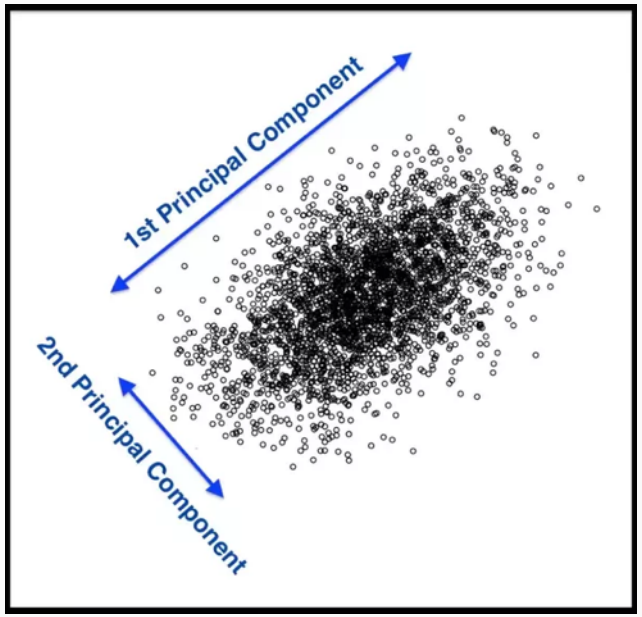

Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks
Experimental Design & Causal Inference


Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks

The Intuition Behind PCA

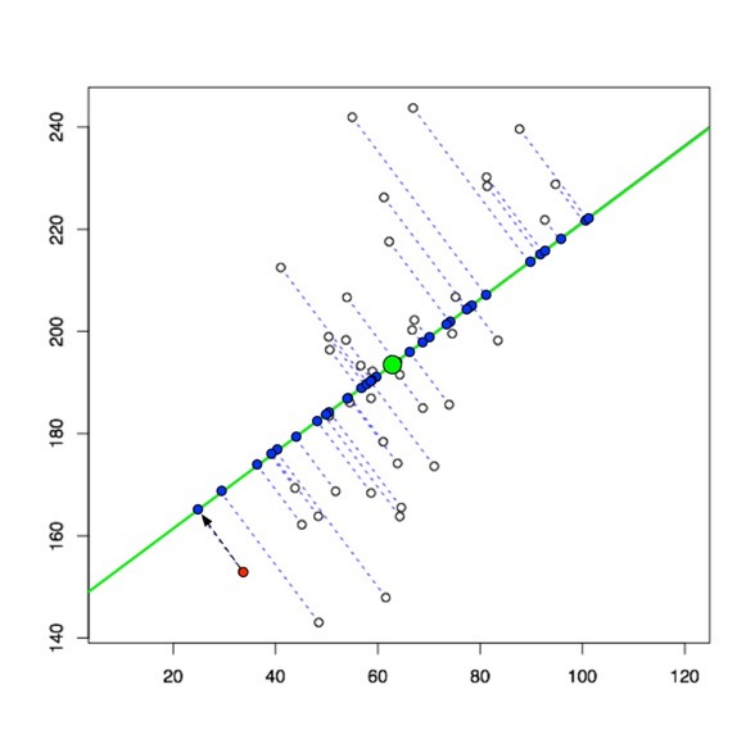
Top PCA components capture the most of amount of variation (interesting features) of the data.

Each component is a linear combination of the original predictors - we visualize them as vectors in the feature space.



The Intuition Behind PCA (cont.)

Transforming our observed data means projecting our dataset onto the space defined by the top *m* PCA components, these components are our new predictors.



A Framework For Dimensionality Reduction

One way to reduce the dimensions of the feature space is to create a new, smaller set of predictors by taking linear combinations of the original predictors.

We choose $Z_1, Z_2, \ldots, Z_m$, where $m \leq p$ and where each Z_i is a linear combination of the original p predictors

$$Z_i = \sum_{j=1}^{p} \phi_{ji} X_j$$

for fixed constants ϕ_{ji}. Then we can build a linear regression regression model using the new predictors

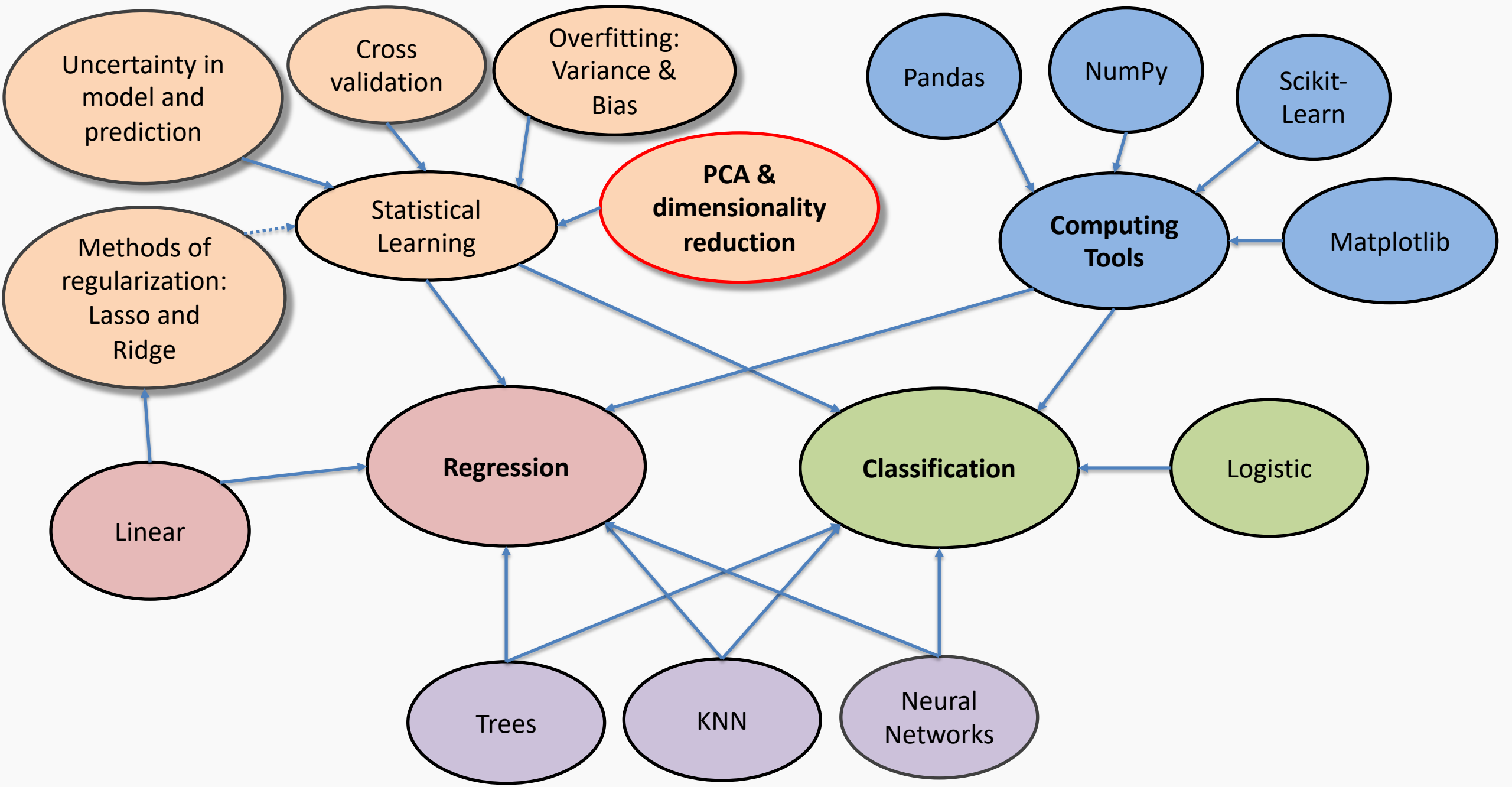
$$Y = \beta_0 + \beta_1 Z_1 + \cdots + \beta_m Z_m + \varepsilon$$

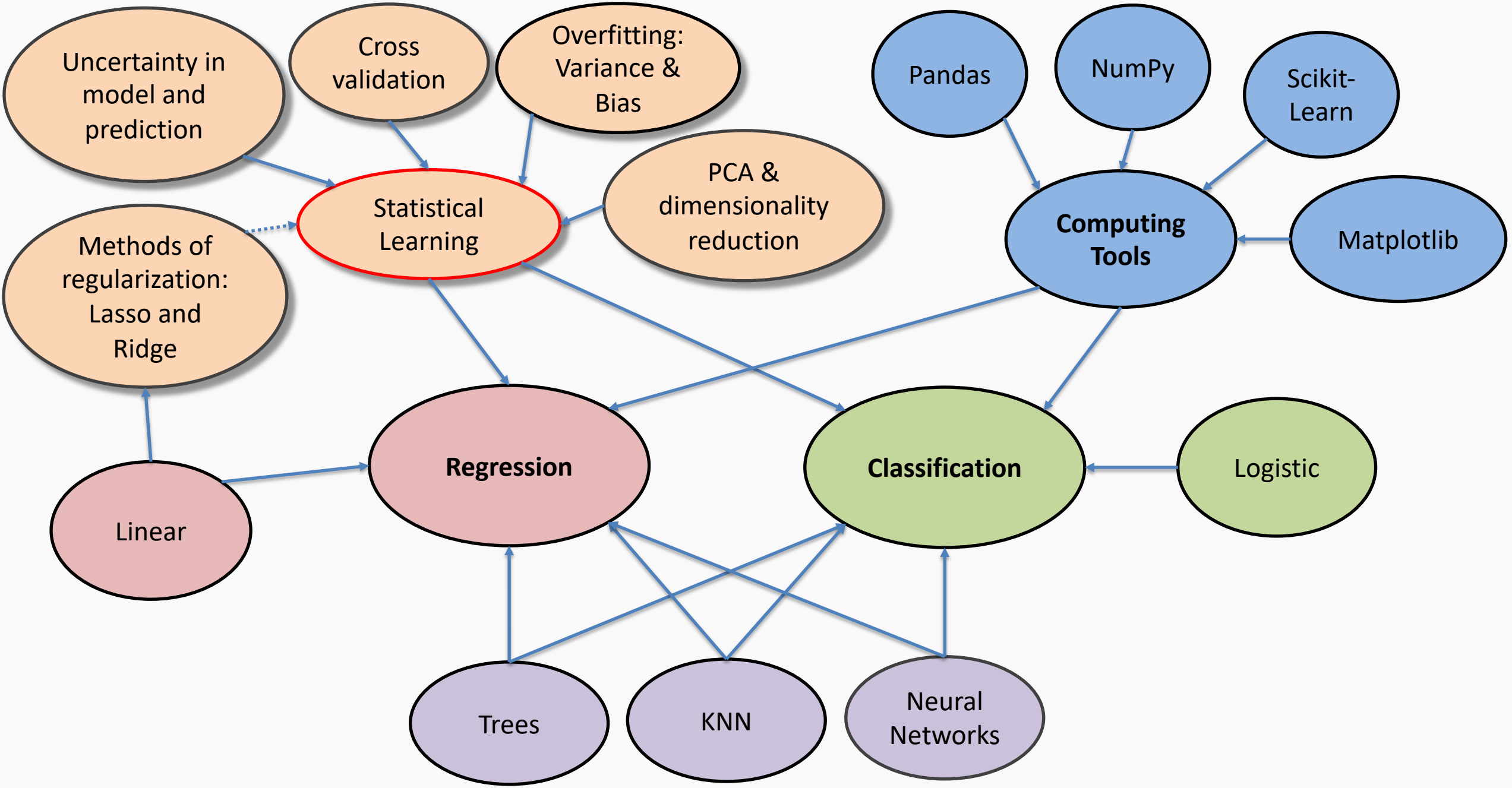
Notice that this model has a smaller number ($m+1 < p+1$) of parameters.

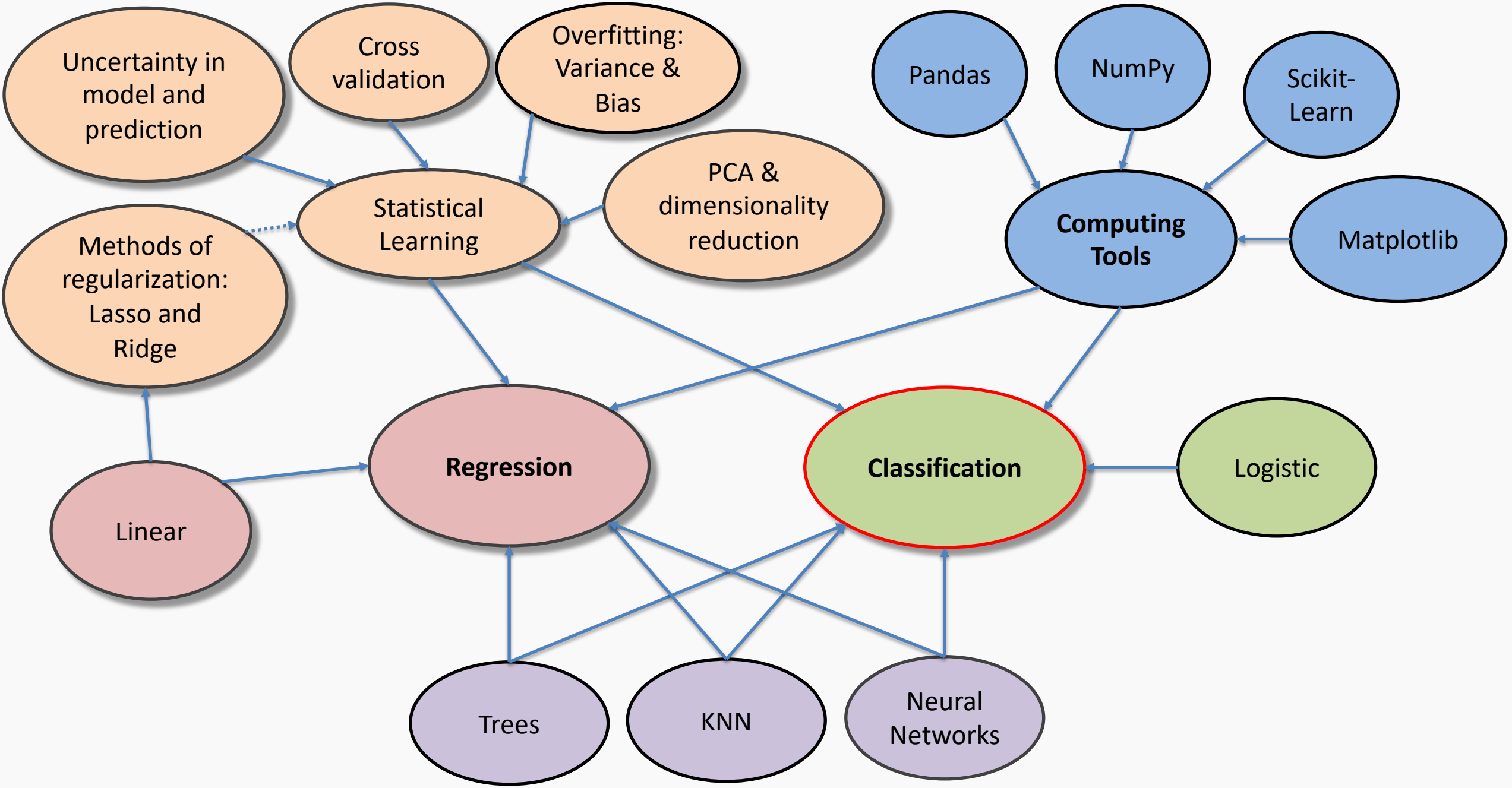
The Math behind PCA

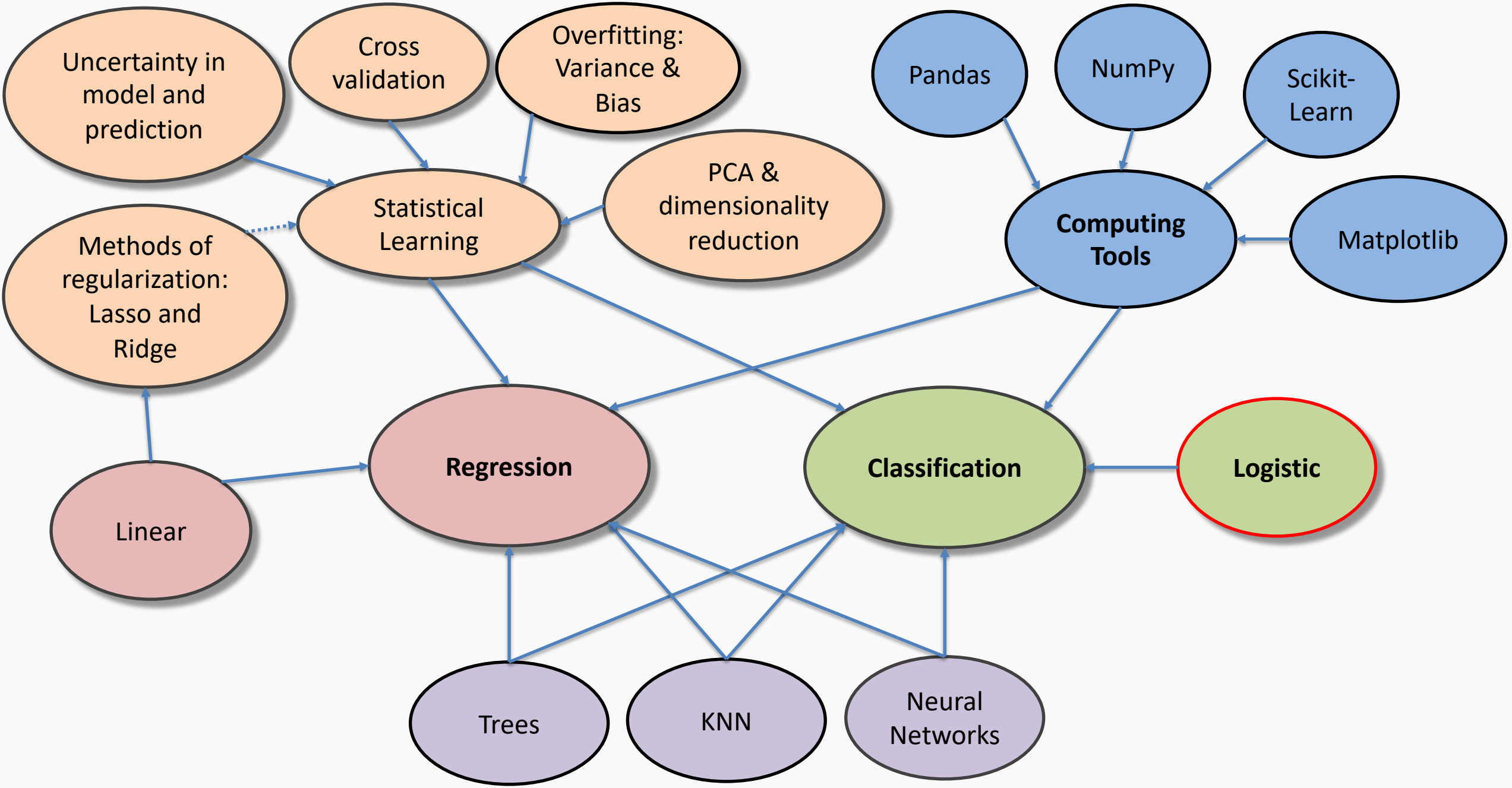
PCA is a well-known result from linear algebra. Let **Z** be the n x p matrix consisting of columns $Z_1, \ldots, Z_p$ (the resulting PCA vectors), **X** be the n x p matrix of $X_1, \ldots, X_p$ of the original data variables (each standardized to have mean zero and variance one, and without the intercept), and let **W** be the p x p matrix whose columns are the eigenvectors of the square matrix $\mathbf{X}^T\mathbf{X}$, then:

$$\mathbf{Z}_{n\times p} = \mathbf{X}_{n\times p}\mathbf{W}_{p\times p}$$


Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks


Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks

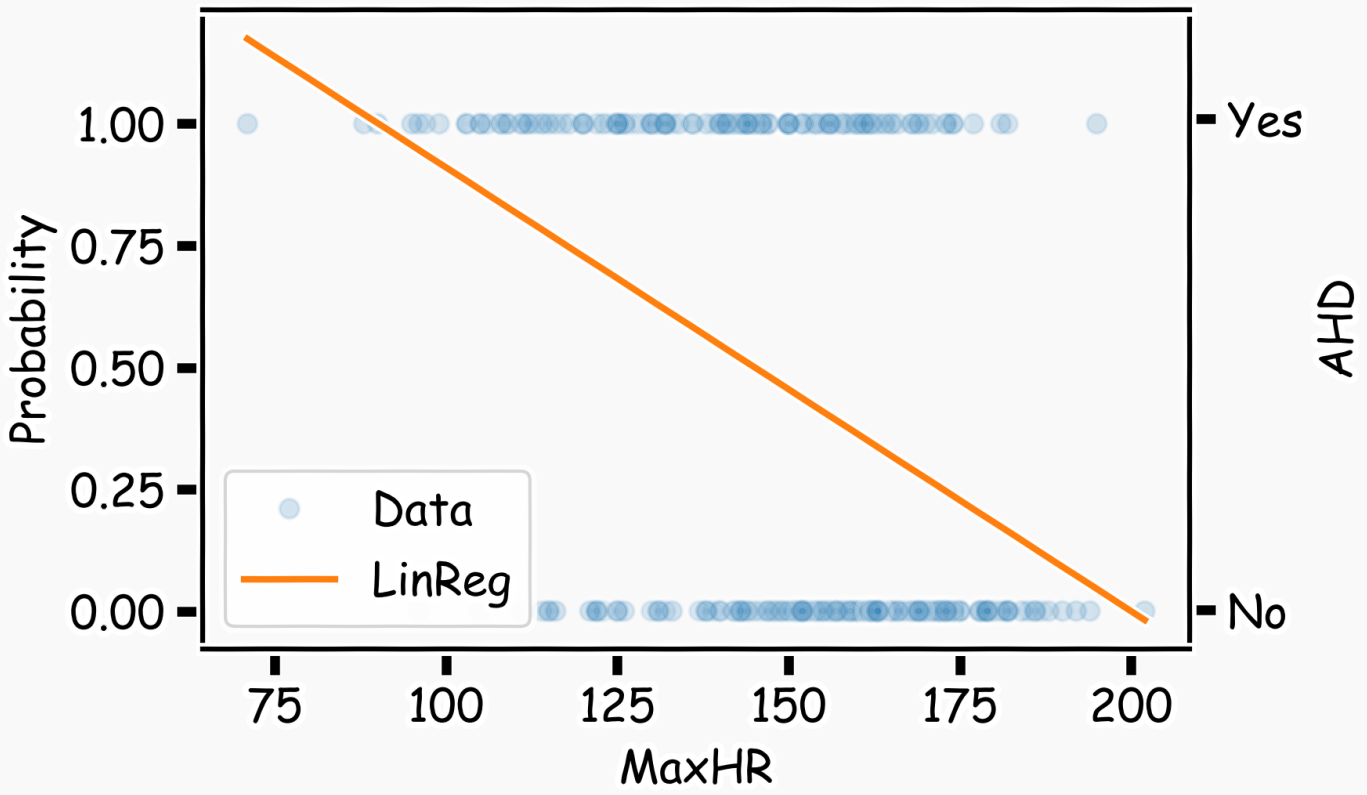

Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks


Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks

Even Simpler Classification Problem: Binary Response (cont)

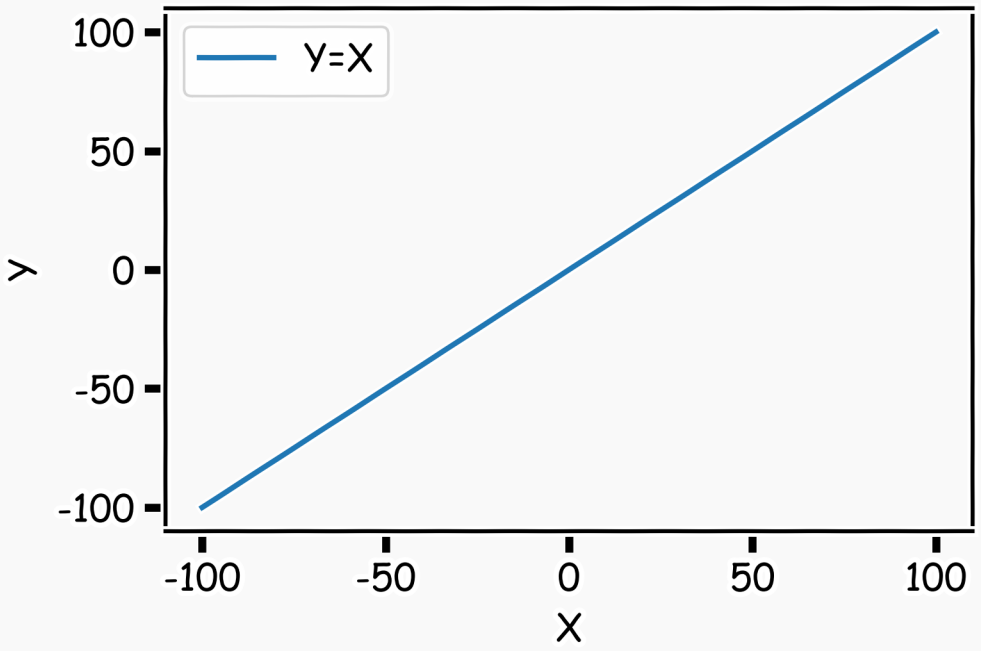
What could go wrong with this linear regression model?

.

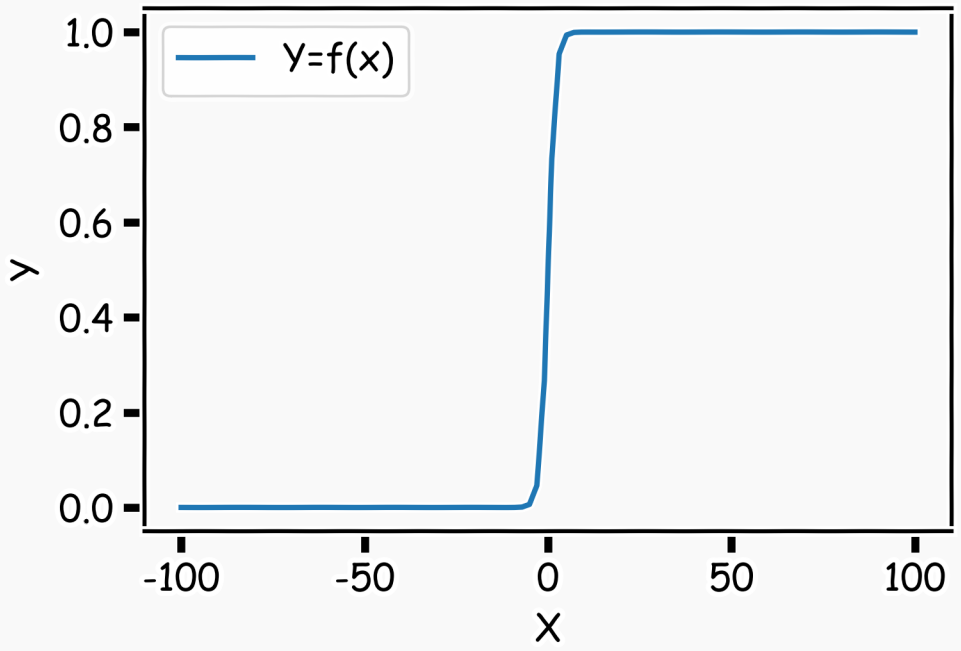


Pavlos Game #45

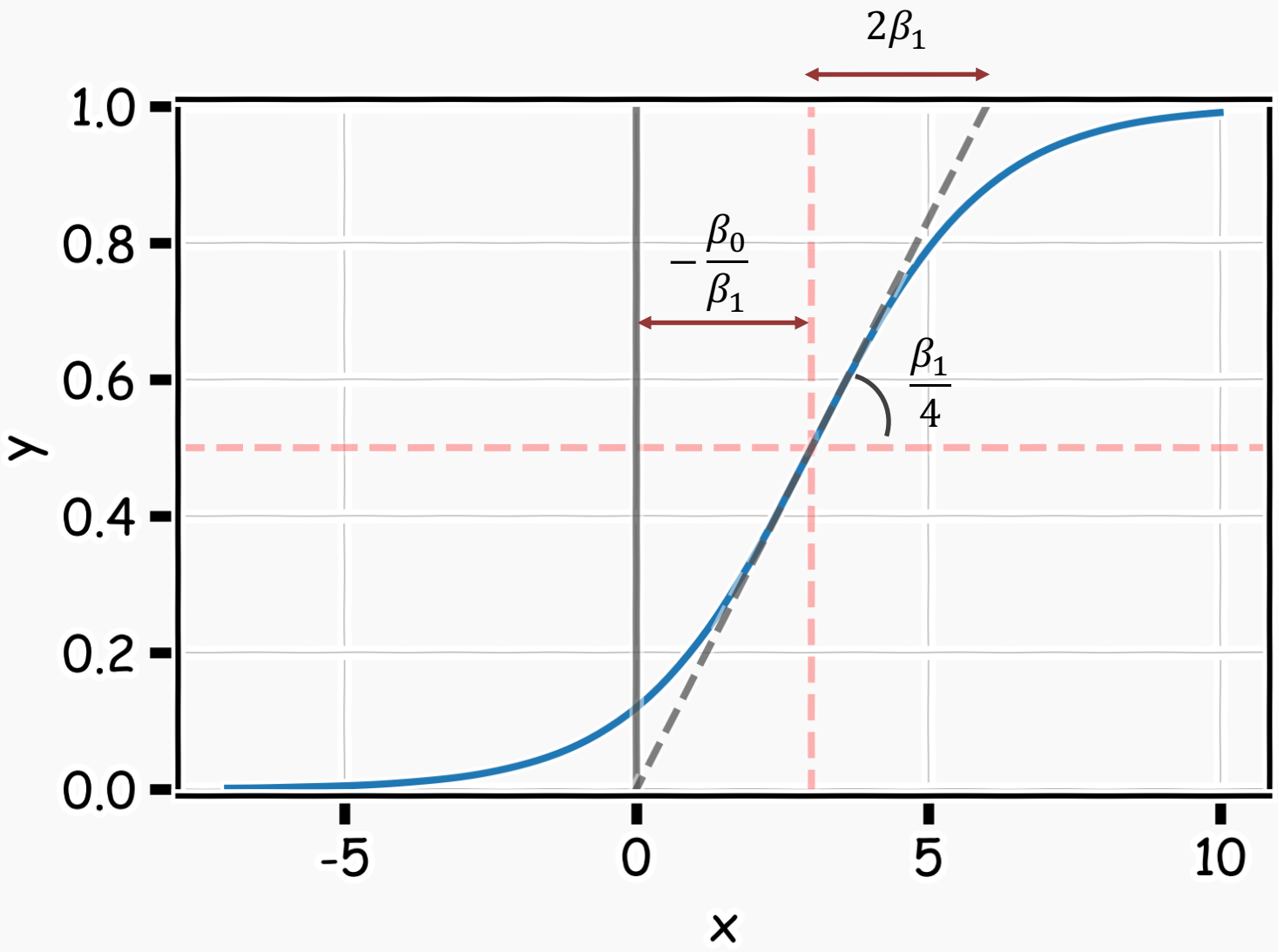
Think of a function that would do this for us



$Y = f(x)$



Logistic Regression



Estimation in Logistic Regression

Probability Mass Function (PMF):

$$P(Y = 1) = p$$
$$P(Y = 0) = 1 - p$$

$$P(Y = y) = p^y (1 - p)^{(1-y)}$$

where:

$$p = P(Y = 1|X = x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

and therefore p depends on X.

Thus not every p_i is the same for each individual measurement.

Likelihood

The likelihood of a single observation for *p* given *x and y* is:

$$L(p_i|Y_i) = P(Y_i = y_i) = p_i^{y_i}(1-p_i)^{1-y_i}$$

Given the observations are independent, what is the likelihood function for *p*?

$$L(p|Y) = \prod_i P(Y_i = y_i) = \prod_i p_i^{y_i}(1-p_i)^{1-y_i}$$

$$l(p|Y) = -\log L(p|Y) = -\sum_i y_i \log p_i + (1-y_i)\log(1-p_i)$$

Loss Function

$$l(p|Y) = -\sum_i \left[y_i \log \frac{1}{1+e^{-(\beta_0+\beta_1 X_i)}} + (1-y_i)\log\left(1 - \frac{1}{1+e^{-(\beta_0+\beta_1 X_i)}}\right)\right]_i$$

How do we minimize this?

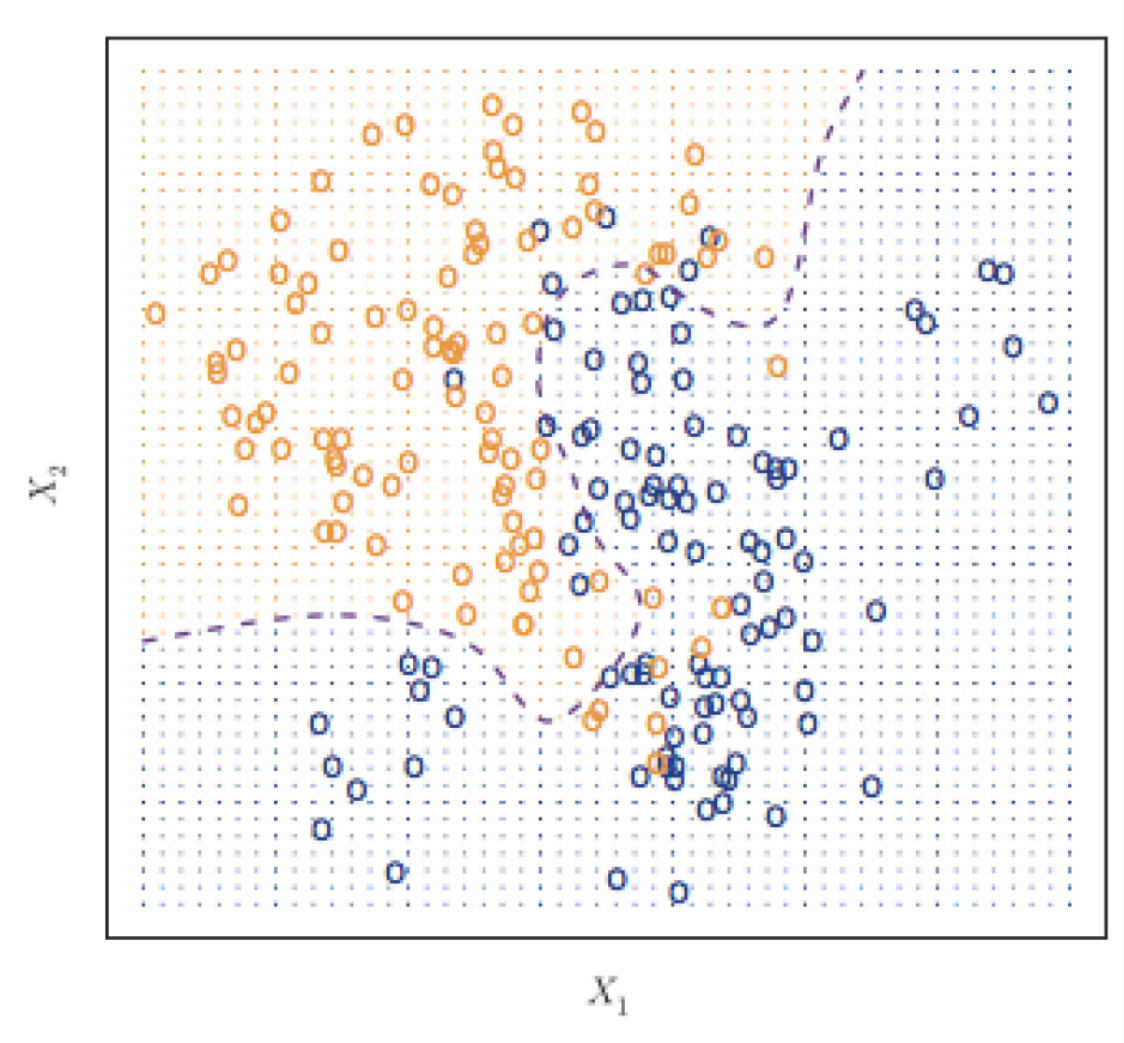
Differentiate, equate to zero and solve for it!

But jeeze does this look messy?! It will not necessarily have a closed form solution.

So how do we determine the parameter estimates? Through an iterative approach (we will talk about this *at length* in future lectures).

Classifier with two predictors

How can we estimate a classifier, based on logistic regression, for the following plot?



Multiple Logistic Regression

Earlier we saw the general form of *simple* logistic regression, meaning when there is just one predictor used in the model. What was the model statement (in terms of linear predictors)?

$$\log\left(\frac{P(Y=1)}{1-P(Y=1)}\right) = \beta_0 + \beta_1 X$$

Multiple logistic regression is a generalization to multiple predictors. More specifically we can define a multiple logistic regression model to predict $P(Y=1)$ as such:

$$\log\left(\frac{P(Y=1)}{1-P(Y=1)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$$

Regularization in Logistic Regression

A penalty factor can then be added to this loss function and results in a new loss function that penalizes large values of the parameters:

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\operatorname{argmin}} \left[-\sum_{i=1}^{n} \left(y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i) \right) + \lambda \sum_{j=1}^{p} \beta_j^2 \right]$$

The result is just like in linear regression: shrink the parameter estimates towards zero.

In practice, the intercept is usually not part of the penalty factor.

Note: the sklearn package uses a different tuning parameter: instead of λ they use a constant that is essentially $C = \frac{1}{\lambda}$.

Diagnostic Testing in Classification (cont.)

Bayes' theorem can be rewritten for classification:

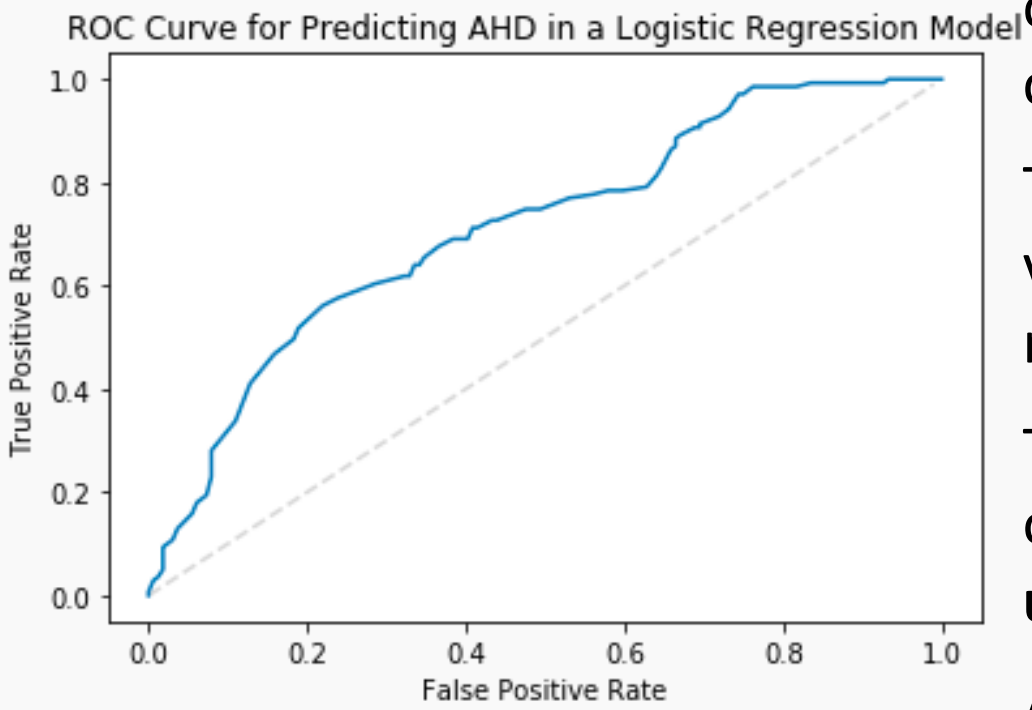
$$P(y=1|\hat{y}=1) = \frac{P(\hat{y}=1|y=1)P(y=1)}{P(\hat{y}=1|y=1)P(y=1) + P(\hat{y}=1|y=0)P(y=0)}$$

These probability quantities can then be defined as:

- *Sensitivity*: $P(\hat{y}=1|y=1)$
- *Specificity*: $P(\hat{y}=0|y=0)$
- *Prevalence*: $P(y=1)$
- *Positive Predictive Value*: $P(y=1|\hat{y}=1)$
- *Negative Predictive Value*: $P(y=0|\hat{y}=0)$

How do positive and negative predictive values relate? Be careful…

ROC Curve Example



The Radio Operator Characteristics (ROC) curve illustrates the trade-off for all possible thresholds chosen for the two types of error (or correct classification).

The vertical axis displays the true positive predictive value and the horizontal axis depicts the true negative predictive value.

The overall performance of a classifier, calculated over all possible thresholds, is given by the **area under the ROC curve** (AUC).

An ideal ROC curve will hug the top left corner, so the larger the AUC the better the classifier.

Multinomial Logistic Regression: the model

To predict K classes ($K > 2$) from a set of predictors X, a multinomial logistic regression can be fit:

$$\ln\left(\frac{P(Y=1)}{P(Y=K)}\right) = \beta_{0,1} + \beta_{1,1}X_1 + \beta_{2,1}X_2 + \cdots + \beta_{p,1}X_p$$

$$\ln\left(\frac{P(Y=2)}{P(Y=K)}\right) = \beta_{0,2} + \beta_{1,2}X_1 + \beta_{2,2}X_2 + \cdots + \beta_{p,2}X_p$$

$$\vdots$$

$$\ln\left(\frac{P(Y=K-1)}{P(Y=K)}\right) = \beta_{0,K-1} + \beta_{1,K-1}X_1 + \beta_{2,K-1}X_2 + \cdots + \beta_{p,K-1}X_p$$

Each separate model can be fit as independent standard logistic regression models!

One vs. Rest (OvR) Logistic Regression: the model

To predict K classes ($K > 2$) from a set of predictors X, a multinomial logistic regression can be fit:

$$\ln\left(\frac{P(Y=1)}{P(Y\neq 1)}\right) = \beta_{0,1} + \beta_{1,1}X_1 + \beta_{2,1}X_2 + \cdots + \beta_{p,1}X_p$$

$$\ln\left(\frac{P(Y=2)}{P(Y\neq 2)}\right) = \beta_{0,2} + \beta_{1,2}X_1 + \beta_{2,2}X_2 + \cdots + \beta_{p,2}X_p$$

$$\vdots$$

$$\ln\left(\frac{P(Y=K)}{P(Y\neq K)}\right) = \beta_{0,K} + \beta_{1,K}X_1 + \beta_{2,K}X_2 + \cdots + \beta_{p,K}X_p$$

Again, each separate model can be fit as independent standard logistic regression models!

Softmax

So how do we convert a set of probability estimates from separate models to one set of probability estimates?

The ***softmax*** function is used. That is, the weights are just normalized for each predicted probability. AKA, predict the 3 class probabilities from each of the 3 models, and just rescale so they add up to 1.

Mathematically that is:

$$P(y = k|\vec{x}) = \frac{e^{\vec{x}^T \hat{\vec{\beta}}_k}}{\sum_{j=1}^{K} e^{\vec{x}^T \hat{\vec{\beta}}_j}}$$

where $\vec{x}$ is the vector of predictors for that observation and $\hat{\vec{\beta}}_k$ are the associated logistic regression coefficient estimates.

Estimation and Regularization in multiclass settings

There is no difference in the approach to estimating the coefficients in the multiclass setting: we maximize the log-likelihood (or minimize negative log-likelihood).

This combined negative log-likelihood of all *K* classes is sometimes called the **multiclass cross-entropy**:

$$\ell = \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} \mathbb{1}(y_i = k)\ln(\hat{P}\,(y_i = k)) + \mathbb{1}(y_i \neq k)\ln(1 - \hat{P}\,(y_i = k))$$

And regularization can be done like always: add on a penalty term to this loss function based on L1 (sum of the absolute values) or L2 (sum of squares) norms.

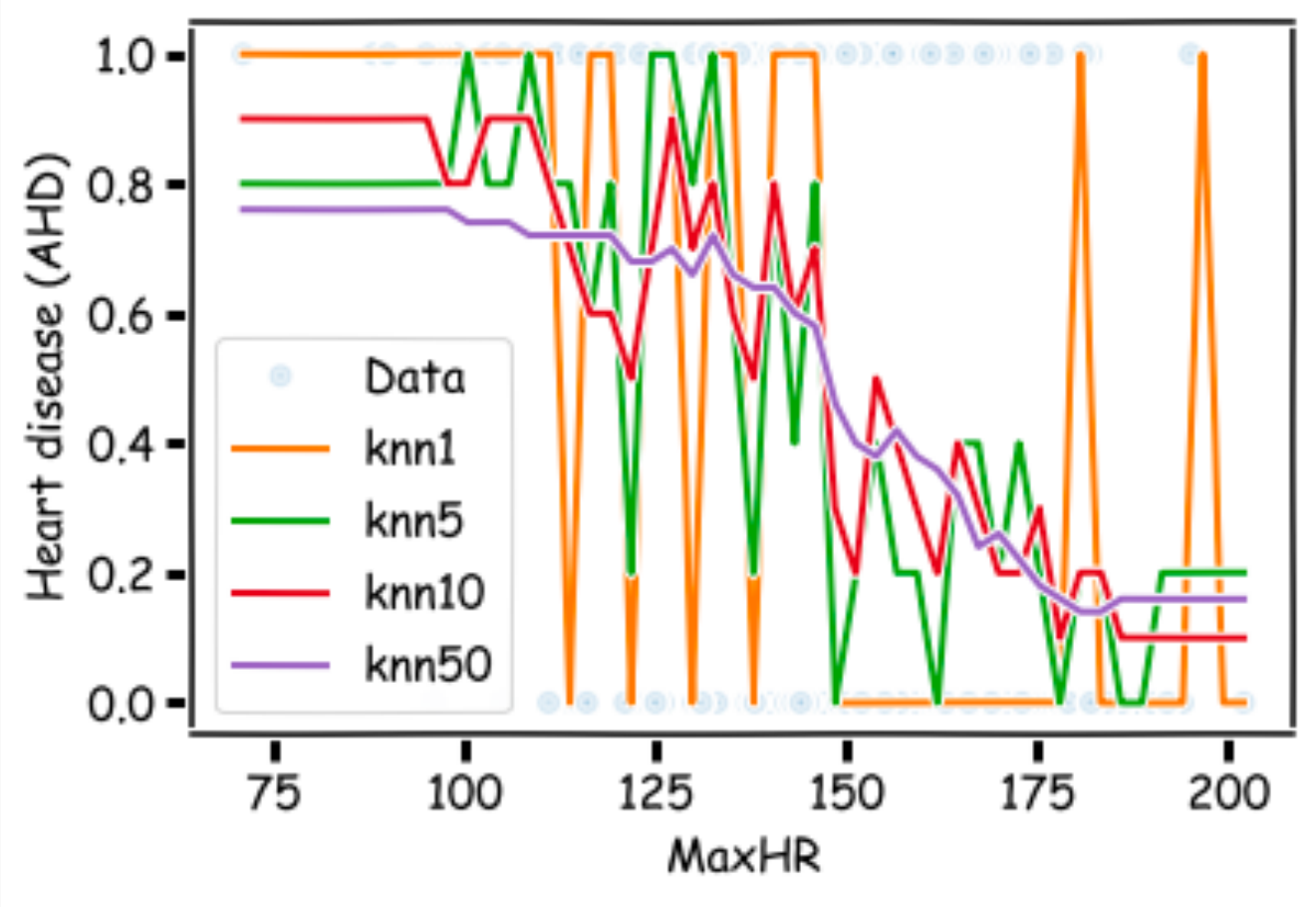
k-NN for Classification: formal definition

The k-NN classifier first identifies the k points in the training data that are closest to x_0, represented by $\mathcal{N}_0$. It then estimates the conditional probability for class j as the fraction of points in $\mathcal{N}_0$ whose response values equal j:

$$P(Y = j | X = x_0) = \frac{1}{k} \sum_{i \in \mathcal{N}_0} I(y_i = j)$$

Then, the k-NN classifier predicts this new observation, x_0, to be in the class with largest estimated probability.

Estimated Probabilities in *k*-NN Classification



Classification: Concept Checks

What is the interpretation of $\hat{\beta}_1$ in a multiple logistic regression problem?

How is the loss function minimized in logistic regression?

What happens in logistic regression when there is *perfect separation*?

Why are the logistic regression classification boundaries linear? How can this be adapted to be non-linear?

What is the loss function that is penalized in regularized multiple logistic regression?

Classification: Concept Checks (cont.)

Should predictors be standardized in *k*-NN? How can categorical predictors be handled?

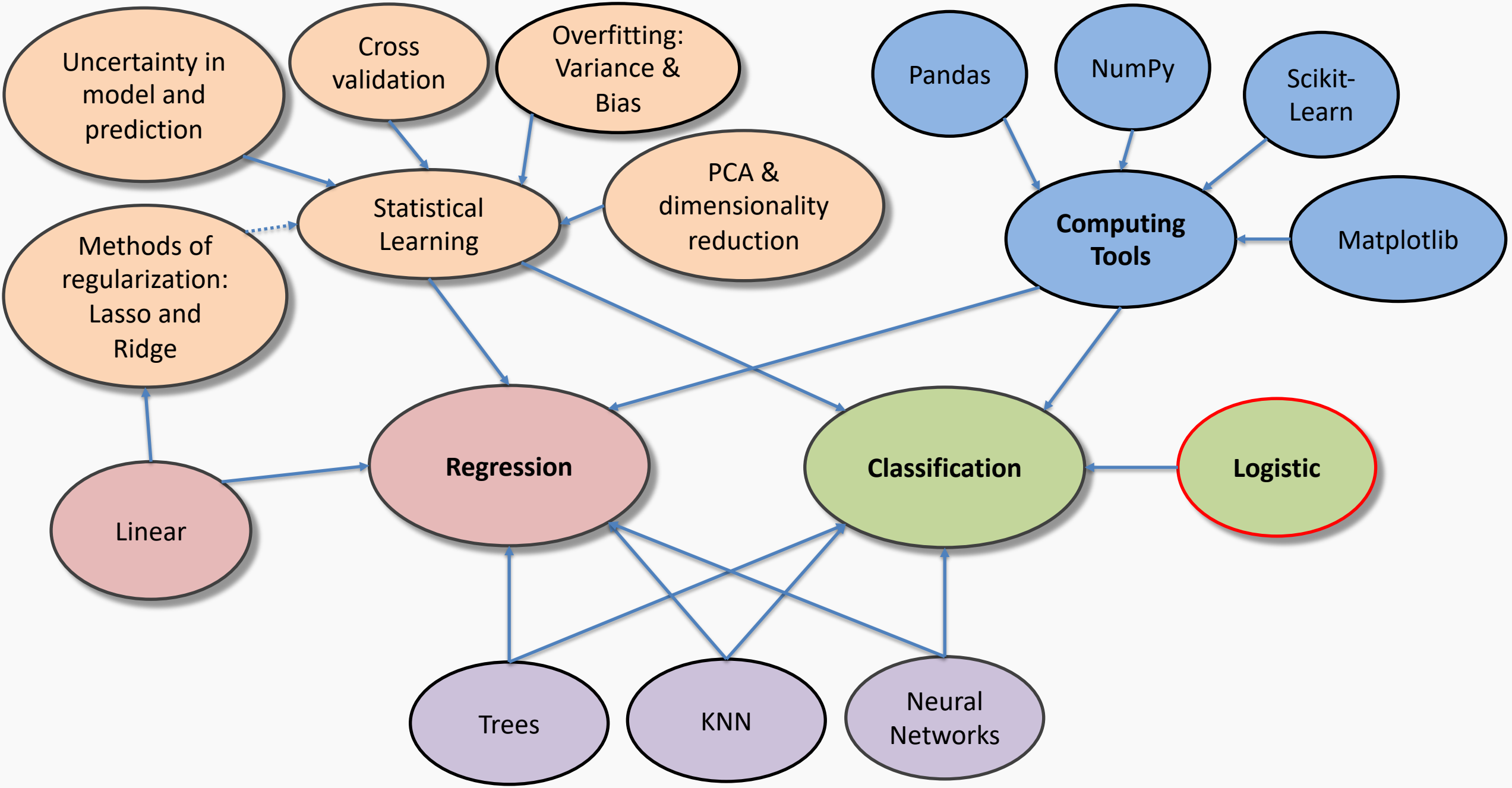
How can probability predictions be made in *k*-NN classification?

When is PCA used most commonly in practice?

Is the first PCA component going to be the best predictor in a PCR regression? Why?

How many PCA components should be used in a PCR egression?

How can the coefficient estimates be interpreted in a regression fit on PCA component vectors?


Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks

Types of Missingness

There are 3 major types of missingness to be concerned about:

1. **Missing Completely at Random (MCAR)** - the probability of missingness in a variable is the same for all units. Like randomly poking holes in a data set.
2. **Missing at Random (MAR)** - the probability of missingness in a variable depends only on available information (in other predictors).
3. **Missing Not at Random (MNAR)** - the probability of missingness depends on information that has not been recorded and this information also predicts the missing values.

What are examples of each these 3 types?

Imputation Methods

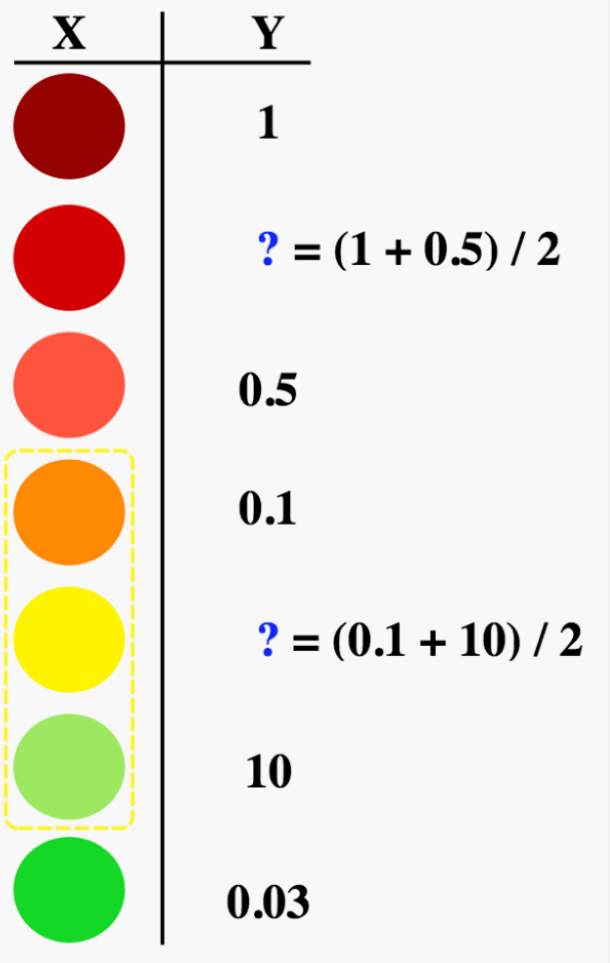
There are several different approaches to imputing missing values:

1. **Impute the mean or median** (quantitative) or most common class (categorical) for all missing values in a variable.
2. Create a new variable that is an **indicator of missingness**, and include it in any model to predict the response (also plug in zero or the mean in the actual variable).
3. **Hot deck imputation**: for each missing entry, randomly select an observed entry in the variable and plug it in.
4. **Model the imputation**: plug in predicted values ($\hat{y}$) from a model based on the other observed predictors.
5. **Model the imputation with uncertainty**: plug in predicted values plus randomness ($\hat{y} + \epsilon$) from a model based on the other observed predictors.

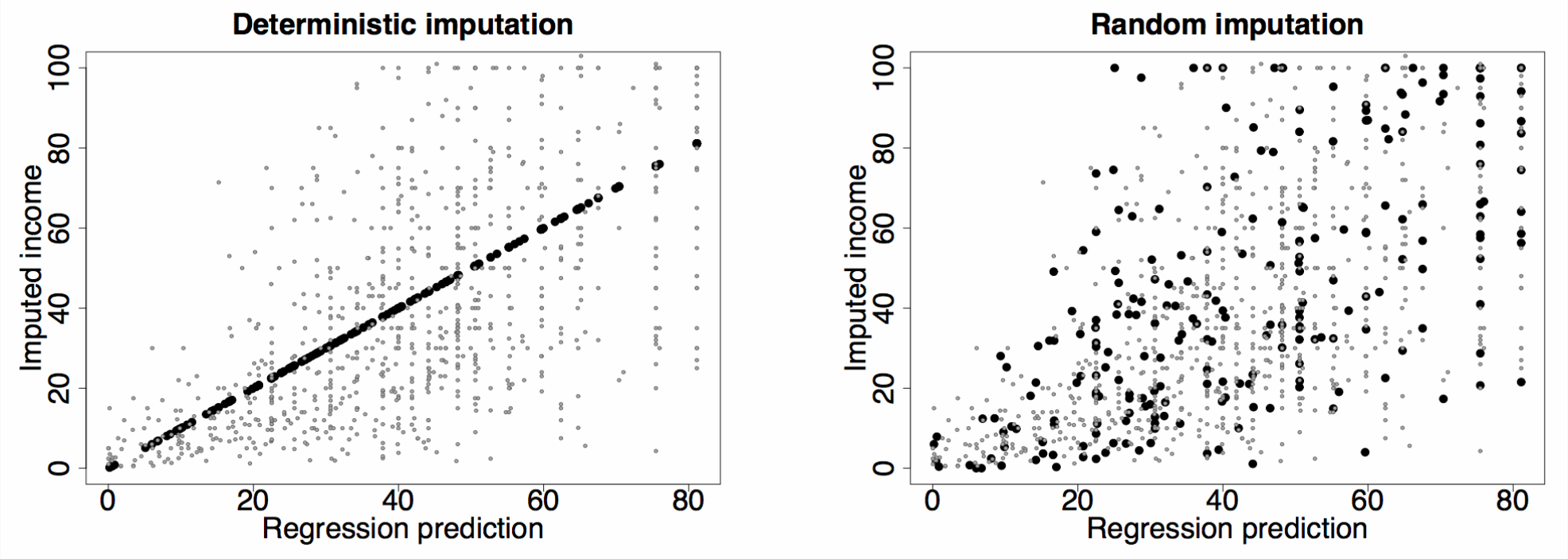
What are the advantages and disadvantages of each approach?

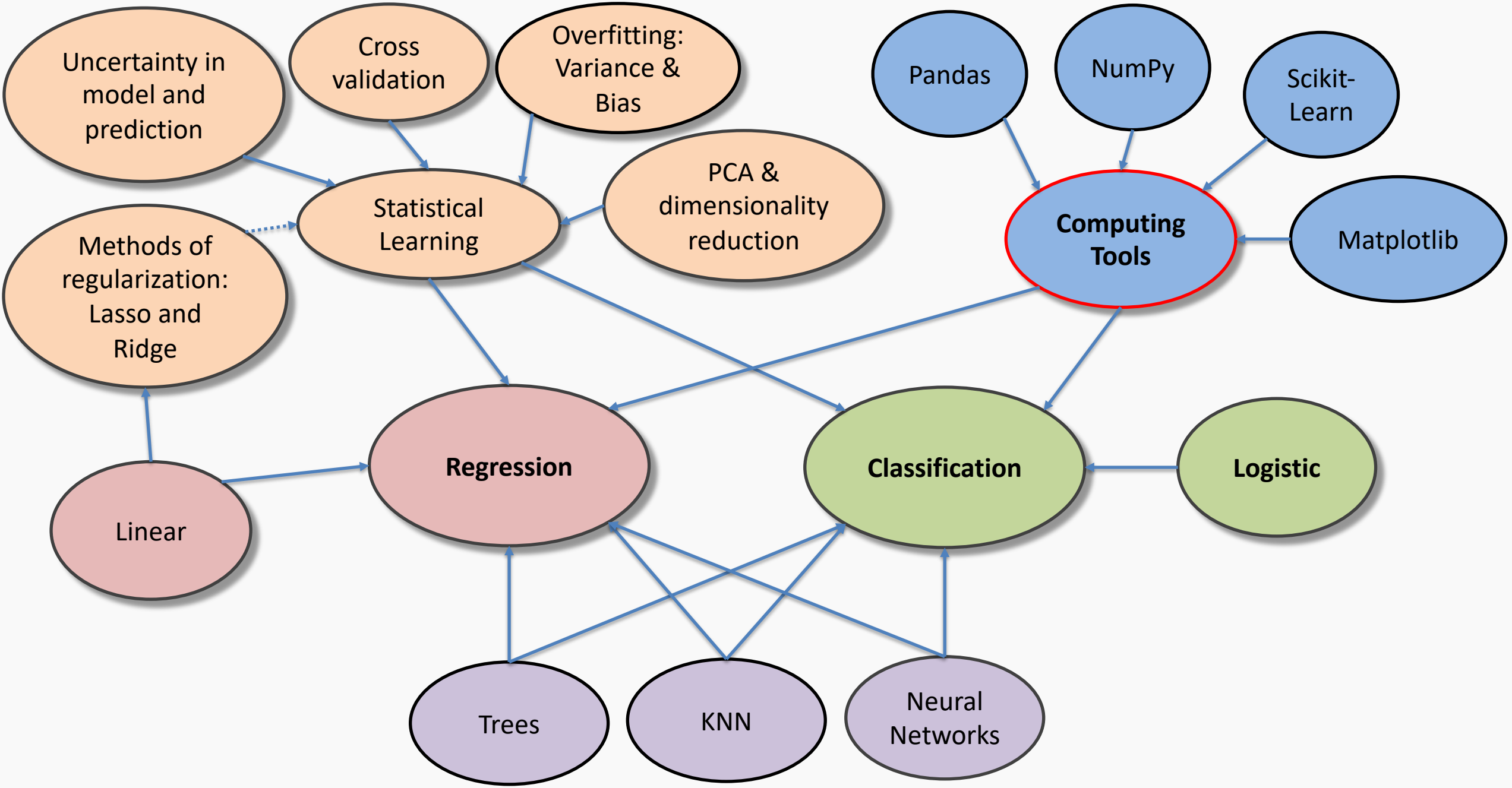
Schematic: imputation through modeling

How do we use models to fill in missing data? Using *k*-NN for *k* = 2?



Imputation through modeling with uncertainty: an illustration




Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks

Python For Data Science *Cheat Sheet*

Pandas Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com

Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

Series

A **one-dimensional** labeled array capable of holding any data type

Index

a	3
b	-5
c	7
d	4

```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame

Columns

Index

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasília	207847528

A **two-dimensional** labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
 -5
```

Get one element

```
>>> df[1:]
   Country     Capital   Population
1   India   New Delhi   1303171035
2  Brazil    Brasília    207847528
```

Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]]
 'Belgium'
>>> df.iat([0],[0])
 'Belgium'
```

Select single value by row & column

By Label

```
>>> df.loc[[0], ['Country']]
 'Belgium'
>>> df.at([0], ['Country'])
 'Belgium'
```

Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
 Country      Brazil
 Capital    Brasília
 Population  207847528
```

Select single row of subset of rows

```
>>> df.ix[:,'Capital']
 0      Brussels
 1     New Delhi
 2      Brasília
```

Select a single column of subset of columns

```
>>> df.ix[1,'Capital']
 'New Delhi'
```

Select rows and columns

Boolean Indexing

```
>>> s[~(s > 1)]
>>> s[(s < -1) | (s > 2)]
>>> df[df['Population']>1200000000]
```

Series s where value is not >1
s where value is <-1 or >2
Use filter to adjust DataFrame

Setting

```
>>> s['a'] = 6
```

Set index a of Series s to 6

Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
```

Drop values from rows (axis=0)
Drop values from columns(axis=1)

Sort & Rank

```
>>> df.sort_index()
>>> df.sort_values(by='Country')
>>> df.rank()
```

Sort by labels along an axis
Sort by the values along an axis
Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```

(rows,columns)
Describe index
Describe DataFrame columns
Info on DataFrame
Number of non-NA values

Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idxmin()/df.idxmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

Sum of values
Cummulative sum of values
Minimum/maximum values
Minimum/Maximum index value
Summary statistics
Mean of values
Median of values

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function
Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
 a      10.0
 b       NaN
 c       5.0
 d       7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
 a     10.0
 b     -5.0
 c      5.0
 d      7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

`read_sql()` is a convenience wrapper around `read_sql_table()` and `read_sql_query()`

```
>>> pd.to_sql('myDf', engine)
```

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com

NumPy

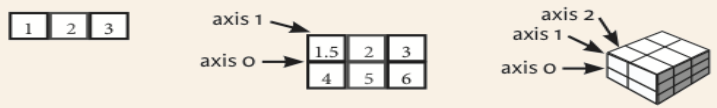
The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays

1D array **2D array** **3D array**



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

`>>> np.zeros((3,4))`	Create an array of zeros
`>>> np.ones((2,3,4),dtype=np.int16)`	Create an array of ones
`>>> d = np.arange(10,25,5)`	Create an array of evenly spaced values (step value)
`>>> np.linspace(0,2,9)`	Create an array of evenly spaced values (number of samples)
`>>> e = np.full((2,2),7)`	Create a constant array
`>>> f = np.eye(2)`	Create a 2X2 identity matrix
`>>> np.random.random((2,2))`	Create an array with random values
`>>> np.empty((3,2))`	Create an empty array

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

`>>> np.int64`	Signed 64-bit integer types
`>>> np.float32`	Standard double-precision floating point
`>>> np.complex`	Complex numbers represented by 128 floats
`>>> np.bool`	Boolean type storing `TRUE` and `FALSE` values
`>>> np.object`	Python object type
`>>> np.string_`	Fixed-length string type
`>>> np.unicode_`	Fixed-length unicode type

Inspecting Your Array

`>>> a.shape`	Array dimensions
`>>> len(a)`	Length of array
`>>> b.ndim`	Number of array dimensions
`>>> e.size`	Number of array elements
`>>> b.dtype`	Data type of array elements
`>>> b.dtype.name`	Name of data type
`>>> b.astype(int)`	Convert an array to a different type

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

`>>> g = a - b`
`array([[-0.5, 0. , 0. ],`
`[-3. , -3. , -3. ]])`
	Subtraction

`>>> np.subtract(a,b)`	Subtraction

`>>> b + a`
`array([[ 2.5, 4. , 6. ],`
`[ 5. , 7. , 9. ]])`
	Addition

`>>> np.add(b,a)`	Addition

`>>> a / b`
`array([[ 0.66666667, 1. , 1. ],`
`[ 0.25 , 0.4 , 0.5 ]])`
	Division

`>>> np.divide(a,b)`	Division

`>>> a * b`
`array([[ 1.5, 4. , 9. ],`
`[ 4. , 10. , 18. ]])`
	Multiplication

`>>> np.multiply(a,b)`	Multiplication
`>>> np.exp(b)`	Exponentiation
`>>> np.sqrt(b)`	Square root
`>>> np.sin(a)`	Print sines of an array
`>>> np.cos(b)`	Element-wise cosine
`>>> np.log(a)`	Element-wise natural logarithm

`>>> e.dot(f)`
`array([[ 7., 7.],`
`[ 7., 7.]])`
	Dot product

Comparison

`>>> a == b`
`array([[False, True, True],`
`[False, False, False]], dtype=bool)`
	Element-wise comparison

`>>> a < 2` `array([True, False, False], dtype=bool)`	Element-wise comparison
`>>> np.array_equal(a, b)`	Array-wise comparison

Aggregate Functions

`>>> a.sum()`	Array-wise sum
`>>> a.min()`	Array-wise minimum value
`>>> b.max(axis=0)`	Maximum value of an array row
`>>> b.cumsum(axis=1)`	Cumulative sum of the elements
`>>> a.mean()`	Mean
`>>> b.median()`	Median
`>>> a.corrcoef()`	Correlation coefficient
`>>> np.std(b)`	Standard deviation

Copying Arrays

`>>> h = a.view()`	Create a view of the array with the same data
`>>> np.copy(a)`	Create a copy of the array
`>>> h = a.copy()`	Create a deep copy of the array

Sorting Arrays

`>>> a.sort()`	Sort an array
`>>> c.sort(axis=0)`	Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

`>>> a[2]` `3`	1 2 3	Select the element at the 2nd index
`>>> b[1,2]` `6.0`	1.5 2 3 4 5 6	Select the element at row 0 column 2 (equivalent to `b[1][2]`)

Slicing

`>>> a[0:2]` `array([1, 2])`	1 2 3	Select items at index 0 and 1
`>>> b[0:2,1]` `array([ 2., 5.])`	1.5 2 3 4 5 6	Select items at rows 0 and 1 in column 1
`>>> b[:1]` `array([[1.5, 2., 3.]])`	1.5 2 3 4 5 6	Select all items at row 0 (equivalent to `b[0:1, :]`)

`>>> c[1,...]`
`array([[[ 3., 2., 1.],`
`[ 4., 5., 6.]]])`
		Same as `[1,:,:]`

`>>> a[ : :-1]` `array([3, 2, 1])`		Reversed array `a`

Boolean Indexing

`>>> a[a<2]` `array([1])`	1 2 3	Select elements from `a` less than 2

Fancy Indexing

`>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]` `array([ 4. , 2. , 6. , 1.5])`	Select elements `(1,0)`, `(0,1)`, `(1,2)` and `(0,0)`

`>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]`
`array([[ 4. ,5. , 6. , 4. ],`
`[ 1.5, 2. , 3. , 1.5],`
`[ 4. , 5. , 6. , 4. ],`
`[ 1.5, 2. , 3. , 1.5]])`
	Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

`>>> i = np.transpose(b)`	Permute array dimensions
`>>> i.T`	Permute array dimensions

Changing Array Shape

`>>> b.ravel()`	Flatten the array
`>>> g.reshape(3,-2)`	Reshape, but don't change data

Adding/Removing Elements

`>>> h.resize((2,6))`	Return a new array with shape (2,6)
`>>> np.append(h,g)`	Append items to an array
`>>> np.insert(a, 1, 5)`	Insert items in an array
`>>> np.delete(a,[1])`	Delete items from an array

Combining Arrays

`>>> np.concatenate((a,d),axis=0)` `array([ 1, 2, 3, 10, 15, 20])`	Concatenate arrays

`>>> np.vstack((a,b))`
`array([[ 1. , 2. , 3. ],`
`[ 1.5, 2. , 3. ],`
`[ 4. , 5. , 6. ]])`
	Stack arrays vertically (row-wise)

`>>> np.r_[e,f]`	Stack arrays vertically (row-wise)

`>>> np.hstack((e,f))`
`array([[ 7., 7., 1., 0.],`
`[ 7., 7., 0., 1.]])`
	Stack arrays horizontally (column-wise)

`>>> np.column_stack((a,d))`
`array([[ 1, 10],`
`[ 2, 15],`
`[ 3, 20]])`
	Create stacked column-wise arrays

`>>> np.c_[a,d]`	Create stacked column-wise arrays

Splitting Arrays

`>>> np.hsplit(a,3)` `[array([1]),array([2]),array([3])]`	Split the array horizontally at the 3rd index

`>>> np.vsplit(c,2)`
`[array([[[ 1.5, 2. , 1. ],`
`[ 4. , 5. , 6. ]]]),`
`array([[[ 3., 2., 3.],`
`[ 4., 5., 6.]]])]`
	Split the array vertically at the 2nd index

Python For Data Science *Cheat Sheet*

SciPy - Linear Algebra

Learn More Python for Data Science Interactively at www.datacamp.com

SciPy

The **SciPy** library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.

Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])
```

Index Tricks

`>>> np.mgrid[0:5,0:5]`	Create a dense meshgrid
`>>> np.ogrid[0:2,0:2]`	Create an open meshgrid
`>>> np.r_[3,[0]*5,-1:1:10j]`	Stack arrays vertically (row-wise)
`>>> np.c_[b,c]`	Create stacked column-wise arrays

Shape Manipulation

`>>> np.transpose(b)`	Permute array dimensions
`>>> b.flatten()`	Flatten the array
`>>> np.hstack((b,c))`	Stack arrays horizontally (column-wise)
`>>> np.vstack((a,b))`	Stack arrays vertically (row-wise)
`>>> np.hsplit(c,2)`	Split the array horizontally at the 2nd index
`>>> np.vpslit(d,2)`	Split the array vertically at the 2nd index

Polynomials

`>>> from numpy import poly1d`	
`>>> p = poly1d([3,4,5])`	Create a polynomial object

Vectorizing Functions

`>>> def myfunc(a):`
`  if a < 0:`
`  return a*2`
`  else:`
`  return a/2`	

`>>> np.vectorize(myfunc)`	Vectorize functions

Type Handling

`>>> np.real(b)`	Return the real part of the array elements
`>>> np.imag(b)`	Return the imaginary part of the array elements
`>>> np.real_if_close(c,tol=1000)`	Return a real array if complex parts close to 0
`>>> np.cast['f'](np.pi)`	Cast object to a data type

Other Useful Functions

`>>> np.angle(b,deg=True)`	Return the angle of the complex argument
`>>> g = np.linspace(0,np.pi,num=5)`	Create an array of evenly spaced values (number of samples)
`>>> g [3:] += np.pi`	
`>>> np.unwrap(g)`	Unwrap
`>>> np.logspace(0,10,3)`	Create an array of evenly spaced values (log scale)
`>>> np.select([c<4],[c*2])`	Return values from a list of arrays depending on conditions
`>>> misc.factorial(a)`	Factorial
`>>> misc.comb(10,3,exact=True)`	Combine N things taken at k time
`>>> misc.central_diff_weights(3)`	Weights for Np-point central derivative
`>>> misc.derivative(myfunc,1.0)`	Find the n-th derivative of a function at a point

Linear Algebra

Also see NumPy

You'll use the `linalg` and `sparse` modules. Note that `scipy.linalg` contains and expands on `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

Basic Matrix Routines

Inverse	
`>>> A.I`	Inverse
`>>> linalg.inv(A)`	Inverse
Transposition	
`>>> A.T`	Tranpose matrix
`>>> A.H`	Conjugate transposition
Trace	
`>>> np.trace(A)`	Trace
Norm	
`>>> linalg.norm(A)`	Frobenius norm
`>>> linalg.norm(A,1)`	L1 norm (max column sum)
`>>> linalg.norm(A,np.inf)`	L inf norm (max row sum)
Rank	
`>>> np.linalg.matrix_rank(C)`	Matrix rank
Determinant	
`>>> linalg.det(A)`	Determinant
Solving linear problems	
`>>> linalg.solve(A,b)`	Solver for dense matrices
`>>> E = np.mat(a).T`	Solver for dense matrices
`>>> linalg.lstsq(F,E)`	Least-squares solution to linear matrix equation
Generalized inverse	
`>>> linalg.pinv(C)`	Compute the pseudo-inverse of a matrix (least-squares solver)
`>>> linalg.pinv2(C)`	Compute the pseudo-inverse of a matrix (SVD)

Creating Sparse Matrices

`>>> F = np.eye(3, k=1)`	Create a 2X2 identity matrix
`>>> G = np.mat(np.identity(2))`	Create a 2x2 identity matrix
`>>> C[C > 0.5] = 0`	
`>>> H = sparse.csr_matrix(C)`	Compressed Sparse Row matrix
`>>> I = sparse.csc_matrix(D)`	Compressed Sparse Column matrix
`>>> J = sparse.dok_matrix(A)`	Dictionary Of Keys matrix
`>>> E.todense()`	Sparse matrix to full matrix
`>>> sparse.isspmatrix_csc(A)`	Identify sparse matrix

Sparse Matrix Routines

Inverse	
`>>> sparse.linalg.inv(I)`	Inverse
Norm	
`>>> sparse.linalg.norm(I)`	Norm
Solving linear problems	
`>>> sparse.linalg.spsolve(H,I)`	Solver for sparse matrices

Sparse Matrix Functions

`>>> sparse.linalg.expm(I)`	Sparse matrix exponential

Matrix Functions

Addition	
`>>> np.add(A,D)`	Addition
Subtraction	
`>>> np.subtract(A,D)`	Subtraction
Division	
`>>> np.divide(A,D)`	Division
Multiplication	
`>>> A @ D`	Multiplication operator (Python 3)
`>>> np.multiply(D,A)`	Multiplication
`>>> np.dot(A,D)`	Dot product
`>>> np.vdot(A,D)`	Vector dot product
`>>> np.inner(A,D)`	Inner product
`>>> np.outer(A,D)`	Outer product
`>>> np.tensordot(A,D)`	Tensor dot product
`>>> np.kron(A,D)`	Kronecker product
Exponential Functions	
`>>> linalg.expm(A)`	Matrix exponential
`>>> linalg.expm2(A)`	Matrix exponential (Taylor Series)
`>>> linalg.expm3(D)`	Matrix exponential (eigenvalue decomposition)
Logarithm Function	
`>>> linalg.logm(A)`	Matrix logarithm
Trigonometric Functions	
`>>> linalg.sinm(D)`	Matrix sine
`>>> linalg.cosm(D)`	Matrix cosine
`>>> linalg.tanm(A)`	Matrix tangent
Hyperbolic Trigonometric Functions	
`>>> linalg.sinhm(D)`	Hypberbolic matrix sine
`>>> linalg.coshm(D)`	Hyperbolic matrix cosine
`>>> linalg.tanhm(A)`	Hyperbolic matrix tangent
Matrix Sign Function	
`>>> np.signm(A)`	Matrix sign function
Matrix Square Root	
`>>> linalg.sqrtm(A)`	Matrix square root
Arbitrary Functions	
`>>> linalg.funm(A, lambda x: x*x)`	Evaluate matrix function

Decompositions

Eigenvalues and Eigenvectors	
`>>> la, v = linalg.eig(A)`	Solve ordinary or generalized eigenvalue problem for square matrix
`>>> l1, l2 = la`	Unpack eigenvalues
`>>> v[:,0]`	First eigenvector
`>>> v[:,1]`	Second eigenvector
`>>> linalg.eigvals(A)`	Unpack eigenvalues
Singular Value Decomposition	
`>>> U,s,Vh = linalg.svd(B)`	Singular Value Decomposition (SVD)
`>>> M,N = B.shape`	
`>>> Sig = linalg.diagsvd(s,M,N)`	Construct sigma matrix in SVD
LU Decomposition	
`>>> P,L,U = linalg.lu(C)`	LU Decomposition

Sparse Matrix Decompositions

`>>> la, v = sparse.linalg.eigs(F,1)`	Eigenvalues and eigenvectors
`>>> sparse.linalg.svds(H, 2)`	SVD

Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

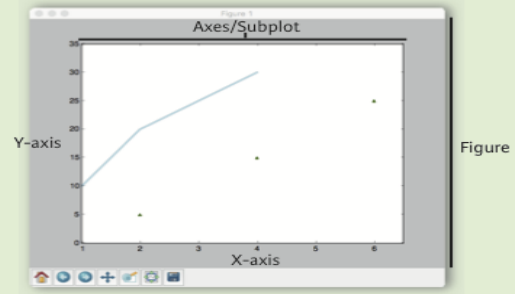
Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

3 Plotting Routines

1D Data

`>>> fig, ax = plt.subplots()`	
`>>> lines = ax.plot(x,y)`	Draw points with lines or markers connecting them
`>>> ax.scatter(x,y)`	Draw unconnected points, scaled or colored
`>>> axes[0,0].bar([1,2,3],[3,4,5])`	Plot vertical rectangles (constant width)
`>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])`	Plot horiontal rectangles (constant height)
`>>> axes[1,1].axhline(0.45)`	Draw a horizontal line across axes
`>>> axes[0,1].axvline(0.65)`	Draw a vertical line across axes
`>>> ax.fill(x,y,color='blue')`	Draw filled polygons
`>>> ax.fill_between(x,y,color='yellow')`	Fill between y-values and o

Vector Fields

`>>> axes[0,1].arrow(0,0,0.5,0.5)`	Add an arrow to the axes
`>>> axes[1,1].quiver(y,z)`	Plot a 2D field of arrows
`>>> axes[0,1].streamplot(X,Y,U,V)`	Plot a 2D field of arrows

Data Distributions

`>>> ax1.hist(y)`	Plot a histogram
`>>> ax3.boxplot(y)`	Make a box and whisker plot
`>>> ax3.violinplot(z)`	Make a violin plot

2D Data or Images

`>>> fig, ax = plt.subplots()`	
`>>> im = ax.imshow(img, cmap='gist_earth', interpolation='nearest', vmin=-2, vmax=2)`	Colormapped or RGB arrays
`>>> axes2[0].pcolor(data2)`	Pseudocolor plot of 2D array
`>>> axes2[0].pcolormesh(data)`	Pseudocolor plot of 2D array
`>>> CS = plt.contour(Y,X,U)`	Plot contours
`>>> axes2[2].contourf(data1)`	Plot filled contours
`>>> axes2[2]= ax.clabel(CS)`	Label a contour plot

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling	
`>>> ax.margins(x=0.0,y=0.1)`	Add padding to a plot
`>>> ax.axis('equal')`	Set the aspect ratio of the plot to 1
`>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])`	Set limits for x-and y-axis
`>>> ax.set_xlim(0,10.5)`	Set limits for x-axis
Legends	
`>>> ax.set(title='An Example Axes', ylabel='Y-Axis', xlabel='X-Axis')`	Set a title and x-and y-axis labels
`>>> ax.legend(loc='best')`	No overlapping plot elements
Ticks	
`>>> ax.xaxis.set(ticks=range(1,5), ticklabels=[3,100,-12,"foo"])`	Manually set x-ticks
`>>> ax.tick_params(axis='y', direction='inout', length=10)`	Make y-ticks longer and go in and out
Subplot Spacing	
`>>> fig3.subplots_adjust(wspace=0.5, hspace=0.3, left=0.125, right=0.9, top=0.9, bottom=0.1)`	Adjust the spacing between subplots
`>>> fig.tight_layout()`	Fit subplot(s) in to the figure area
Axis Spines	
`>>> ax1.spines['top'].set_visible(False)`	Make the top axis line for a plot invisible
`>>> ax1.spines['bottom'].set_position(('outward',10))`	Move the bottom axis line outward

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

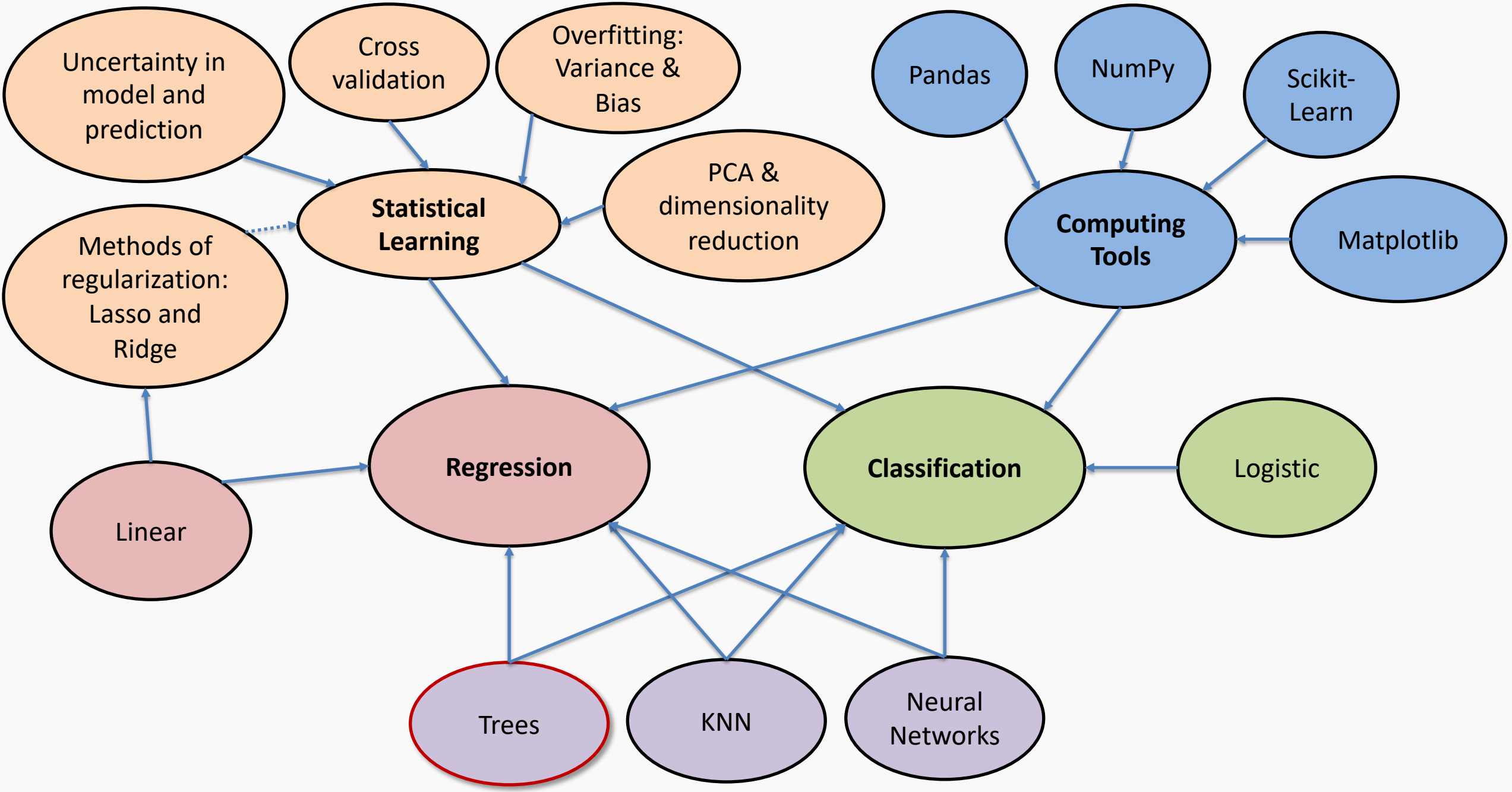
```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

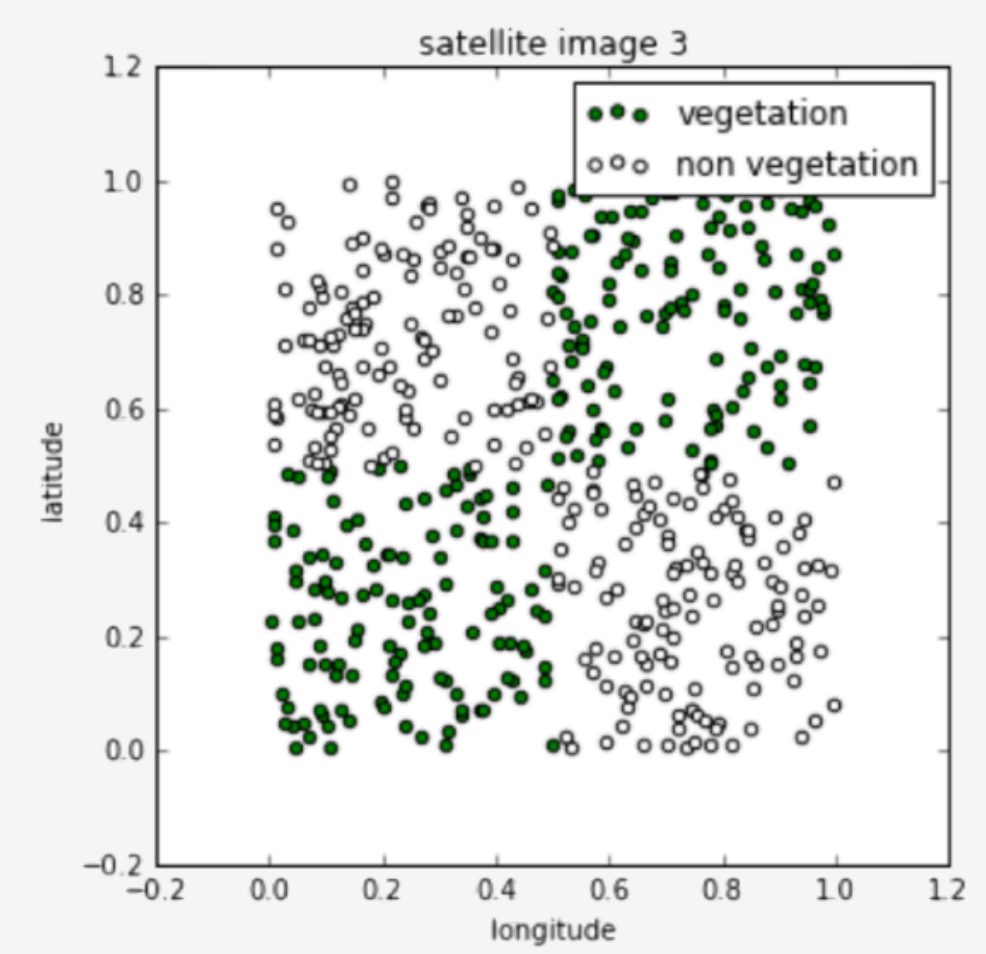
```
>>> plt.show()
```

Close & Clear

`>>> plt.cla()`	Clear an axis
`>>> plt.clf()`	Clear the entire figure
`>>> plt.close()`	Close a window


Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks

SIMPLE DECISION TREE



Although **regression** models with linear boundaries are intuitive to interpret, it's harder to interpret non-linear decision boundaries.

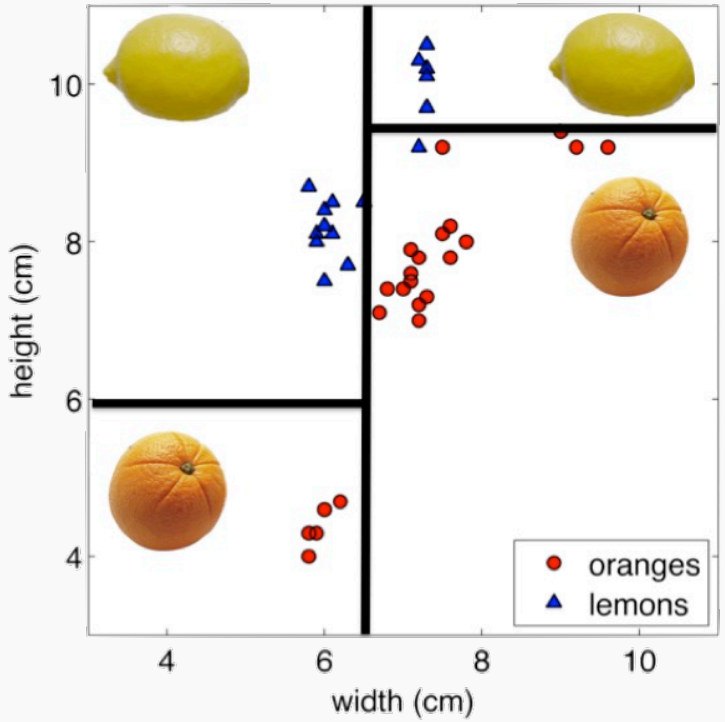
Trees:

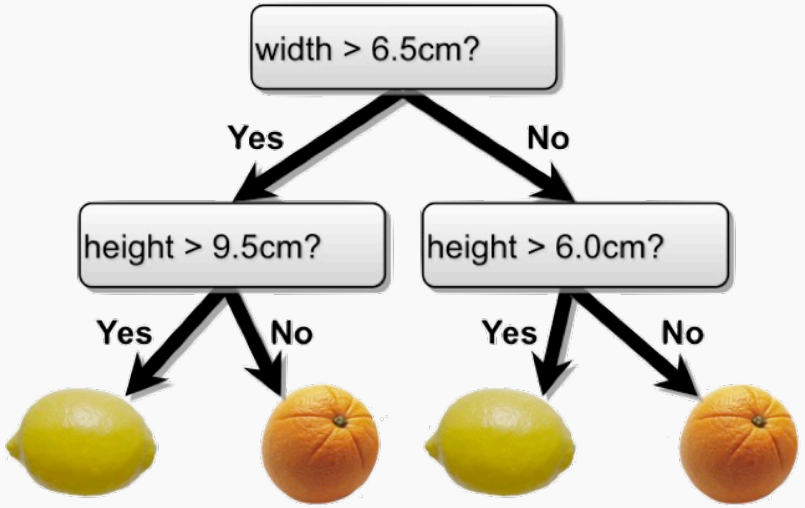
1. Allow for complex decision boundaries
2. Are easy to interpret

The Geometry of Flow Charts

Each comparison and branching represents splitting a region in the feature space on <u>a single feature</u>.

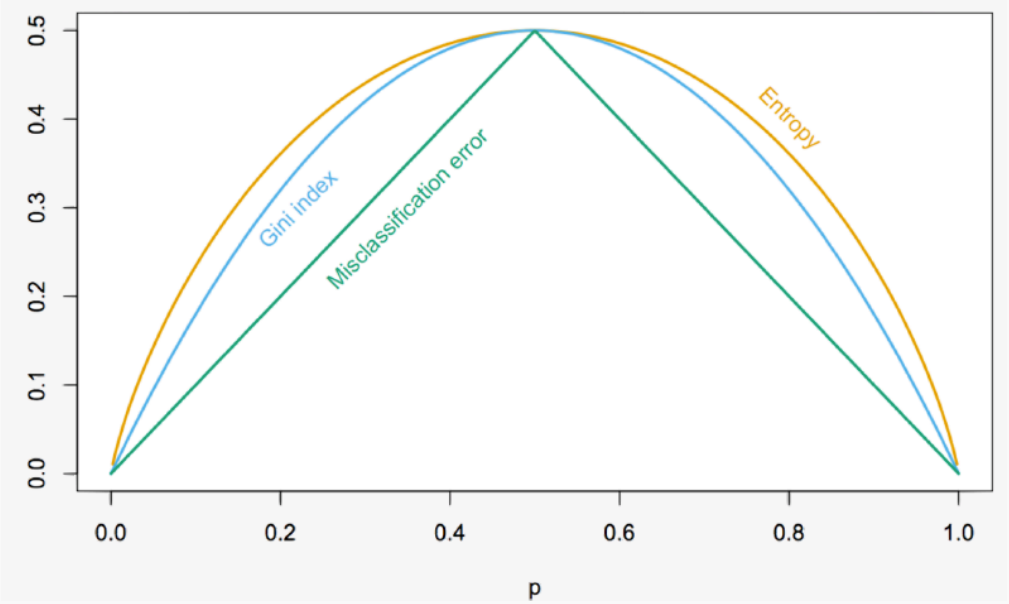
The prediction is based on the most common class (or mean value).





Considerations

1. Splitting Criterion. e.g.,
   - Gini Index
   - misclassification error
   - Entropy



2. Stopping Criterion. e.g.,
   - Minimum MSE
   - Uniformity of the data samples' labels
   - Size of tree, such as maximum depth
   - The "gain" converges

Considerations

Shallow trees have: high bias and low variance

Deep trees have: low bias and high variance

Simple decision trees often:

- Overfit
- Underperform when compared to other classification and regression methods 

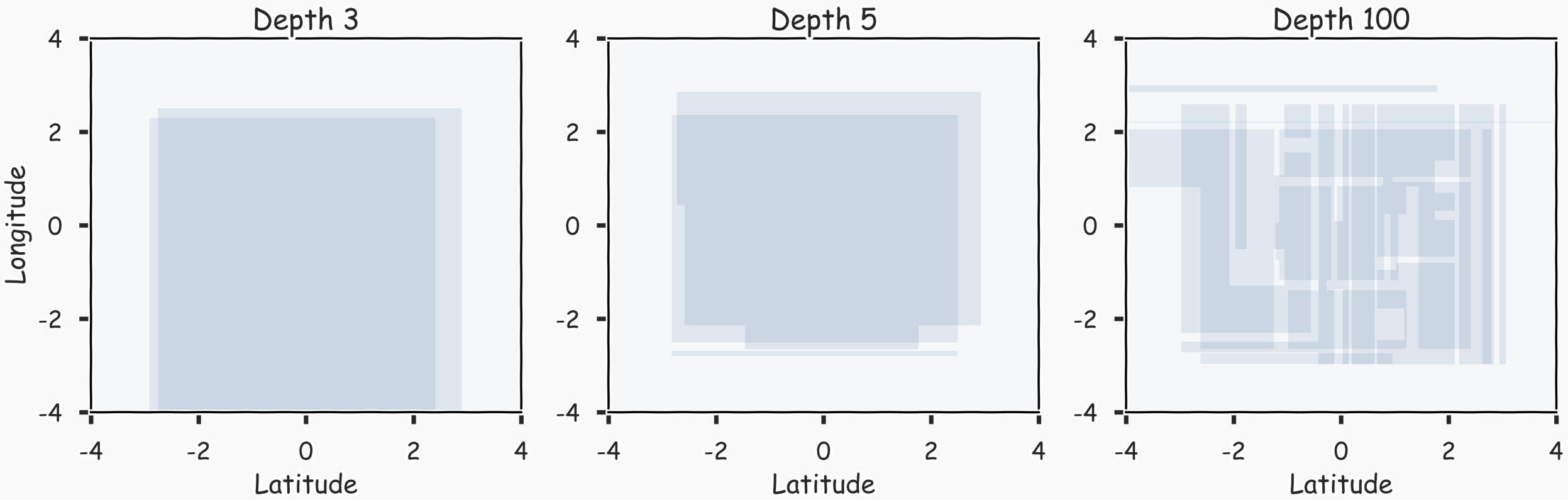
BAGGING

Bootstrap Aggregating

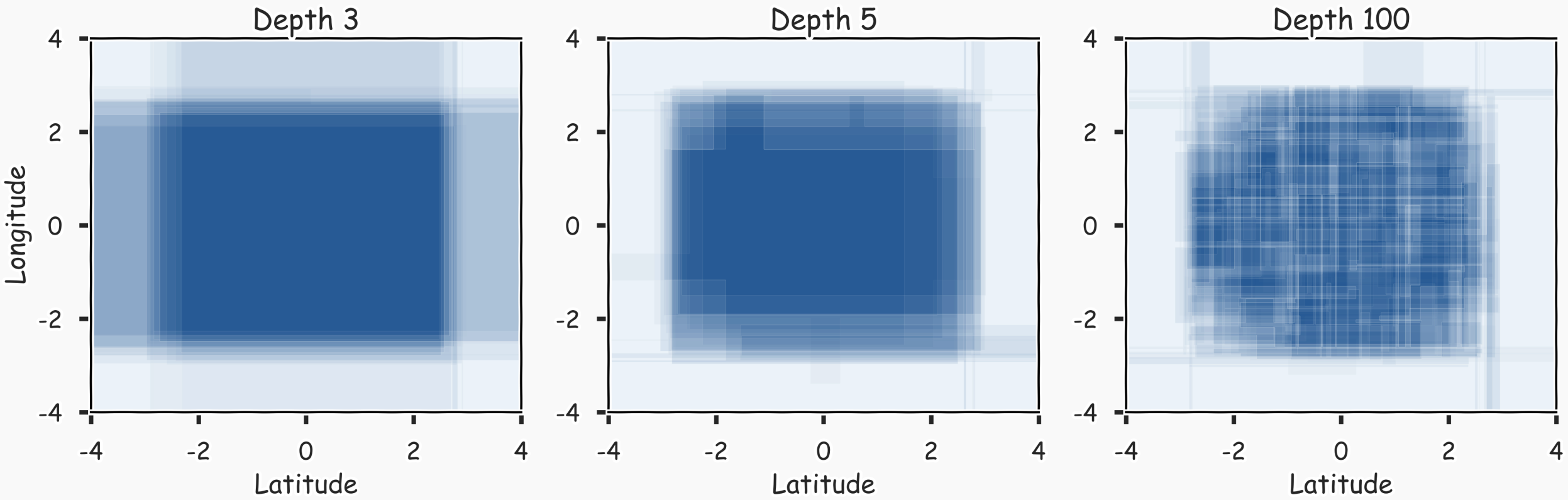
Bootstrap = generate data via sampling w/ replacement

Aggregating = return the average (regression) or majority class (classification)

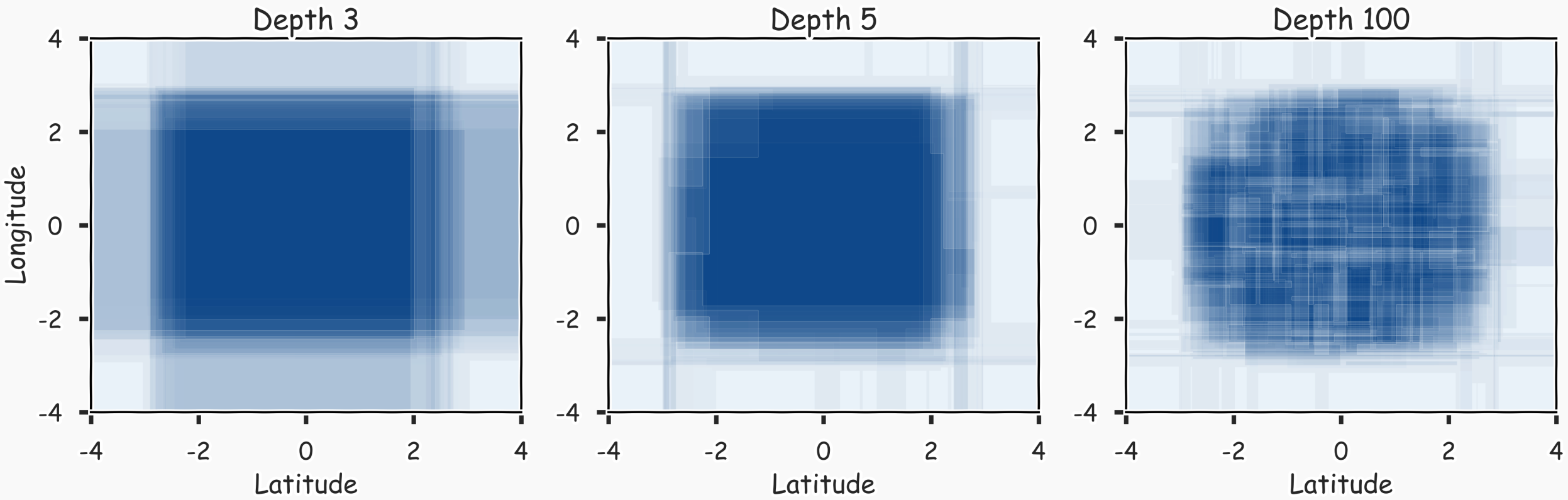
Combine them? 2 magic realisms



Combine them? 20 magic realisms

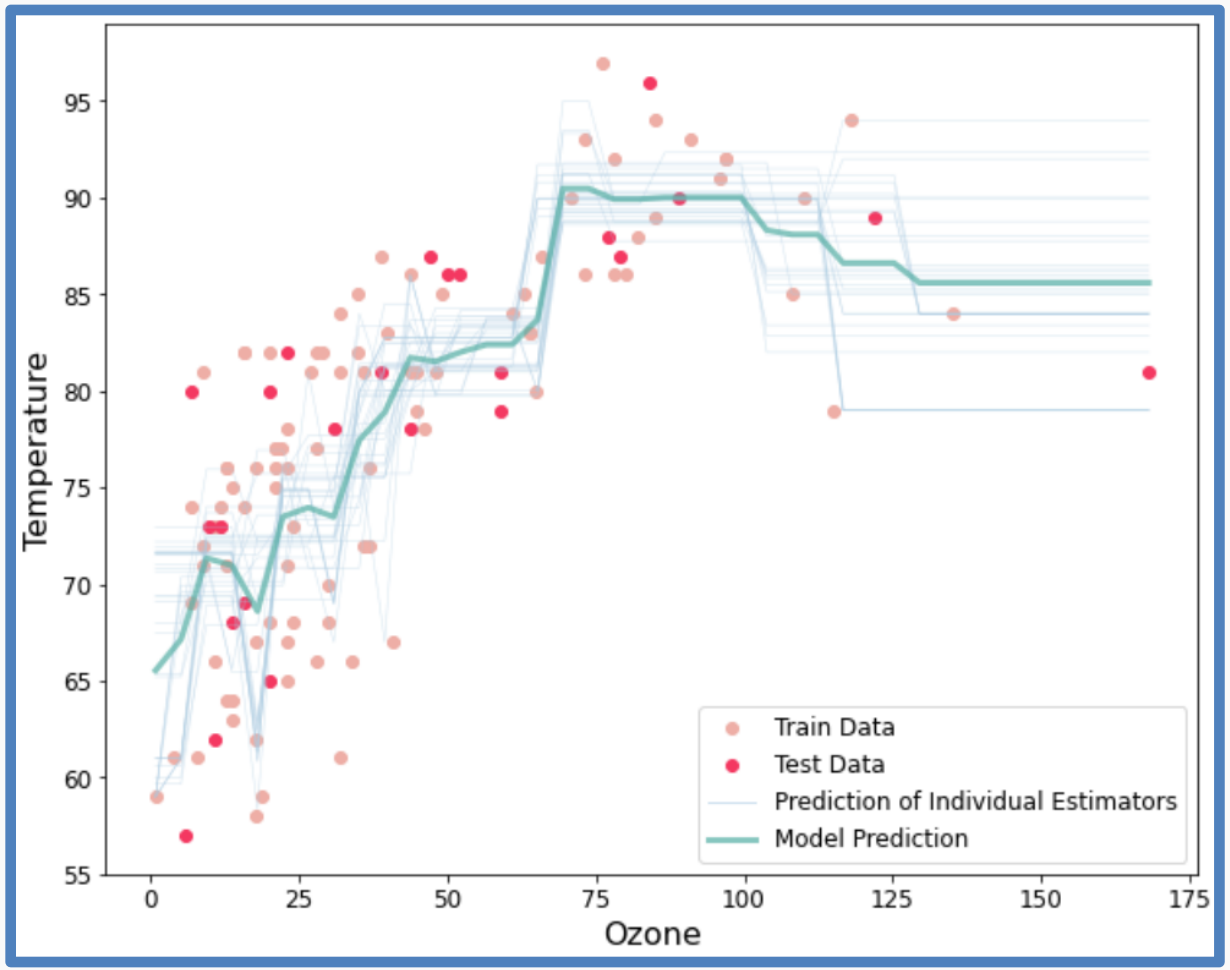


Combine them? 100 magic realisms

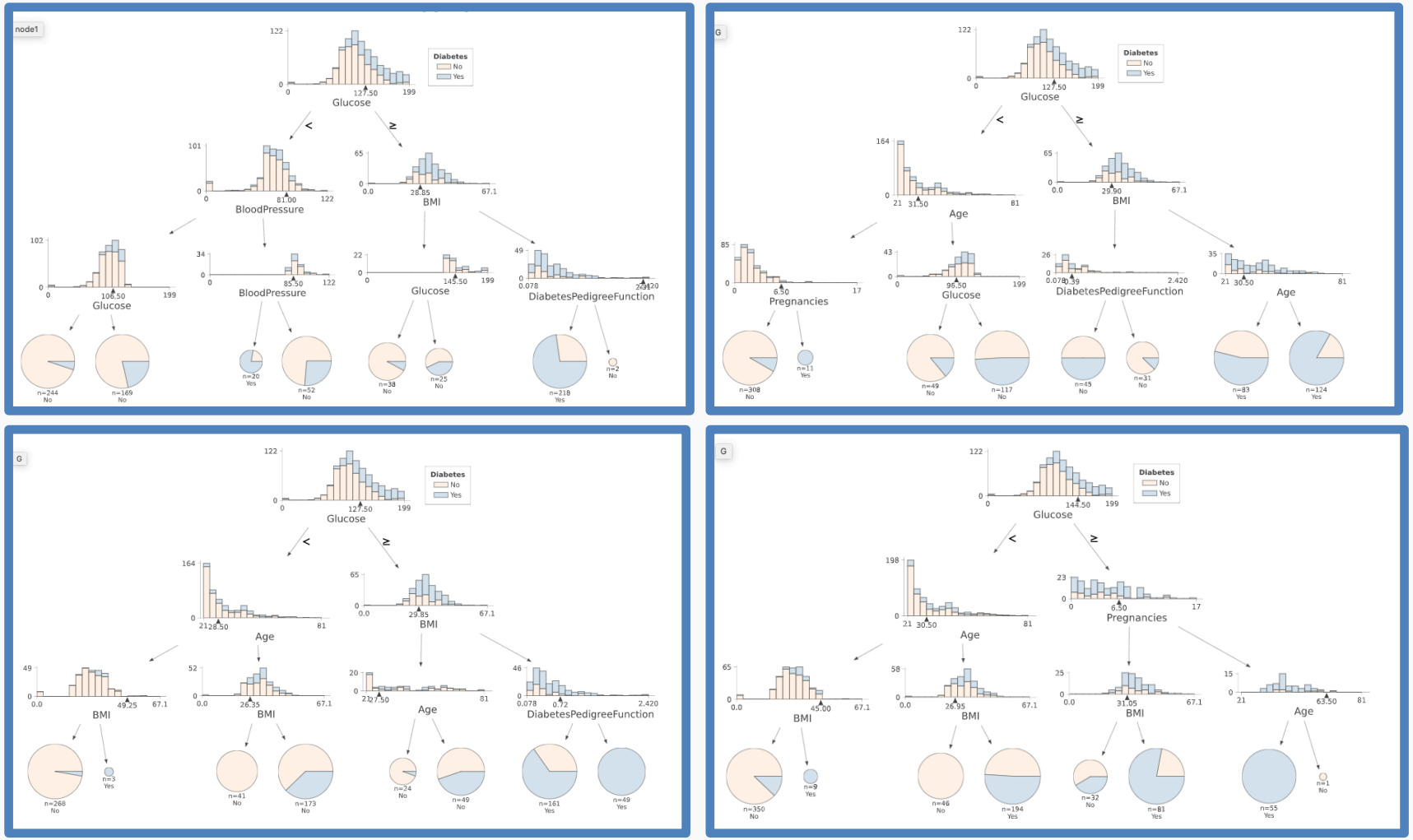


Bagging (regression)

The resulting tree is the average of all tree (estimators).



Bagging (classification)



For each bootstrap, we build a decision tree. The results is a combination (majority) of the predictions from all trees.

Bootstrap Aggregating

BENEFITS

- More expressive
- Helps prevent overfitting
- Decreases variance
  (less sensitive to different data)

ISSUES

- interpretability ("majority")

  solution: variable importance via the avg Gini/MSE

  for each feature

- can still underfit or overfit

  solution: validation via *out-of-bag error*

- Trees tend to be **highly correlated**

  (split the same at the beginning)

  solution: random forests

RANDOM FORESTS

Random Forests

Random Forest is a modified form of bagging that creates ensembles of <u>independent</u> decision trees.

To de-correlate the trees, we:

1. train each tree on a **separate bootstrap sample** of the full training set (same as in bagging)

2. for each tree, at each split, we ***randomly*** select a set of J' predictors from the full set of predictors.

From amongst the J' predictors, we select the optimal predictor and the optimal corresponding threshold for the split.

Random Forests

SPECIFY

- Number of trees (`n_estimators`)
- Number of predictors (`max_features`)

CONSIDERATIONS

- Be careful w/ the # of predictors. If you select a small %, you'll have an ensemble of weak models
- A lot of hyperparameters. Vary all of them together.

BOOSTING

Motivation for Boosting

Question: Could we address the shortcomings of single decision trees models in some other way?

For example, rather than performing variance reduction on complex trees, can we decrease the bias of simple trees - make them more expressive?

Can we learn from our mistakes?

A solution to this problem, making an expressive model from simple trees, is another class of ensemble methods called ***boosting***.

Gradient Boosting

The key intuition behind boosting is that one can take an ensemble of simple models $\{T_h\}_{h \in H}$ and additively combine them into a single, more complex model.

Each model T_h might be a poor fit for the data, but a linear combination of the ensemble:

$$\boxed{T = \sum_h \lambda_h T_H}$$

can be expressive/flexible.

Gradient Boosting: the algorithm

Gradient boosting is a method for iteratively building a complex regression model T by adding simple models.

Each new simple model added to the ensemble compensates for the weaknesses of the current ensemble.

Gradient Boosting: the algorithm

1. Fit a simple model $T^{(0)}$ on the training data

$$\{(x_1, y_1), \dots, (x_N, y_N)\}$$

   Set $T \leftarrow T^{(0)}$.

   Compute the residuals $\{r_1, \dots, r_N\}$ for T.

2. Fit a simple model, $T^{(1)}$, to the current ***residuals***, i.e. train using

$$\{(x_1, r_1), \dots, (x_N, r_N)\}$$

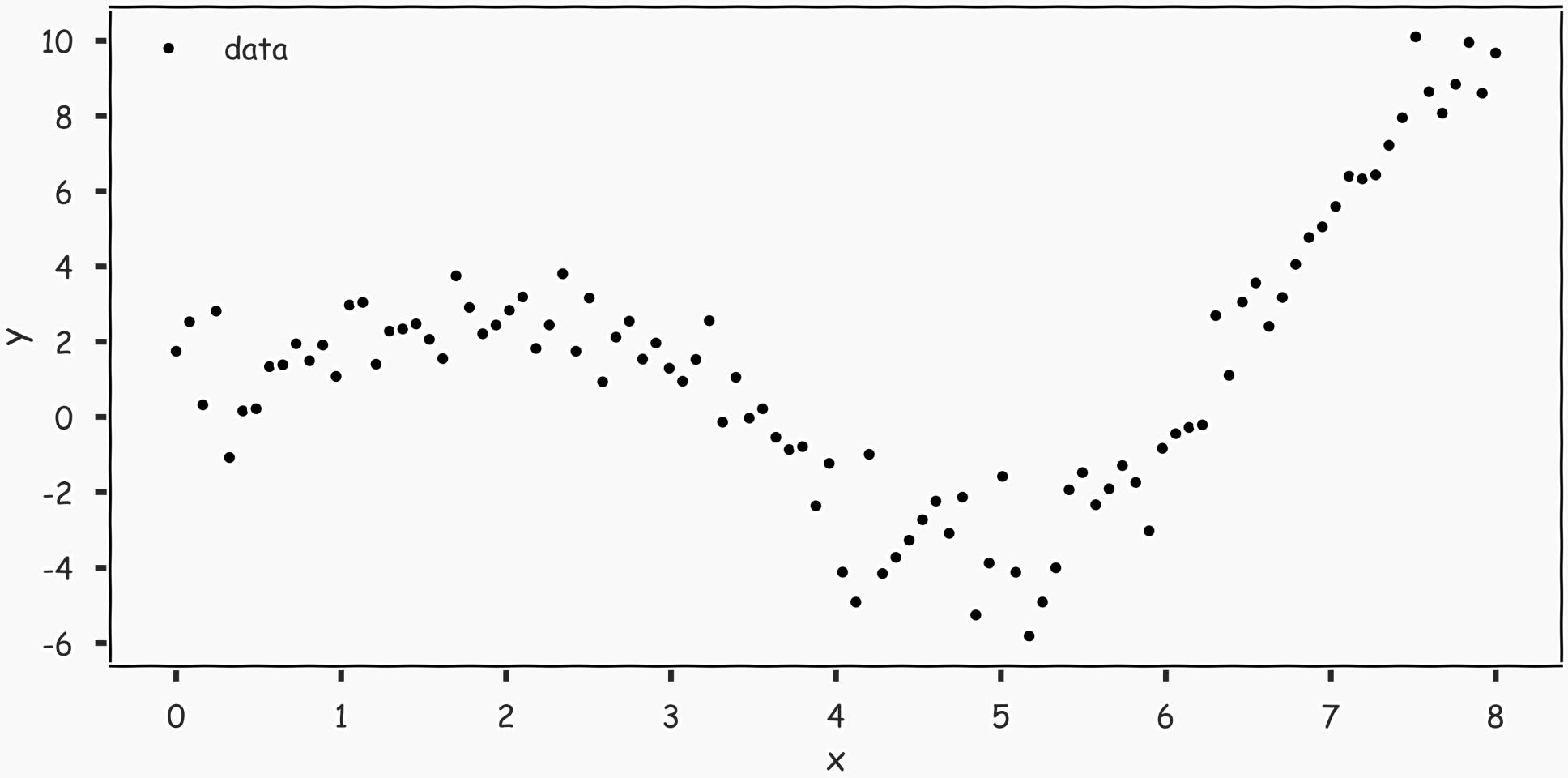
3. Set $T \leftarrow T + \lambda T^{(1)}$

4. Compute residuals, set $r_n \leftarrow r_n - \lambda T^i(x_n),\ n = 1, \dots, N$

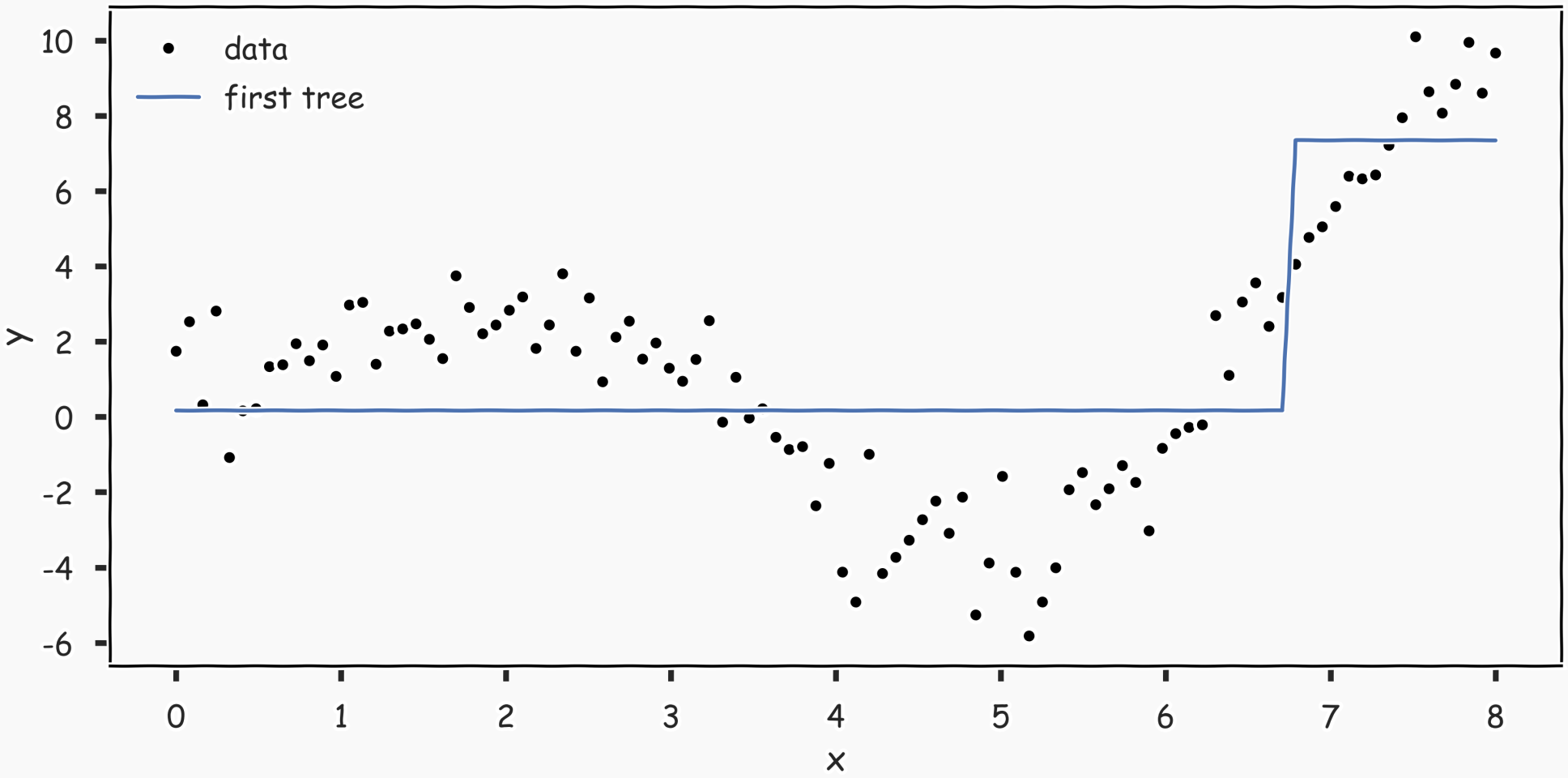
5. Repeat steps 2-4 until ***stopping*** condition met.

   where λ is a constant called the ***learning rate***.

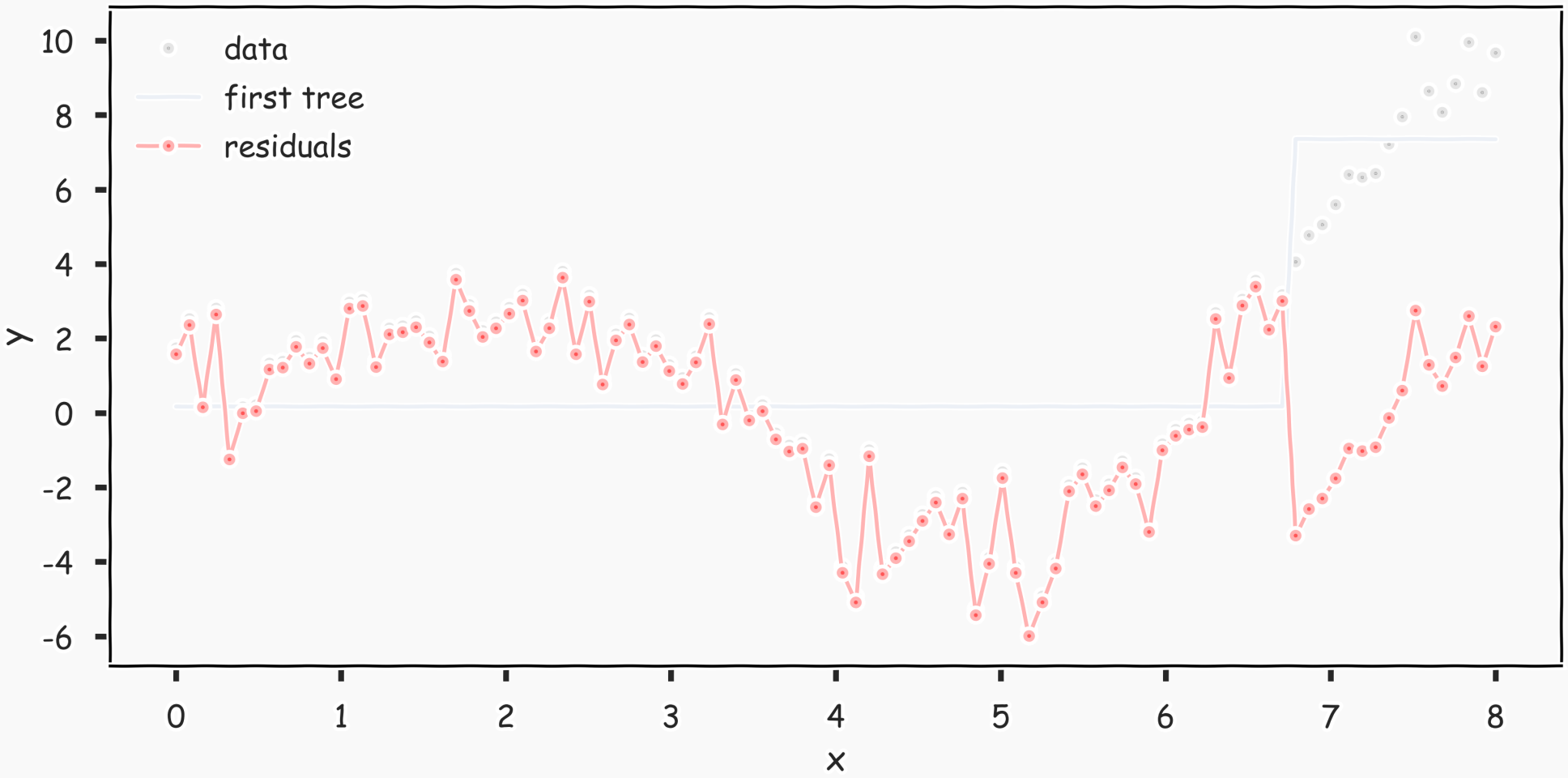
Gradient Boosting: illustration



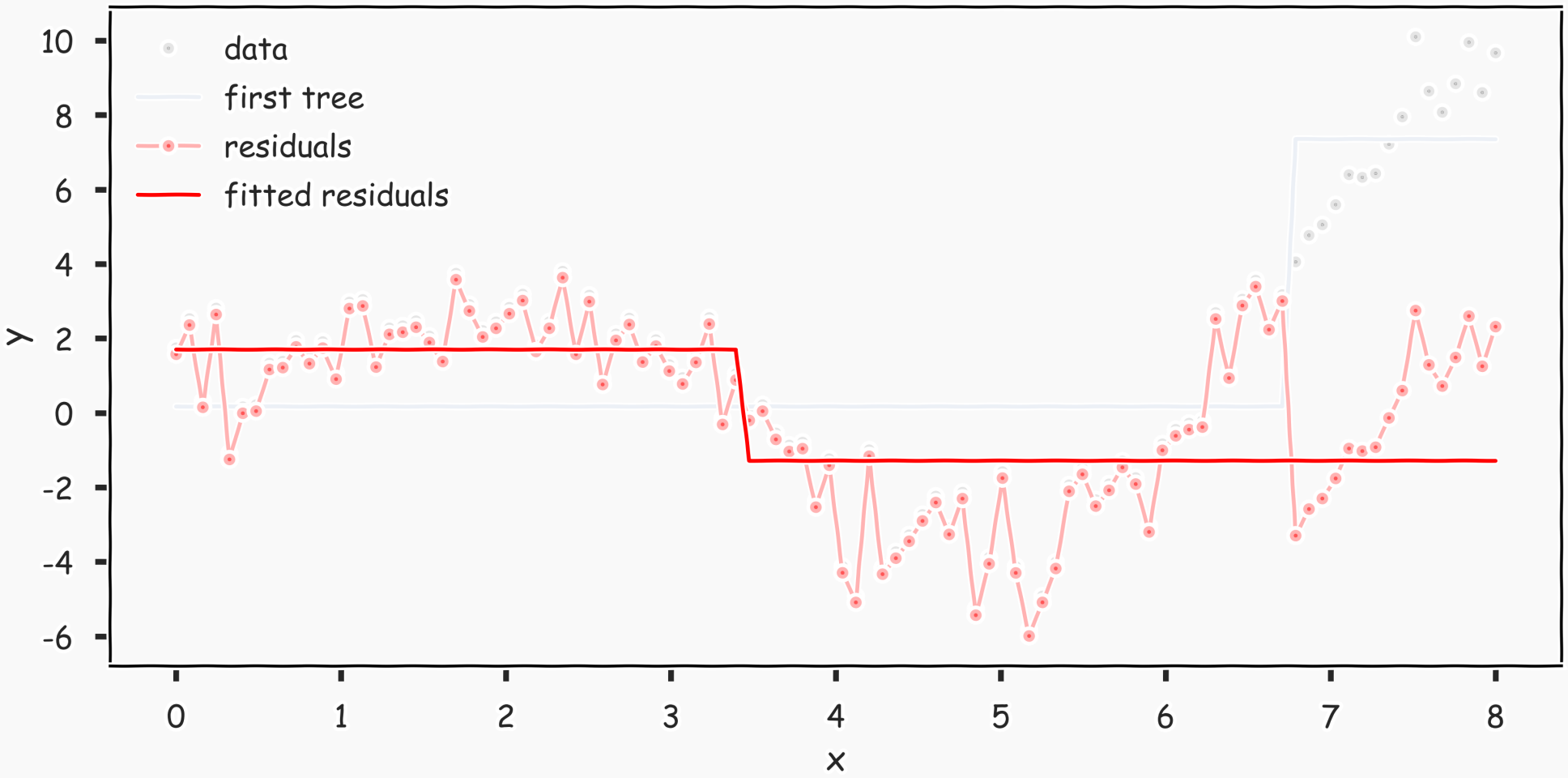
Gradient Boosting: illustration



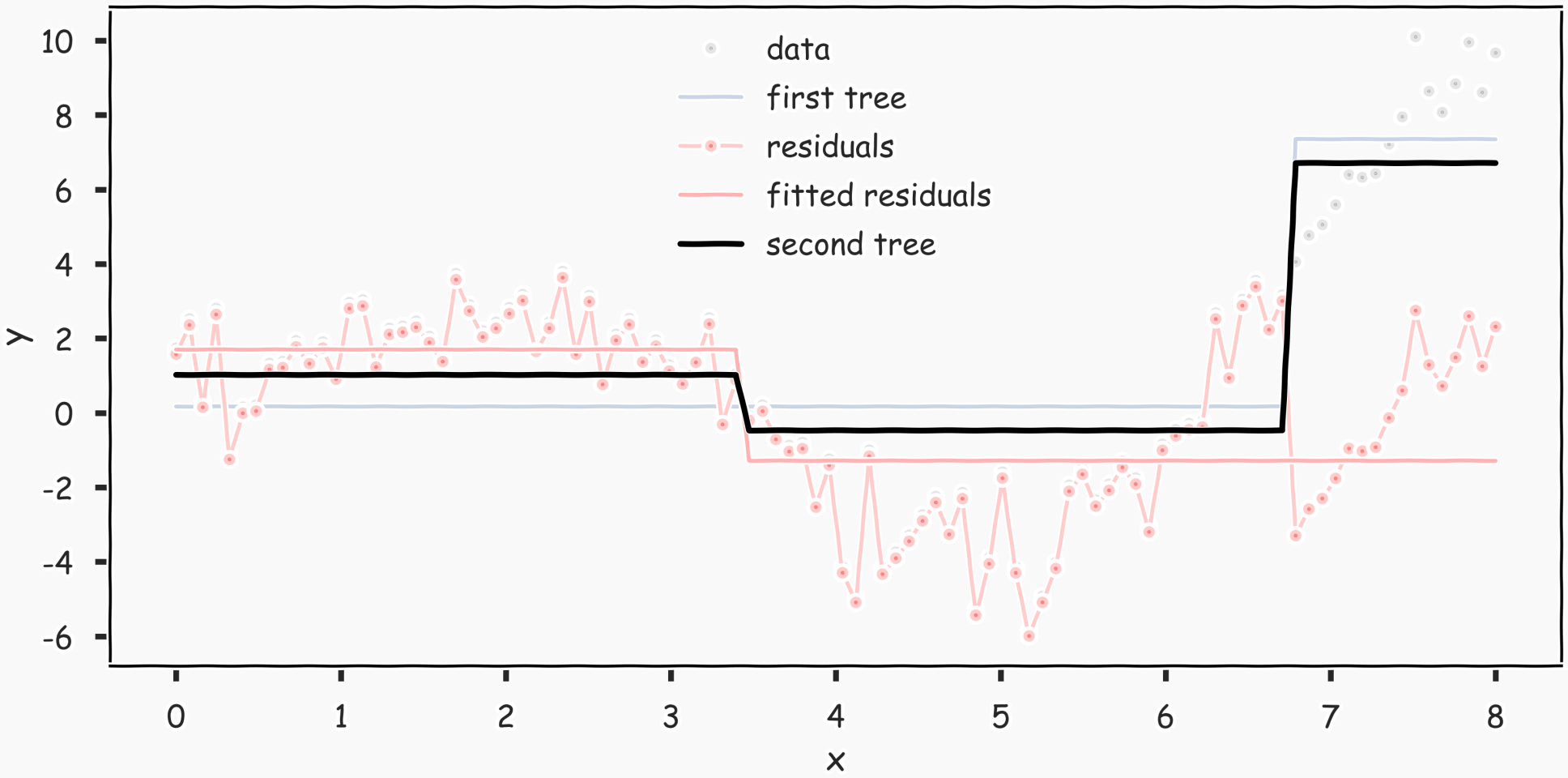
Gradient Boosting: illustration



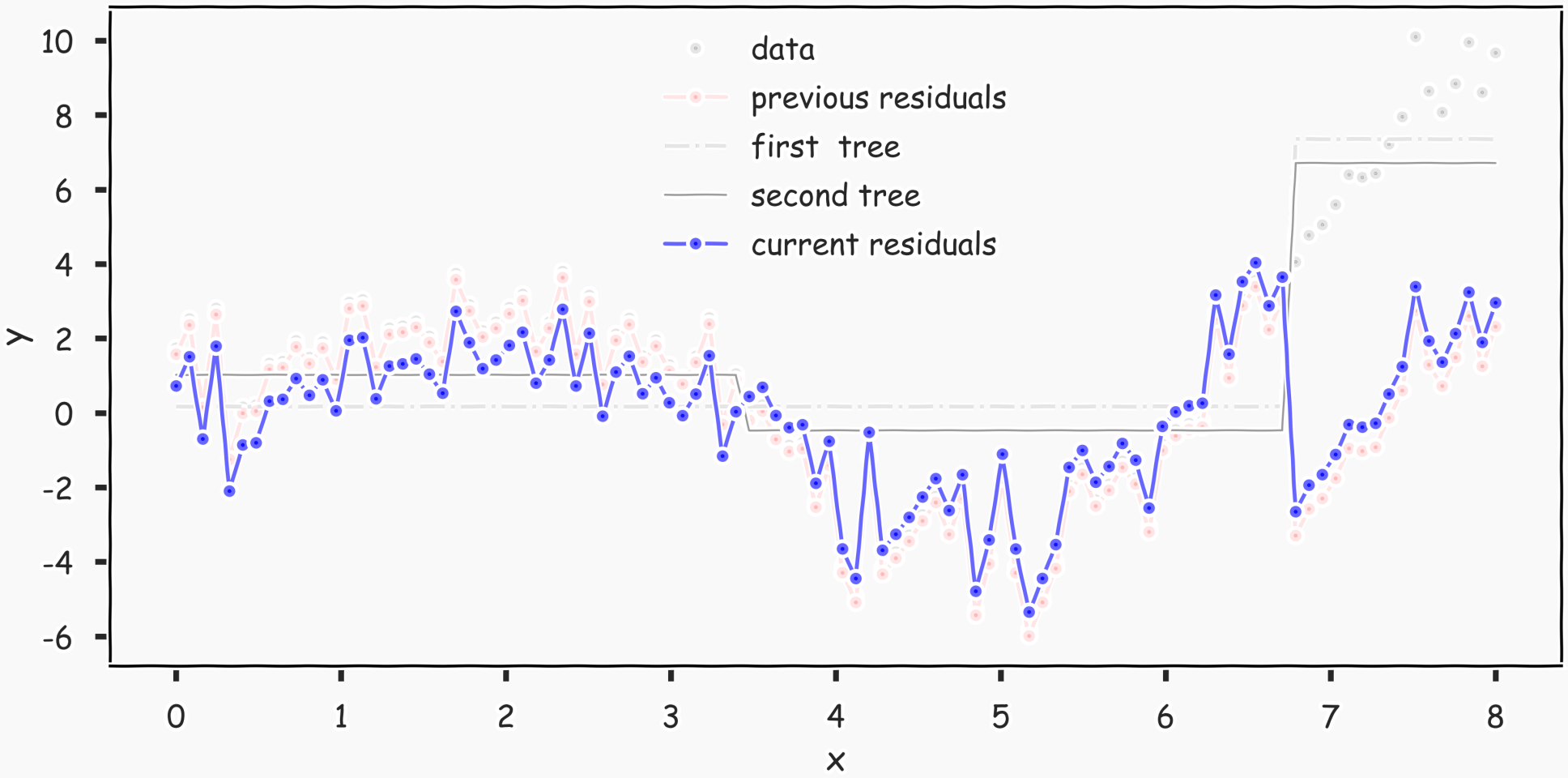
Gradient Boosting: illustration



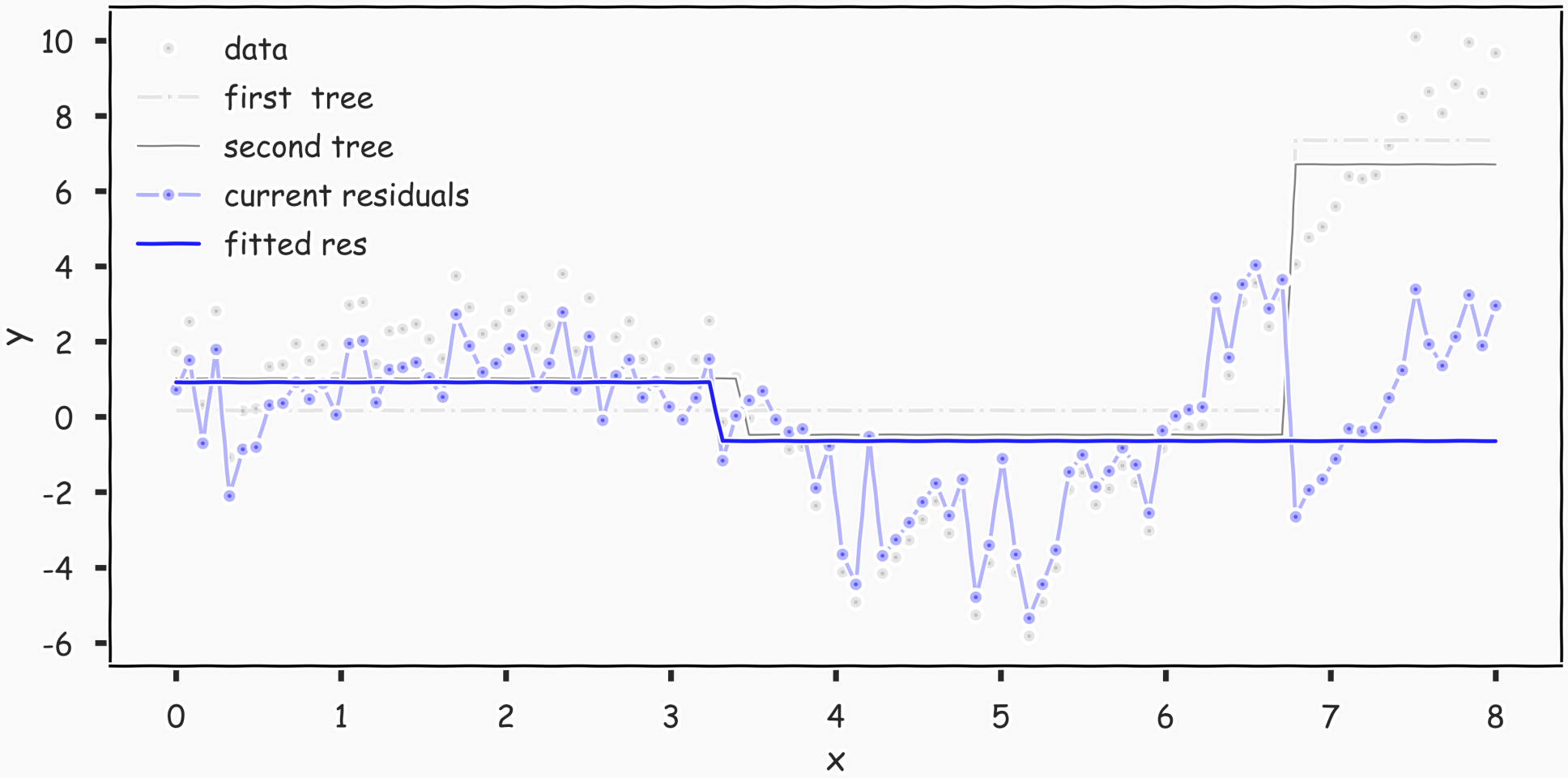
Gradient Boosting: illustration



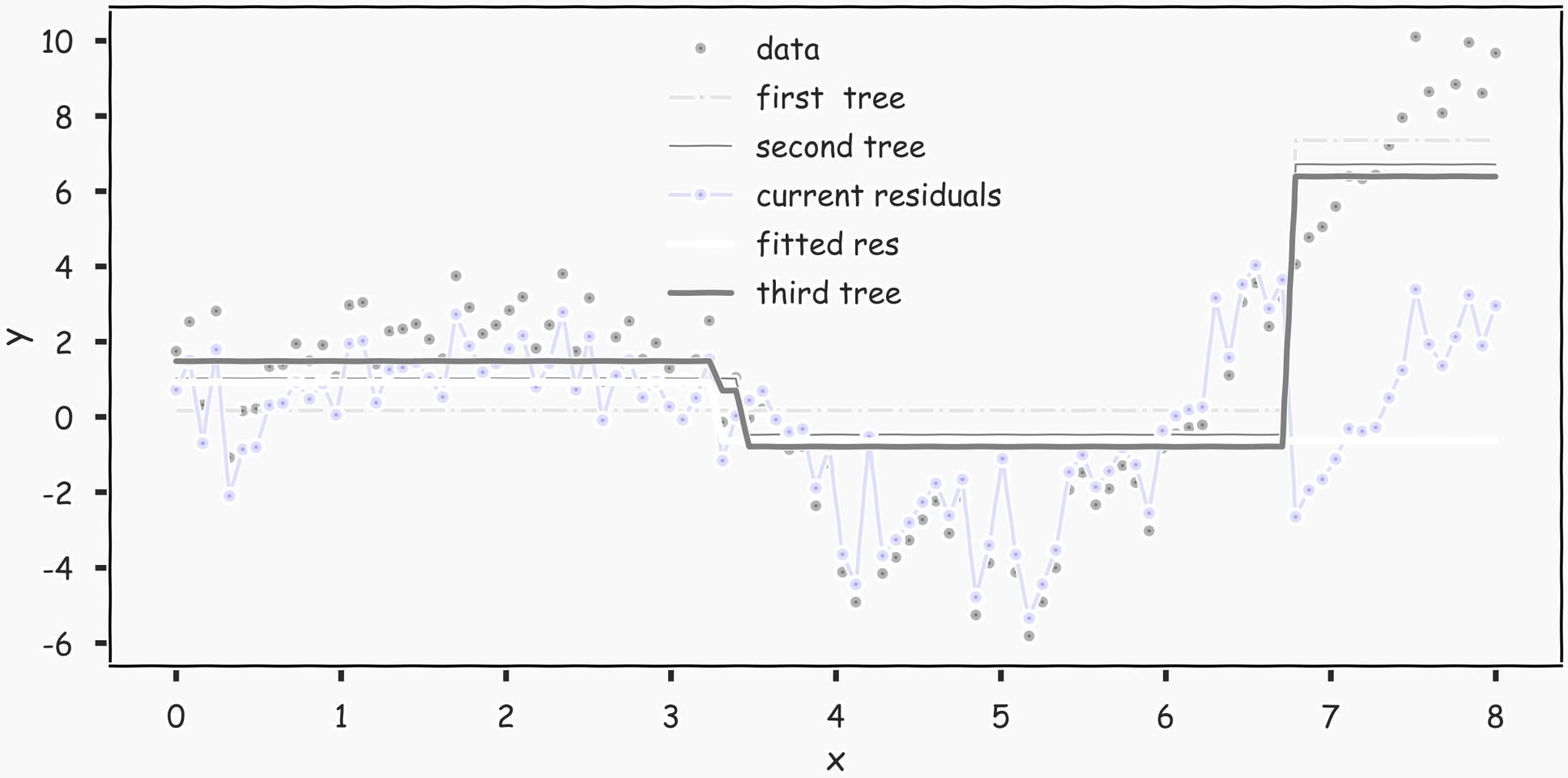
Gradient Boosting: illustration

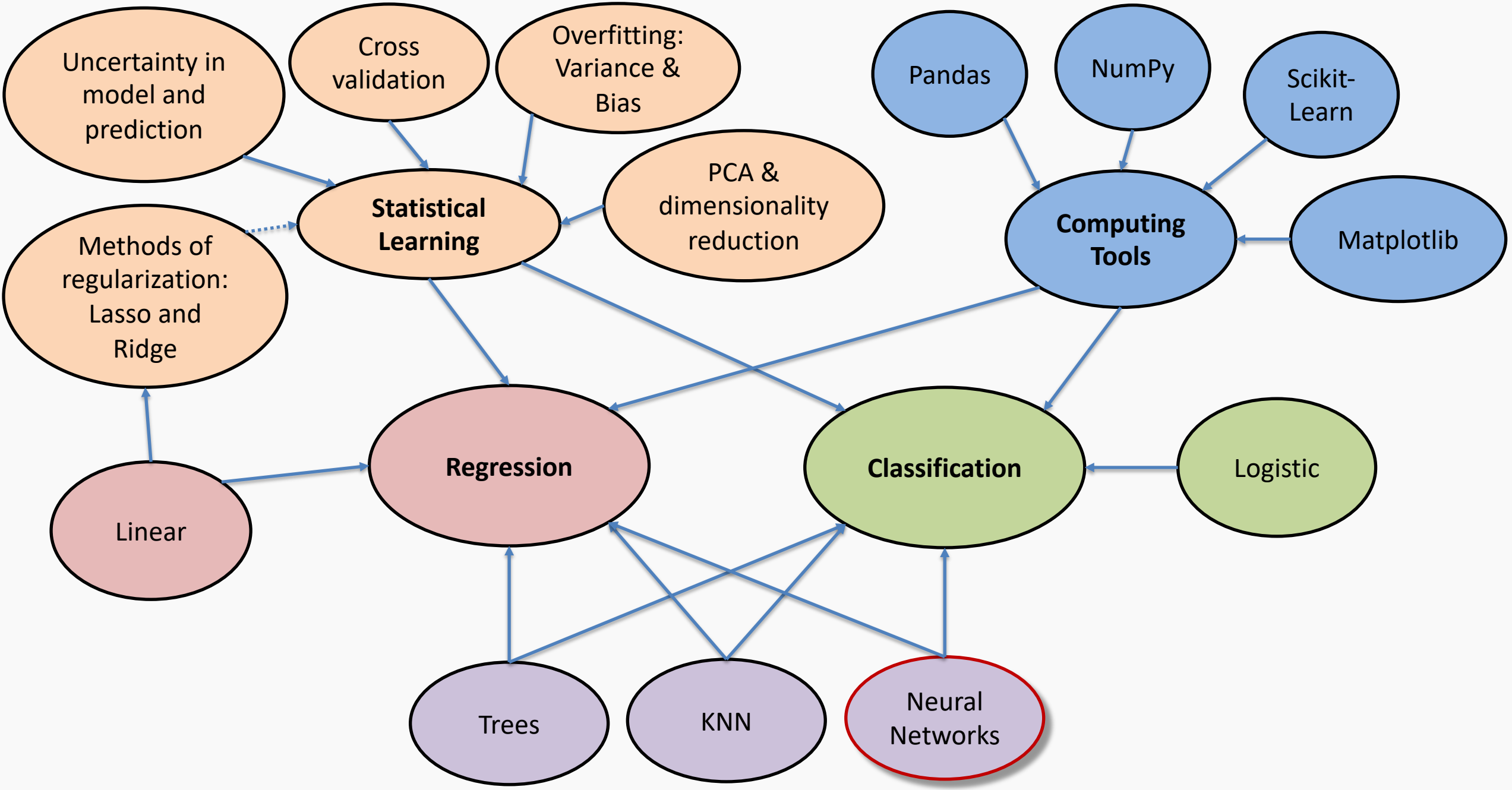


Gradient Boosting: illustration

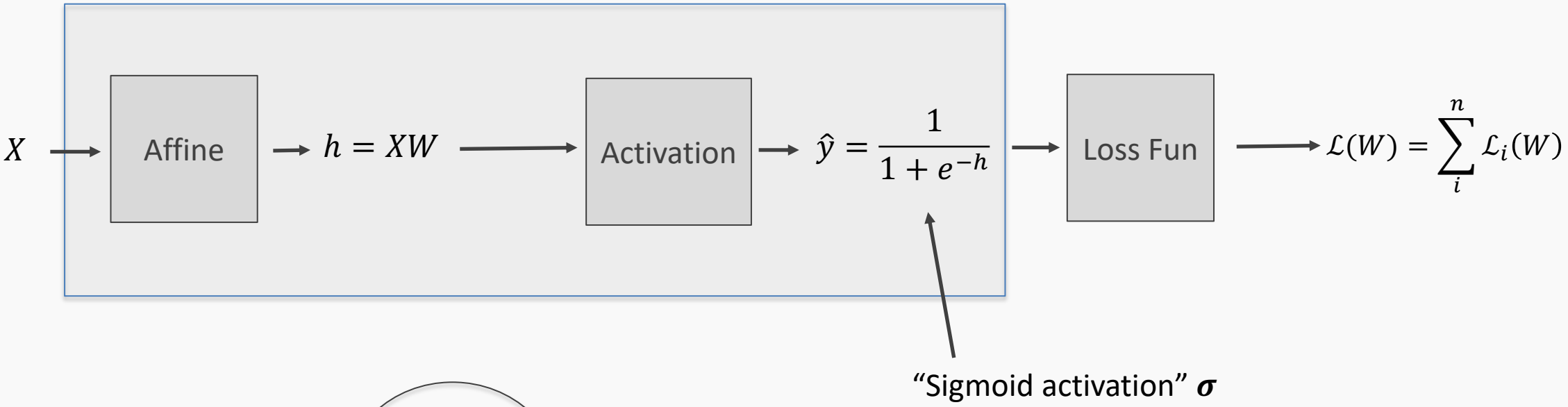


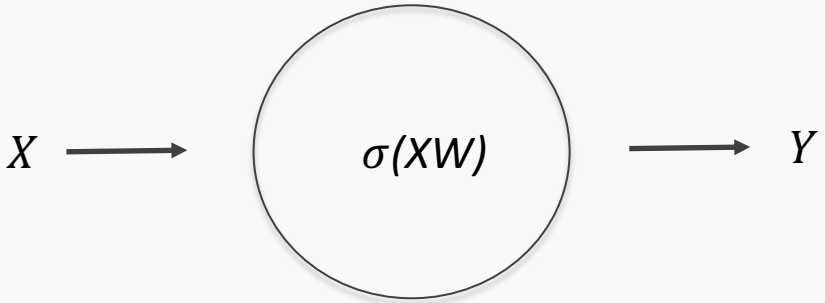
Gradient Boosting: illustration




Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks

Build our first ANN

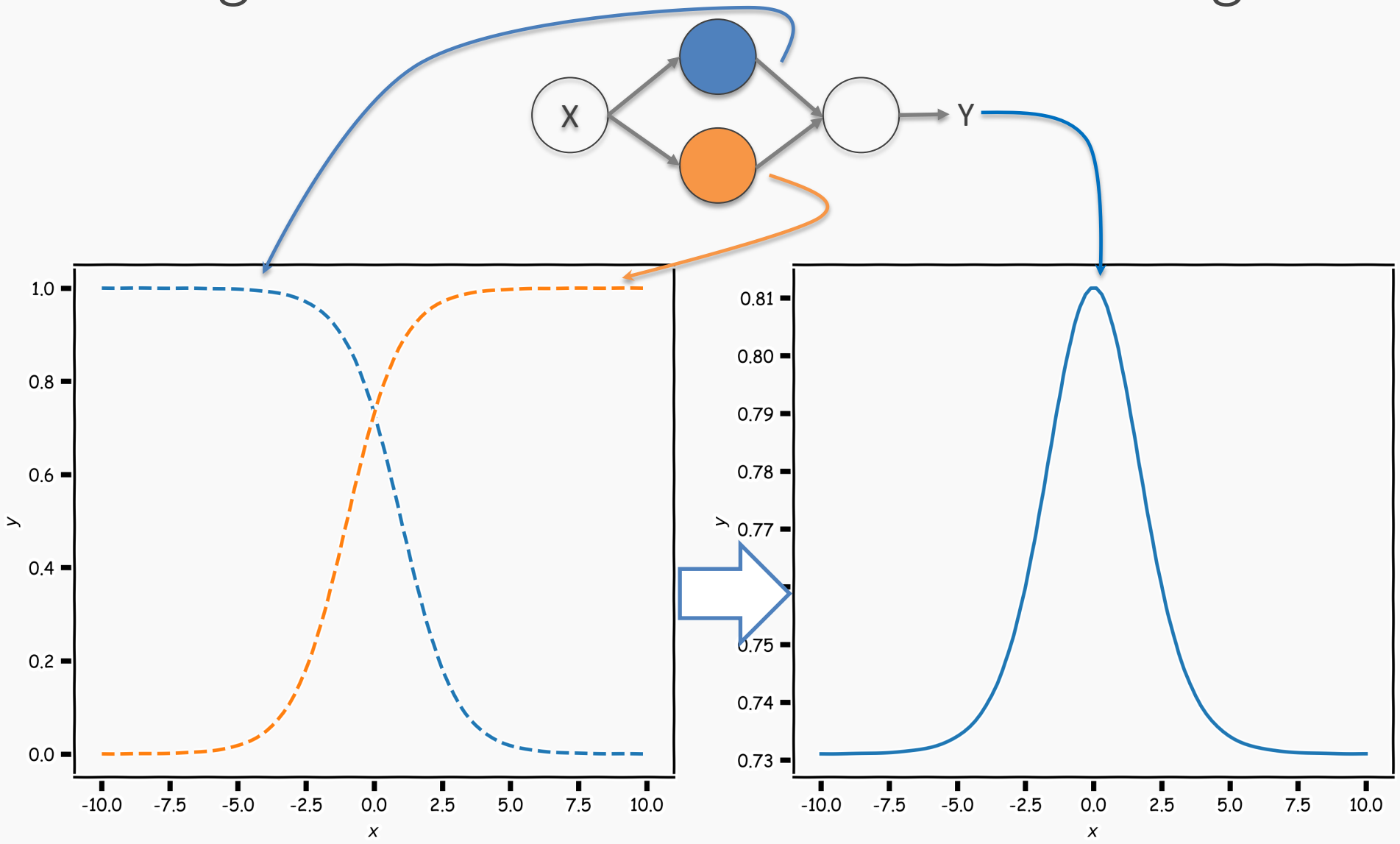




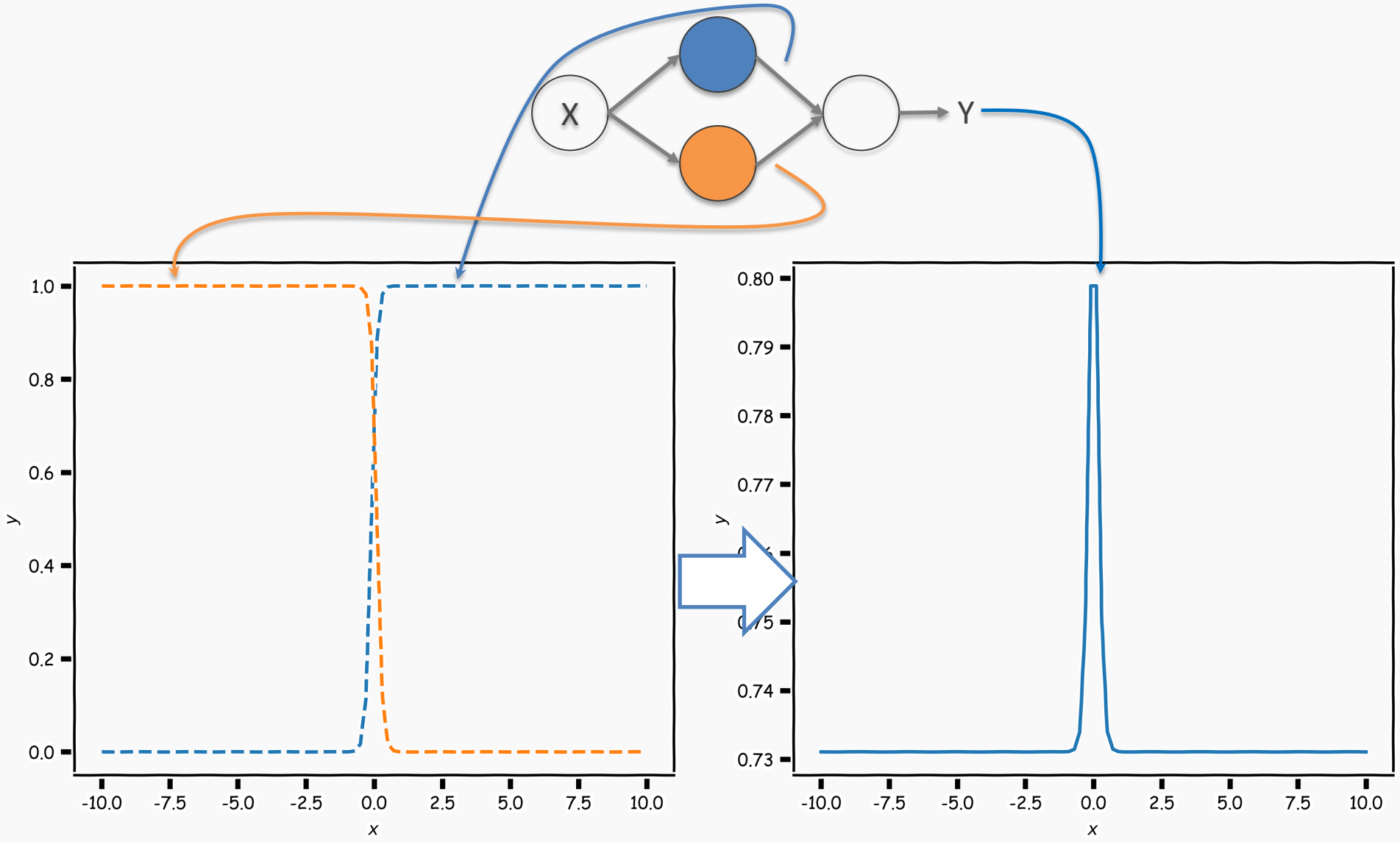
Single Neuron Network

Very similar to Perceptron

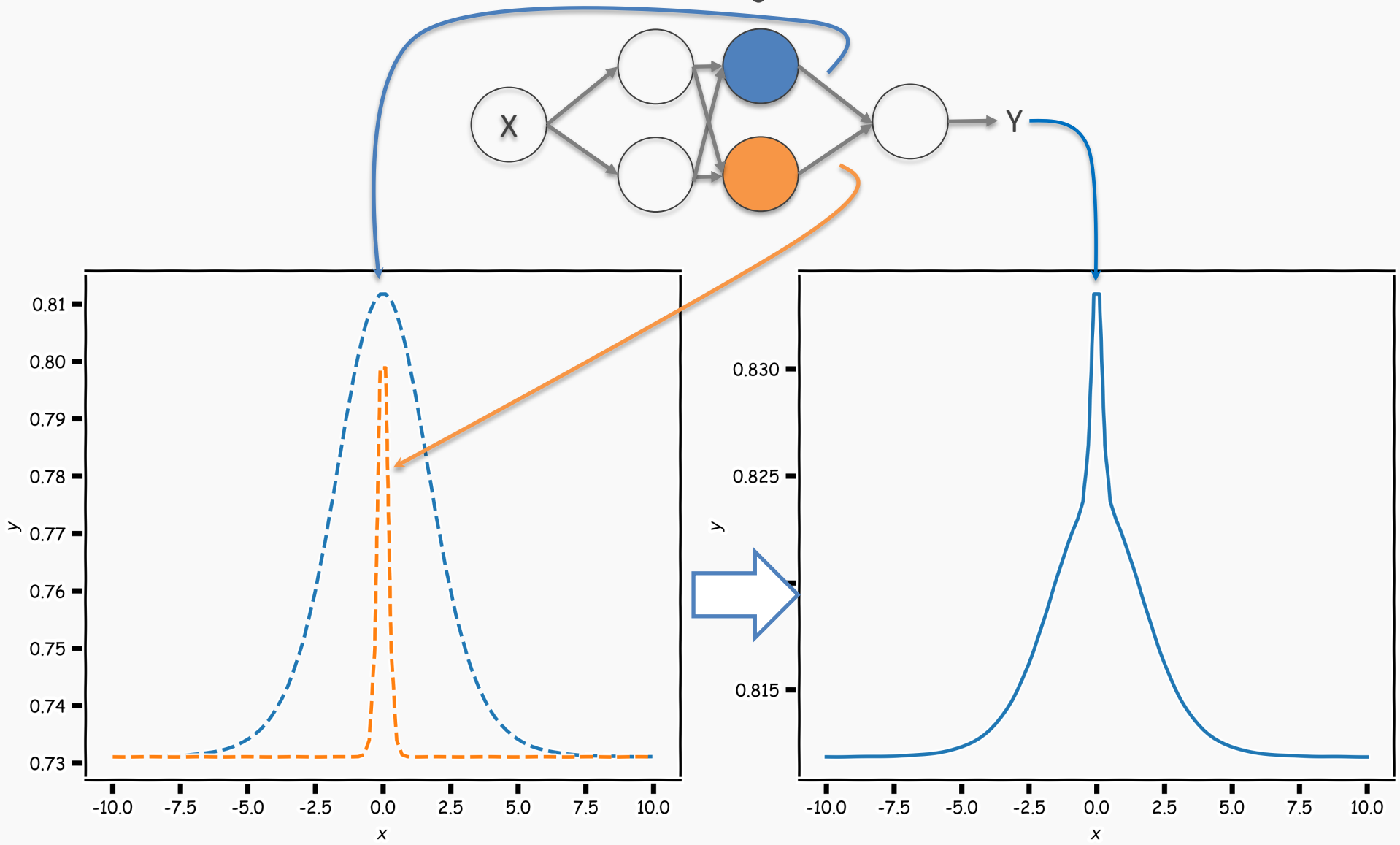
Combining neurons allows us to model interesting functions



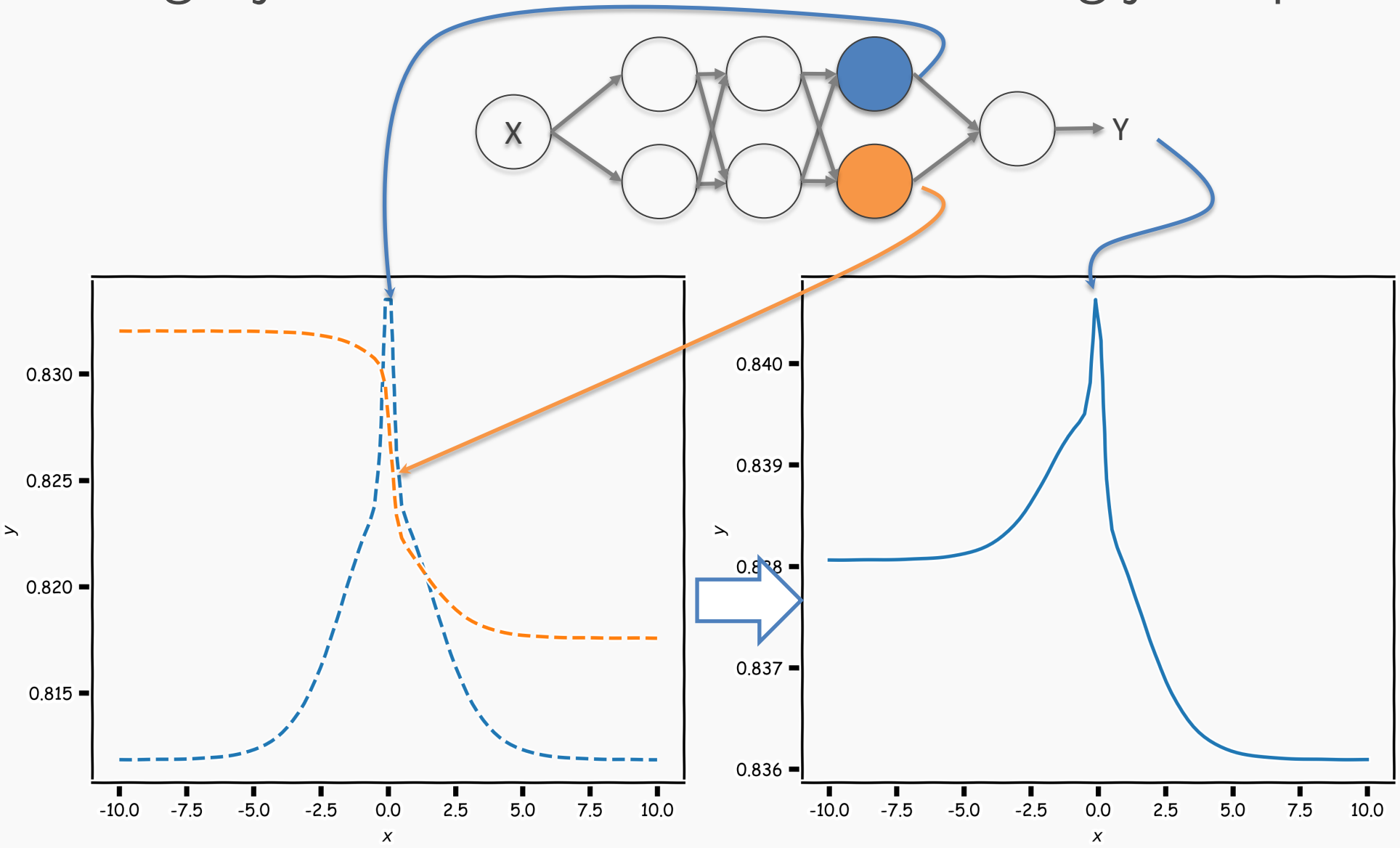
Different weights change the shape and position



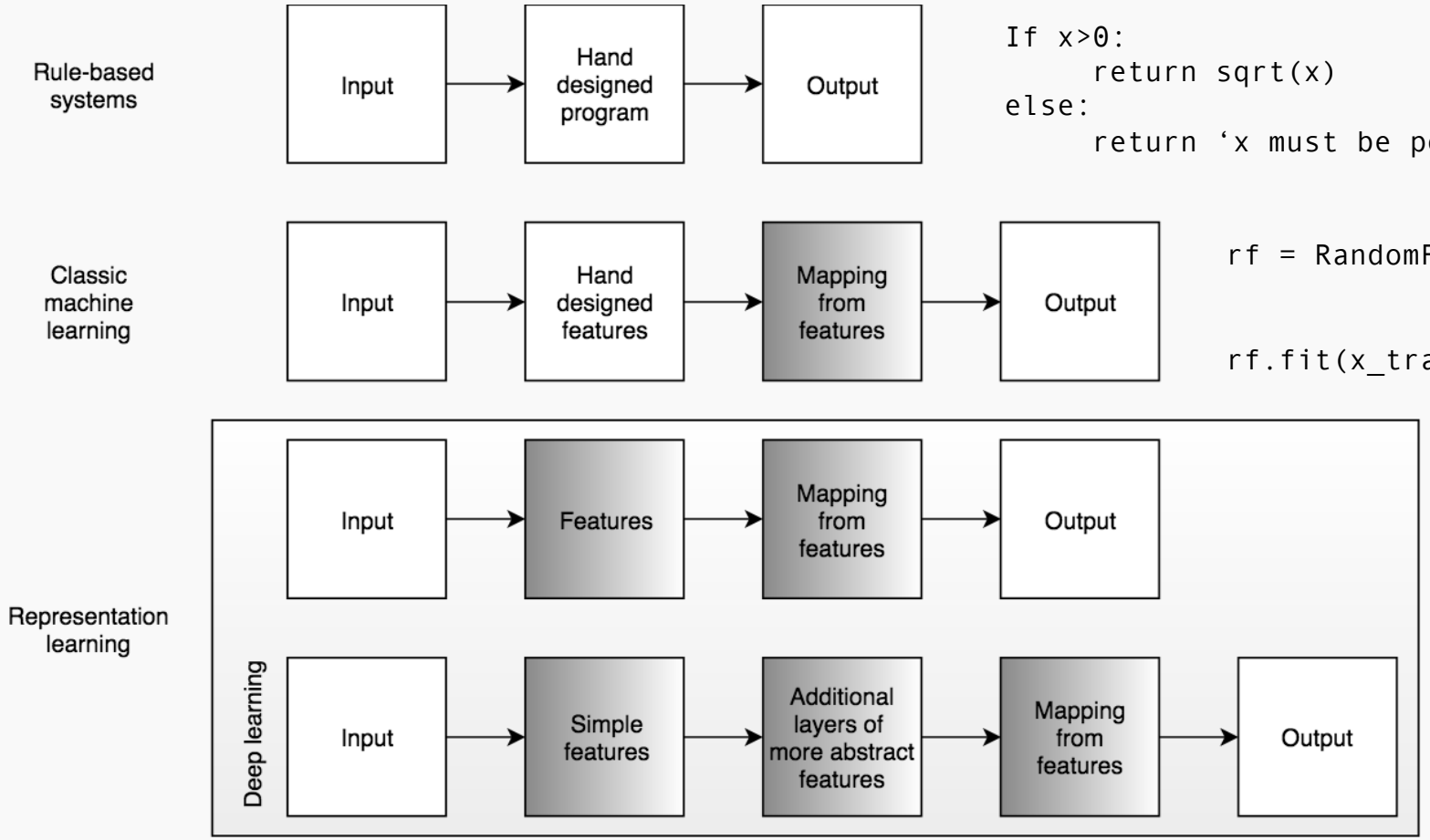
Neural networks can model *any* reasonable function



Adding layers allows us to model increasingly complex functions



Learning Multiple Components



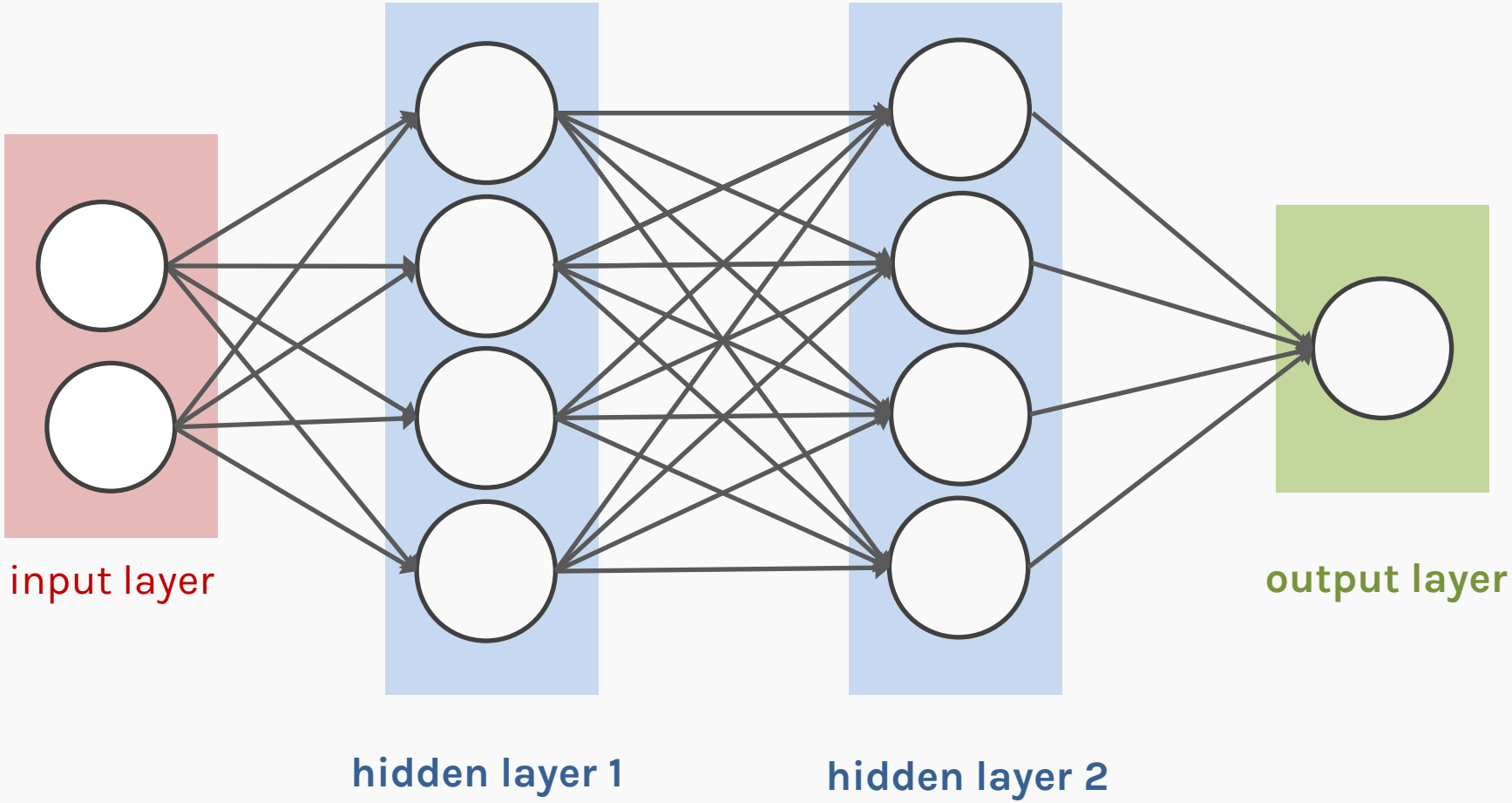

```
If x>0:
     return sqrt(x)
else:
     return ‘x must be positive’
```

```
rf = RandomForestRegressor(
                n_estimators = 1000,
                random_state = 42
rf.fit(x_train, y_train);
```

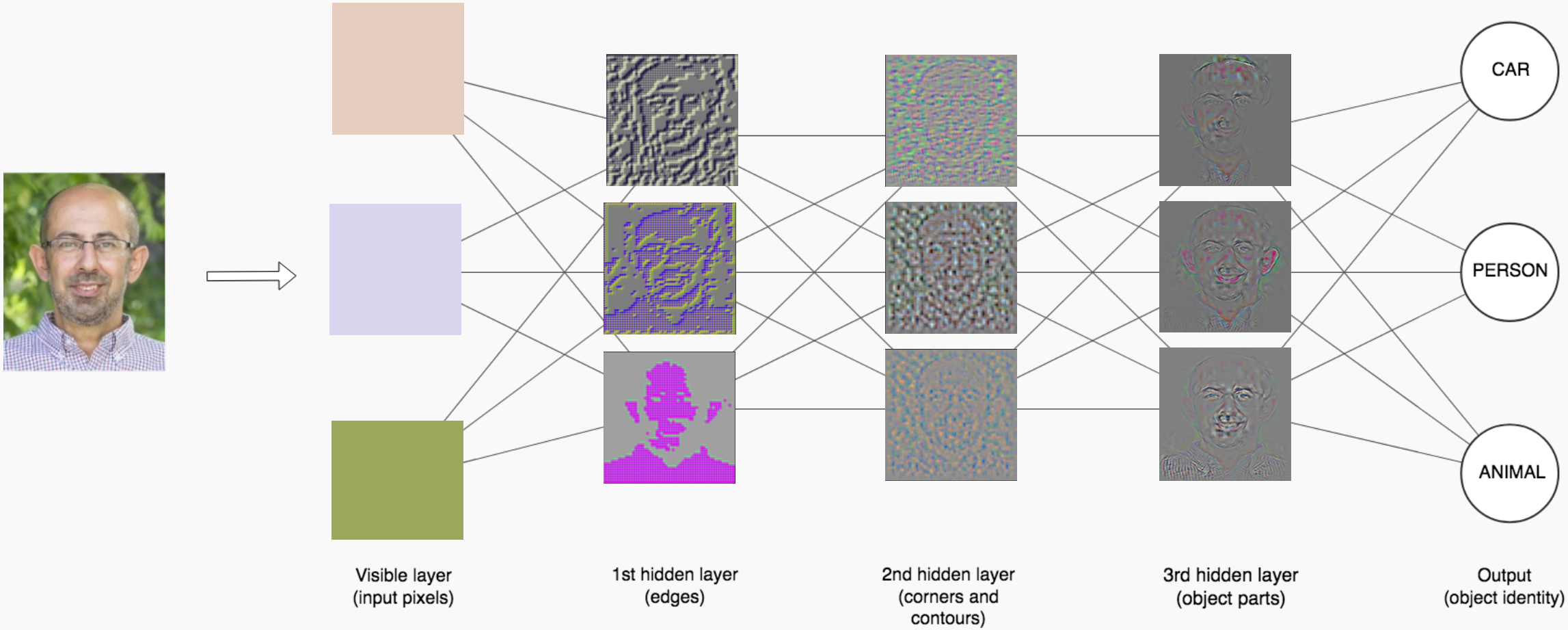
```
model =
tf.keras.Sequential()
model.add(tf.keras.layers.
Flatten(input_shape=(28,
28)))
model.add(tf.keras.layers.
Dense(154,
activation='relu'))
```

Next: Artificial general intelligence ??

Anatomy of artificial neural network (ANN)

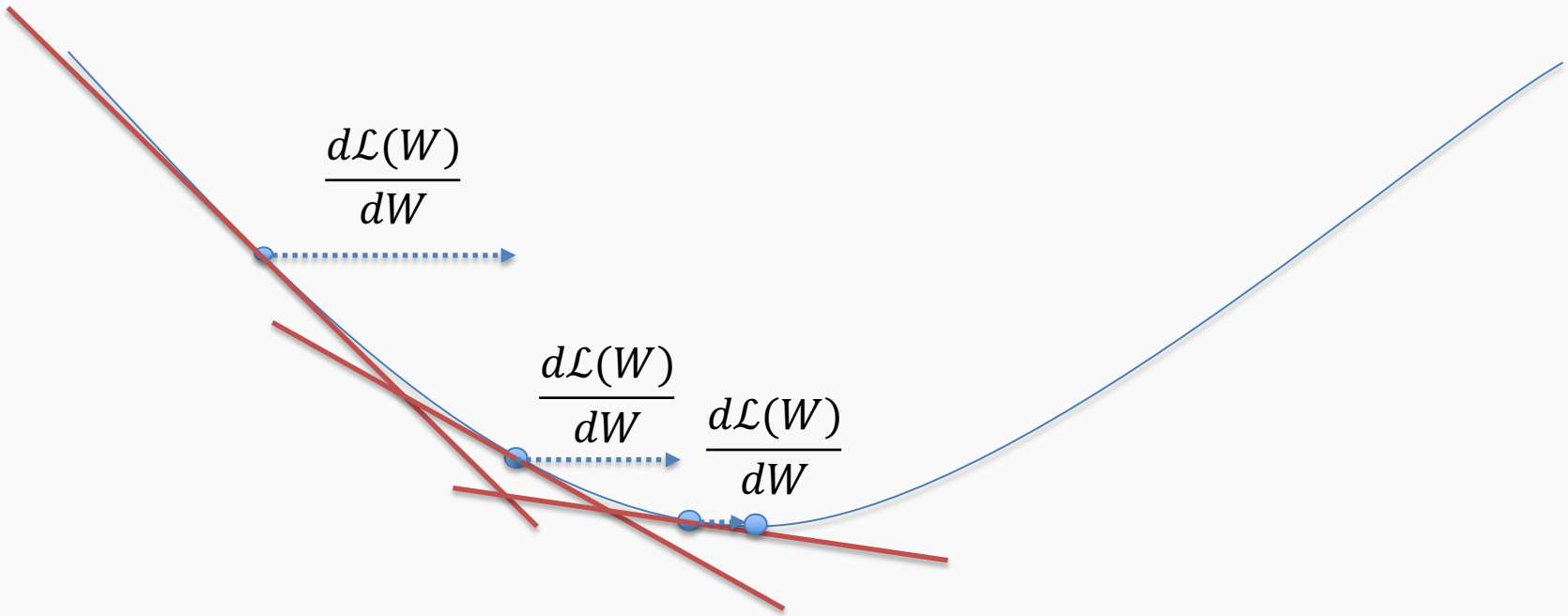


Depth = Repeated Compositions



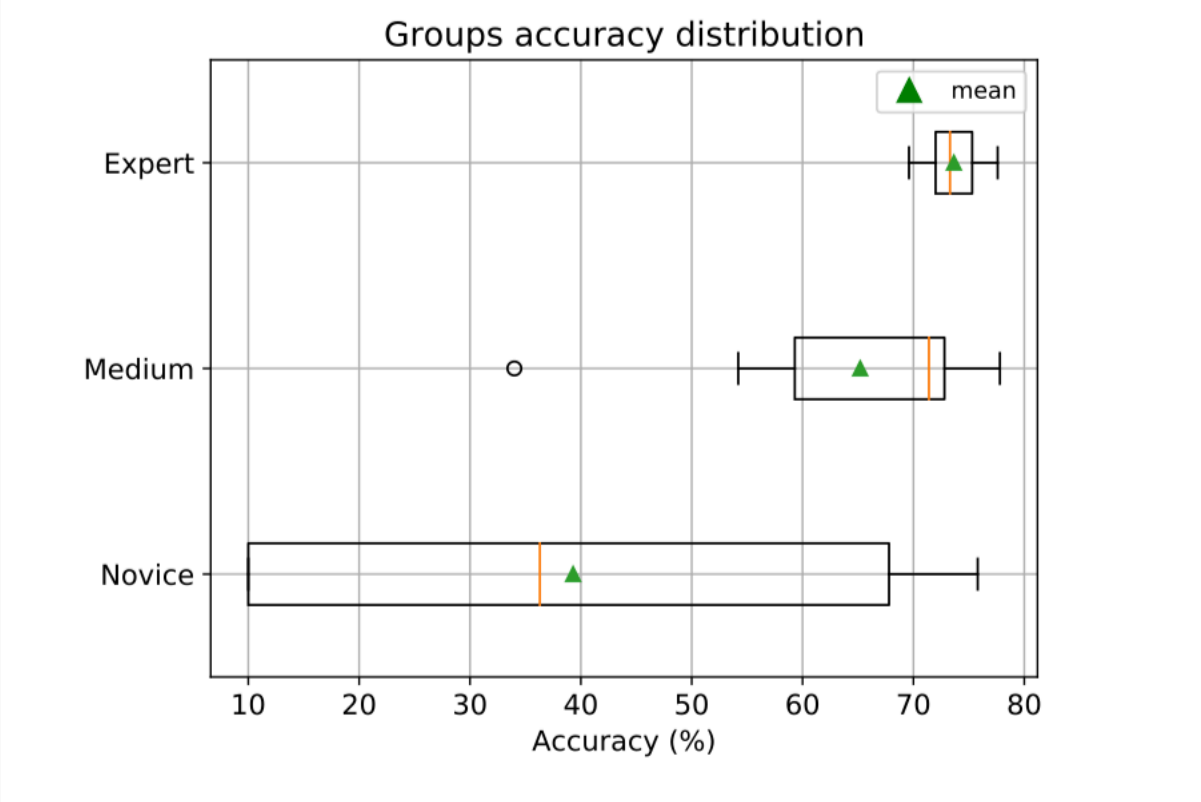
Gradient Descent (cont.)

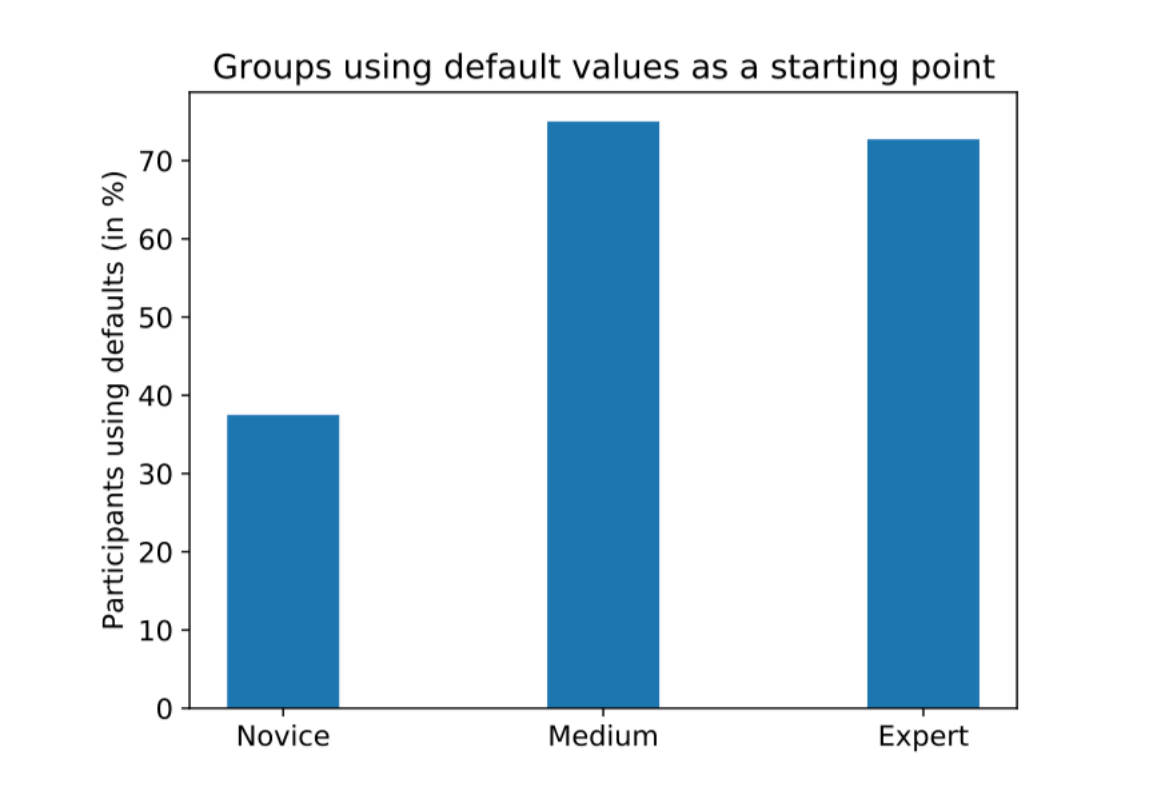
If the step is proportional to the slope then you avoid overshooting the minimum. How?



Hyperparameter tuning

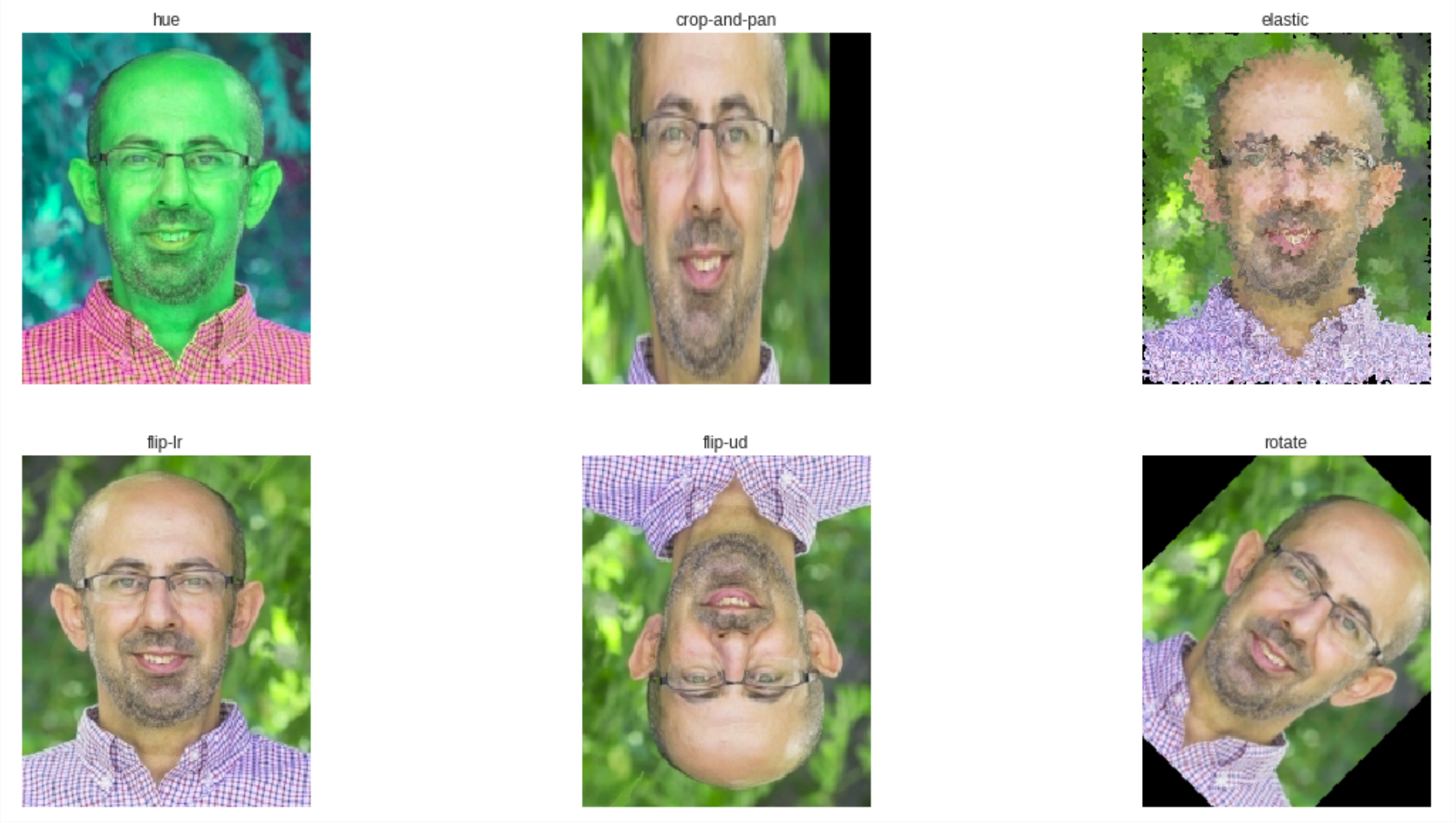
Random search, grid search, developer tools such as `weights and biases', Bayesian optimization, **expertise**





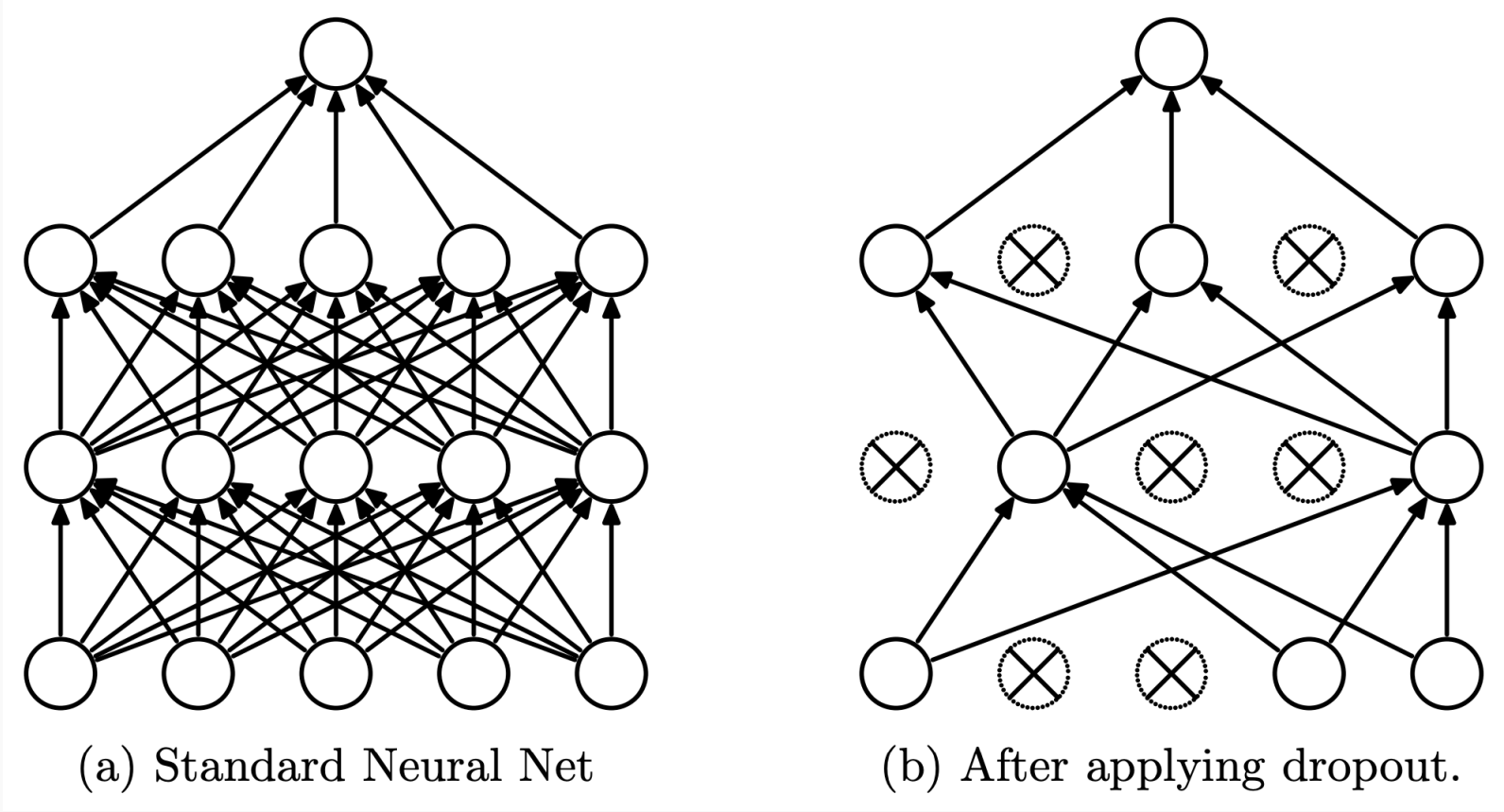
Anand, Kanav, Wang, Ziqi, Loog, Marco, & Van Gemert, Jan. (2020). Black Magic in Deep Learning: How Human Skill Impacts Network Training

Data Augmentation



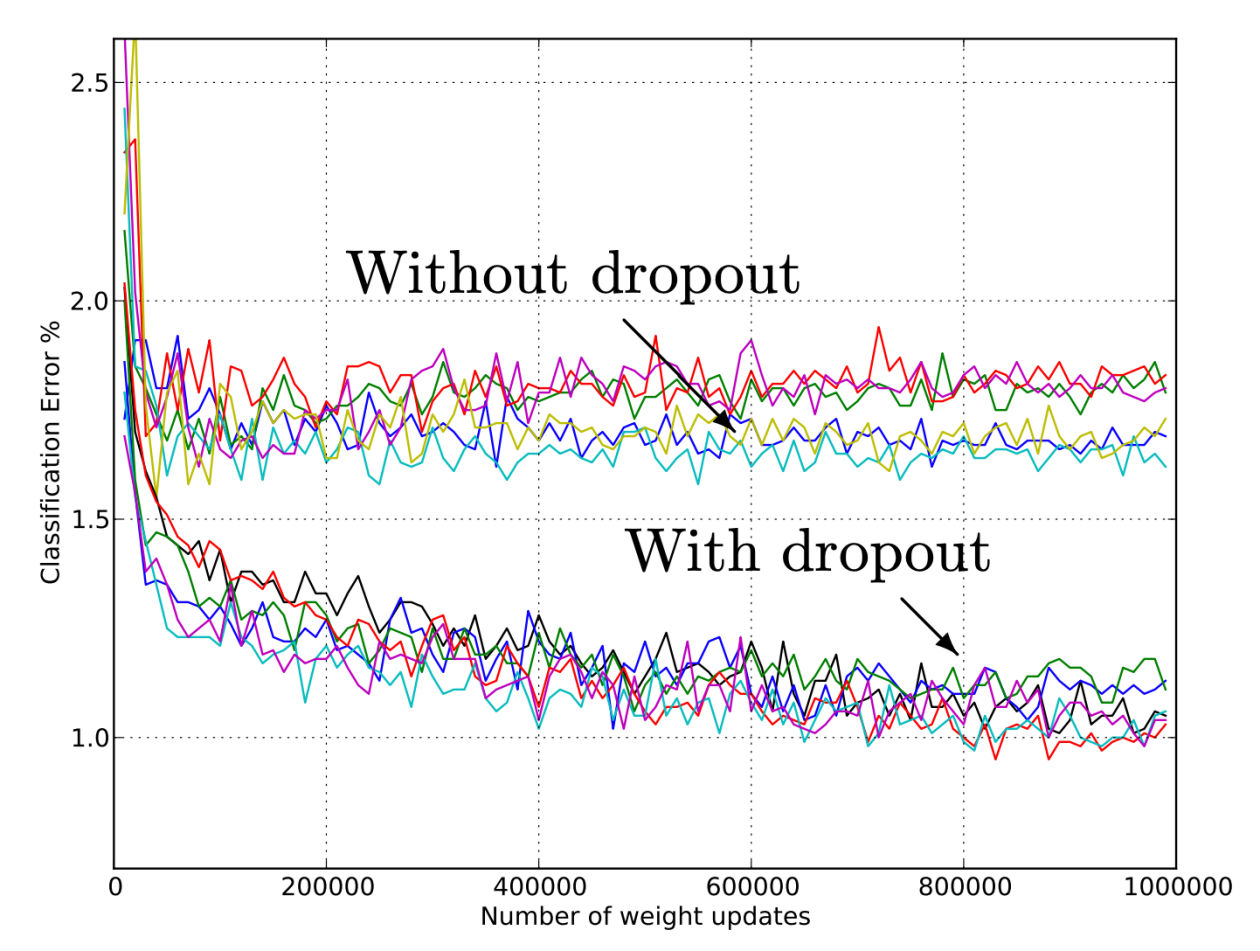
Dropout

- Randomly set some neurons and their connections to zero (i.e. “dropped”)
- Prevent overfitting by reducing co-adaptation of neurons
- Like training many random sub-networks



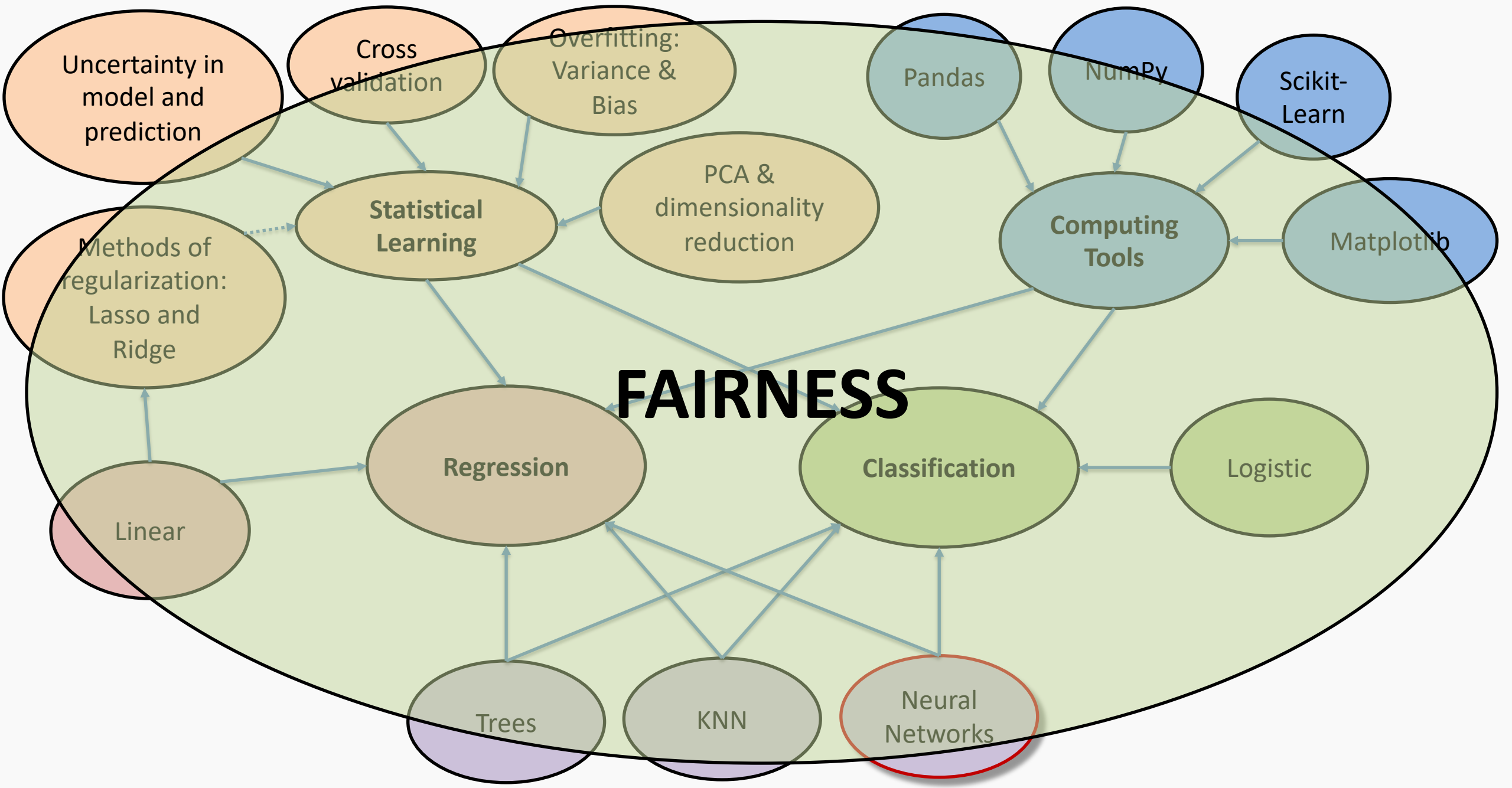
Dropout

- Widely used and highly effective
- Proposed as an alternative to ensembling, which is too expensive for neural nets



Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf


Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
FAIRNESS
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks

The Tyranny of Algorithmic Bias, and How to End It

A Work In Progress

- **Matthew Finney – Harvard University**
- CS109a, Fall 2020
- *All opinions are my own*

AI in the 2010s

THE WALL STREET JOURNAL.

Amazon Wants to Ship Your Package Before You Buy It

By Greg Bensinger
Jan. 17, 2014 3:12 pm ET

SAVE SHARE TEXT

BLOOMBERG

Amazon.com knows you so well it wants to ship your next package before you order it.

The Seattle retailer in December gained a patent for what it calls "anticipatory shipping," a method to start delivering packages even before customers click "buy."

AI in the 2010s

THE WALL STREET JOURNAL.

Amazon Wants to Ship Your Package Before You Buy It

By Greg Bensinger
Jan. 17, 2014 3:12 pm ET

SAVE SHARE

BLOOMBERG

ROAD/SHOW BY CNET

CAR FINDER REVIEWS BEST NEWS PRICES MORE

Tesla's full self-driving Autopilot beta coming in 'a month or so'

Tesla's CEO shared that its engineers have fully overhauled the Autopilot software stack and are almost ready to share a dramatic upgrade.

Antuan Goodwin
Sept. 23, 2020 7:53 a.m. PT

34

LISTEN - 01:44

AI in the 2010s

THE WALL STREET JOURNAL.

Amazon Wants to Ship Your Package Before You Buy It

By Greg Bensinger
Jan. 17, 2014 3:12 pm ET

SAVE SHARE

BLOOMBERG

ROAD/SHOW BY CNET

Tesla's fu Autopilo month o

Tesla's CEO sha overhauled the almost ready to

Antuan Goodwin
Sept. 23, 2020 7:53 a.m. PT

34

LISTEN - 01:44

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute

Help
Learn to edit
Community portal
Recent changes
Upload file

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

Elon Musk's Tesla Roadster

From Wikipedia, the free encyclopedia

"SpaceX Roadster" redirects here. For a planned "SpaceX option package" using cold gas thrusters, see Tesla Roadster (2020).

Elon Musk's Tesla Roadster is an electric sports car that served as the dummy payload for the February 2018 Falcon Heavy test flight and became an artificial satellite of the Sun. "Starman", a mannequin dressed in a spacesuit, occupies the

Elon Musk's Tesla Roadster

Roadster car mounted on Falcon Heavy upper-stage; Earth in the background

AI in the 201

THE WALL STREET JOURNAL.

Amazon Wants to Ship Your Package Before You Buy It

By Greg Bensinger
Jan. 17, 2014 3:12 pm ET

SAVE SHARE

BLOOMBERG

ROAD/SHOW BY CNET

Tesla's fu Autopilo month o

Tesla's CEO sha overhauled the almost ready to

Antuan Goodwin
Sept. 23, 2020 7:53 a.m. PT

34

LISTEN · 01:44

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute

Help
Learn to edit
Community portal
Recent changes
Upload file

Article Talk

Elon Musk's

From Wikipedia, the free e

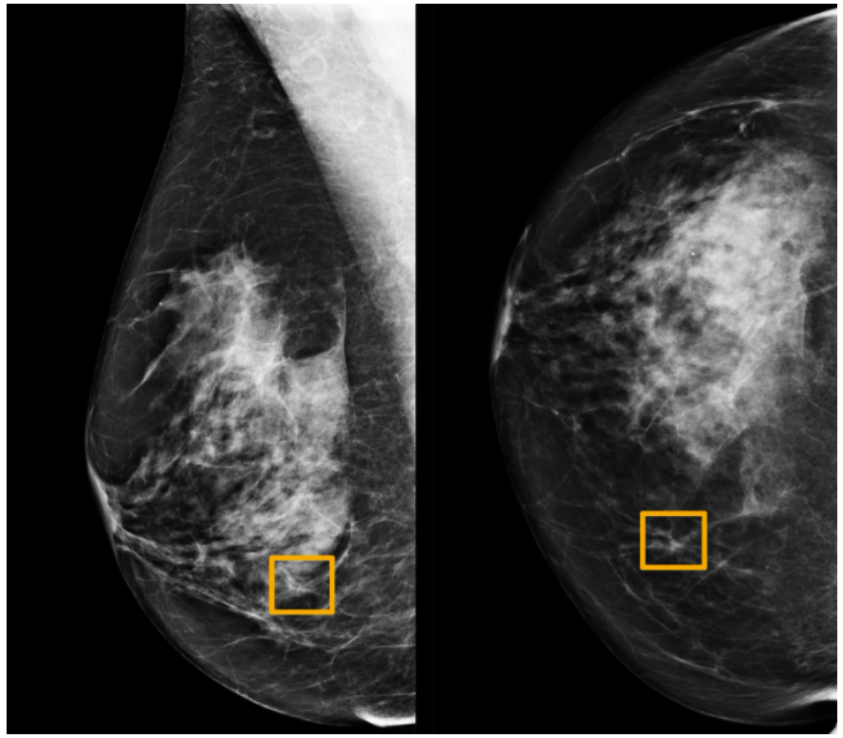
"SpaceX Roadster" using cold gas thrus

Elon Musk's Tesla Roadster is an electric sports car that served as dummy payload for the February 2018 Falcon Heavy test flight and became an artificial sate of the Sun. "Starman", a mannequin dressed in a spacesuit, occupies the

The New York Times

A.I. Is Learning to Read Mammograms

Computers that are trained to recognize patterns and interpret images may outperform humans at finding cancer on X-rays.



A yellow box indicates where an A.I. system found cancer hiding inside breast tissue. Six previous radiologists failed to find the cancer in routine mammograms. Northwestern University

By Denise Grady

Jan. 1, 2020

AI in the 201

Sign in

Contribute →

The Guardian

The Washington Post

Democracy Dies in Darkness

HUD is reviewing Twitter's and Google's ad practices as part of housing discrimination pro

The Dept. of Housing and Urban Development charged Facebook March 28 with violating the Fair Housing Act. (Reuters)

By Tracy Jan and Elizabeth Dwoskin

March 28, 2019 at 6:59 p.m. EDT

The Trump administration delivered its first sanction of a tech giant Thursday, charging Facebook with housing discrimination in a move that could threaten the way the industry makes its profits.

BLOOMBERG

There is a saying in computer science: garbag
garbage out. When we feed machines data th
our prejudices, they mimic them – from anti
chatbots to racially biased software. Does a h
future await people forced to live at the mercy of
algorithms?

Antuan Goodwin

Sept. 23, 2020 7:53 a.m. PT

34

acesuit, occupies the

ALGORITHM WATCH

story

Google apologizes after its Visi produced racist results

Published: April 7, 2020

Category: story

By Nicolas Kayser-Bril • nkb@algorithmwatc

A Google service that automatically labels i
starkly different results depending on skin
image. The company fixed the issue, but th
much broader.

The New York Times

A.I. Is Learning to Read Mammograms

Computers that are trained to recognize patterns and interpret
humans at finding cancer on X-rays.

The New York Times

Surveillance cameras have been deployed across Detroit as part of Project Green Light, which is meant to deter crime. Brittany Greeson for The New York Times

As Cameras Track Detroit's Residents, a Debate Ensues Over Racial Bias

Studies have shown that facial recognition software can return more false matches for African-Americans than for white people, a sign of what experts call "algorithmic bias."

By Amy Harmon

July 8, 2019

The Socially Conscious Data Scientist's Agenda

1. We can define and measure algorithmic bias

2. We can isolate the root cause of (poor) algorithmic behavior

3. We can take action to make algorithms more fair

What is algorithmic bias?

Case study

In the U.S., kidney function measurements are adjusted by race

- The eGFR is the standard-of-care for measuring kidney function
- It’s calculated by measuring the level of creatinine in a blood sample
- Because “African Americans” have higher muscle mass, the CKD-EPI algorithm increases their scores
- A higher score indicates higher kidney function

The CKD-EPI eGFR equation is racially biased

HAS
HAUTE AUTORITÉ DE SANTÉ

TEXTE COURT DU RAPPORT D'EVALUATION TECHNOLOGIQUE

Evaluation du débit de filtration glomérulaire
et du dosage de la créatininémie
dans le diagnostic de la maladie rénale chronique
chez l'adulte

Décembre 2011

Service évaluation des actes profi

EXTRA nephron

Nephron Extra 2012;2:293–302
DOI: 10.1159/000343899
Published online: December 1, 2012

Original Paper

Race Adjustment for Estimating Glomerular Filtration Rate Is Not Always Necessary

Juliana A. Zanocco Sonia K. Nishida Michelle Tiveron Passos
Amélia Rodrigues Pereira Marcelo S. Silva
Aparecido B. Pereira Gianna Mastroianni Kirsztajn

Glomerulopathy Section, Division of Nephrology, Medicine Department, Federal University of São Paulo (UNIFESP), São Paulo, Brazil

ltration rate · Glomerulonephritis

te (eGFR) is very important in clinical practice, populations. We aimed at establishing the best nphasis on the need for race correction. ***Meth-*** idney disease (CKD) and 42 without previously ned by urinalysis. Serum creatinine and plasma GFR was estimated by the Mayo Clinic, abbrevi- RD) and Chronic Kidney Disease Epidemiology e clearance was estimated by the Cockcroft- was used as the gold standard for GFR determi- formula (BreGFR). ***Results:*** Measured and esti- 7% female, with a mean age of 41 years (range clients among the plasma clearance of iohexol) and corresponded to the following scores: CG 304; MDRD 0.848; MDRD1 (without race adjust- ace adjustment) 0.876, and BreGFR 0.844. ***Con-***

ction, Federal University of São Paulo (UNIFESP)

-900 (Brazil)
o.unifesp.br

KARGER

ELSEVIER

European Journal of Obstetrics & Gynecology and Reproductive Biology
Volume 176, May 2014, Pages 200-201

LETTER TO THE EDITOR

Adjustment for race in the estimation of glomerular filtration rate (GFR) is inappropriate in the British postnatal population

Anna L. Roberts, Alastair Ferraro, Amanda Green, Pam Loughna, Fiona Broughton-Pipkin

Many people see this as unfair. Can you think of any reasons why?

What is fairness?

Two definitions used in the algorithmic community

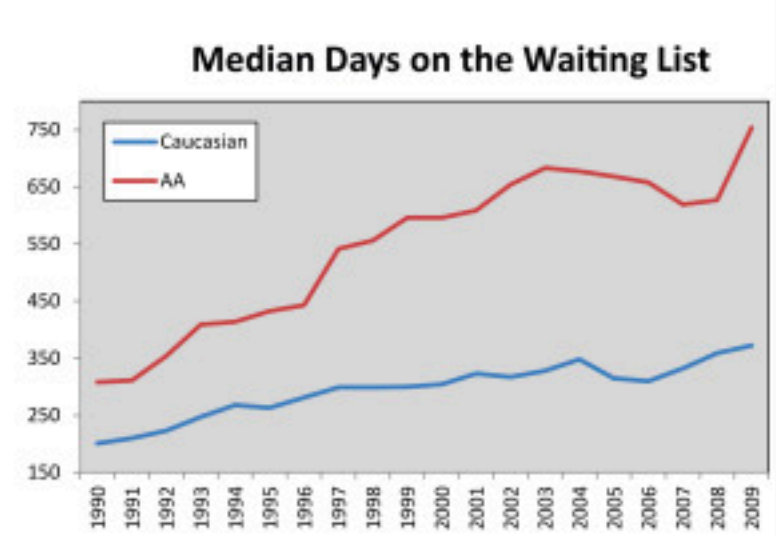
- Group Fairness
- Identifiable groups should be treated similarly to the population as a whole

- Individual Fairness
- Similar individuals should be treated similarly

- Adapted from Sahil Verma and Julia Rubin. 2018. Fairness Definitions Explained. https://fairware.cs.umass.edu/papers/Verma.pdf.

Is the CKD-EPI algorithm Group Fair?

- **Group Fairness Definition**
- Protected groups should be treated similarly to
  non-protected groups and the population as a whole



Source: Taber et al., Twenty years of evolving trends in racial disparities for adult kidney transplant recipients. Kidney Int. 2016.

Is the CKD-EPI algorithm Individually Fair?

- **Individual Fairness Definition**
- Similar individuals should be treated similarly

VIEWPOINT

Reconsidering the Consequences of Using Race to Estimate Kidney Function

Nwamaka Denise Eneanya, MD, MPH
Renal-Electrolyte and Hypertension Division, Perelman School of Medicine, University of Pennsylvania, Philadelphia; and Palliative and Advanced Illness Research Center, Perelman School of Medicine, University of Pennsylvania, Philadelphia.

Wei Yang, PhD
Department of Biostatistics, Epidemiology, and Informatics, Perelman School of Medicine, University of Pennsylvania, Philadelphia.

Peter Philip Reese, MD, MSCE
Renal-Electrolyte and Hypertension Division, Perelman School of Medicine, University of Pennsylvania, Philadelphia; and Department of Biostatistics, Epidemiology, and Informatics, Perelman School of Medicine, University of Pennsylvania, Philadelphia.

Clinicians estimate kidney function to guide important medical decisions across a wide range of settings, including assessing the safety of radiology studies, choosing chemotherapy, and reviewing the use of common nonprescription medications such as nonsteroidal anti-inflammatory drugs. Because direct measurement of kidney function is infeasible at the bedside, the usual approach involves using estimating equations that rely on serum creatinine. These equations assign a higher estimated glomerular filtration rate (eGFR) to patients who are identified as black. Yet in some medical and social science disciplines, a consensus has emerged that race is a social construct rather than a biological one.[1] In this Viewpoint, we argue that the use of kidney function estimating equations that include race as a vari-

equation, were generated in large cohorts of individuals who underwent gold-standard measurement of "true" GFR by infusing iothalamate or another chemical into the blood and quantifying its urine clearance. Investigators found that black race was independently associated with a slightly higher GFR at the same serum creatinine level. This association has been justified by the assertion that black individuals release more creatinine into the blood, perhaps because of more muscle mass, although data remain inconclusive.[2-4] The CKD-EPI equation includes a race coefficient that increases the eGFR in black patients by about 16%. Estimated GFR equations also include age and sex because older individuals and women, on average, have less muscle than younger individuals and men, respectively; these generalizations have a stronger empirical basis than that for race.

Classifying patients according to ancestry (rather than race or ethnicity) has legitimate purposes to identify individuals at risk of complications from rare gene mutations like sickle cell trait or cystic fibrosis. However, eGFR equations are distinct because they instead assert that existing organ function is different between individuals who are otherwise identical except for race. Population studies reveal only small differences in gene distributions between racial groups while showing greater variation between individuals of the same race. Meanwhile, the history of medicine offers abundant evidence that racial categories were often generated arbitrarily and at times implemented to reinforce social inequality.[5]

Racial categorization is often used in a nonstandardized way. Consider a hypothetical 50-year-old woman with a creatinine level of 2.0 mg/dL and no proteinuria. Her father self-identified as black race and her mother

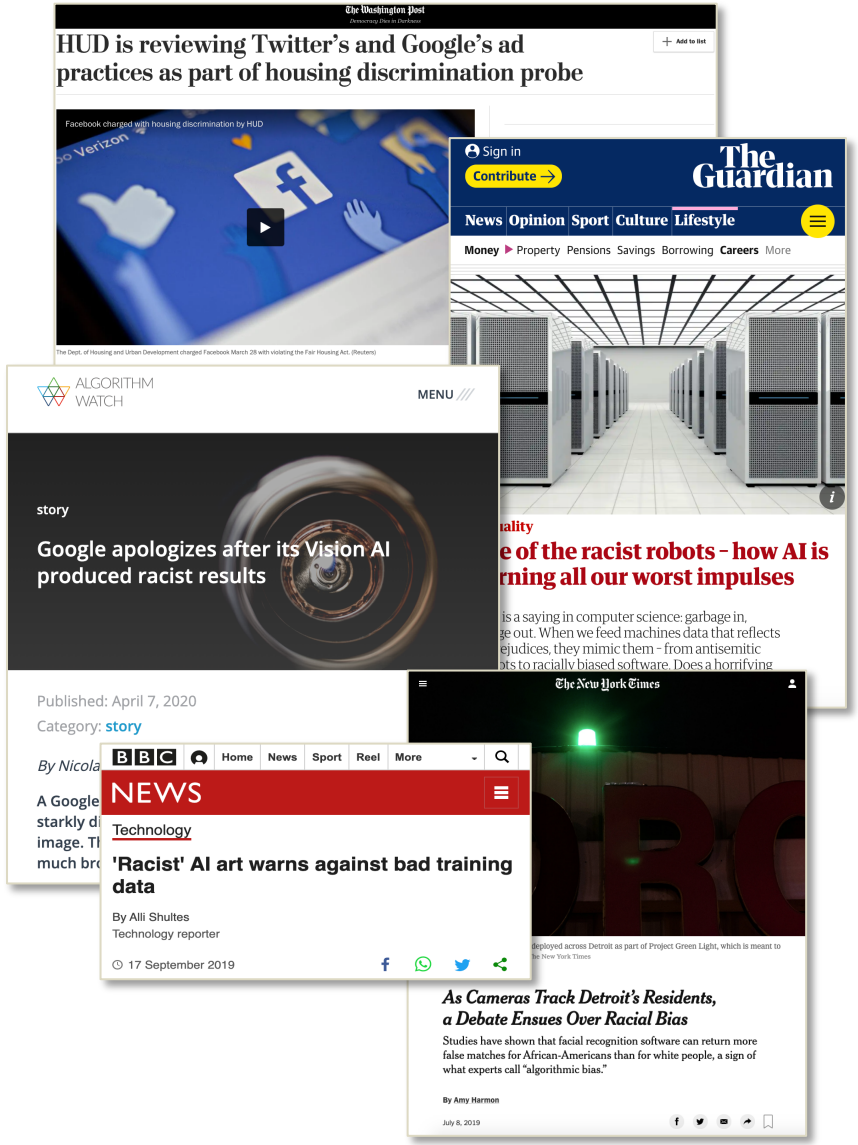
> Estimated GFR equations are distinct because they assert that existing organ function is different between individuals who are identical except for race.

consequences. Many essential medications including antibiotics are withheld from patients with a low eGFR or are administered at reduced doses. The authoritative Kidney Disease: Improving Global Outcomes (KDIGO) guidelines recommend nephrology referral if a patient's eGFR is less than 30 mL/min/1.73 m^2. If the patient in the above example were considered to be black, her eGFR would be 33 mL/min/1.73 m^2, but if she were considered to be white, her eGFR would be 28 mL/min/1.73 m^2 with the CKD-EPI equation (ie, below the threshold for referral). In addition, clinical trials commonly exclude patients with reduced kidney function. If this patient were considered to be black, she could enter some trials that would exclude her if she were considered to be white.

Perhaps the most concerning implication of race in eGFR is that it has the potential to reduce access to kidney transplantation, for which racial disparities are substantial. In the United States, being wait-listed for a kidney transplant requires an eGFR of less than

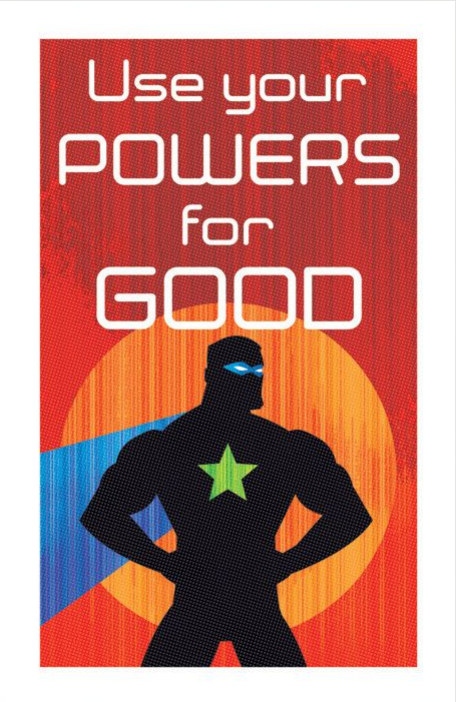
Corresponding Author: Peter Philip Reese, MD, MSCE, Center for Clinical Epidemiology and Biostatistics, University of Pennsylvania, 423 Guardian Dr, 917 Blockley Hall, Philadelphia, PA 19104 (peter.reese@uphs.upenn.edu).

Why does this keep happening?



Why isn’t fairness part of our process?

- **We have good intentions**



- **… but need mechanisms for action**

CHALLENGES

Hard to define

Hard to measure

Fairness is context-specific

(LACK OF) INCENTIVES

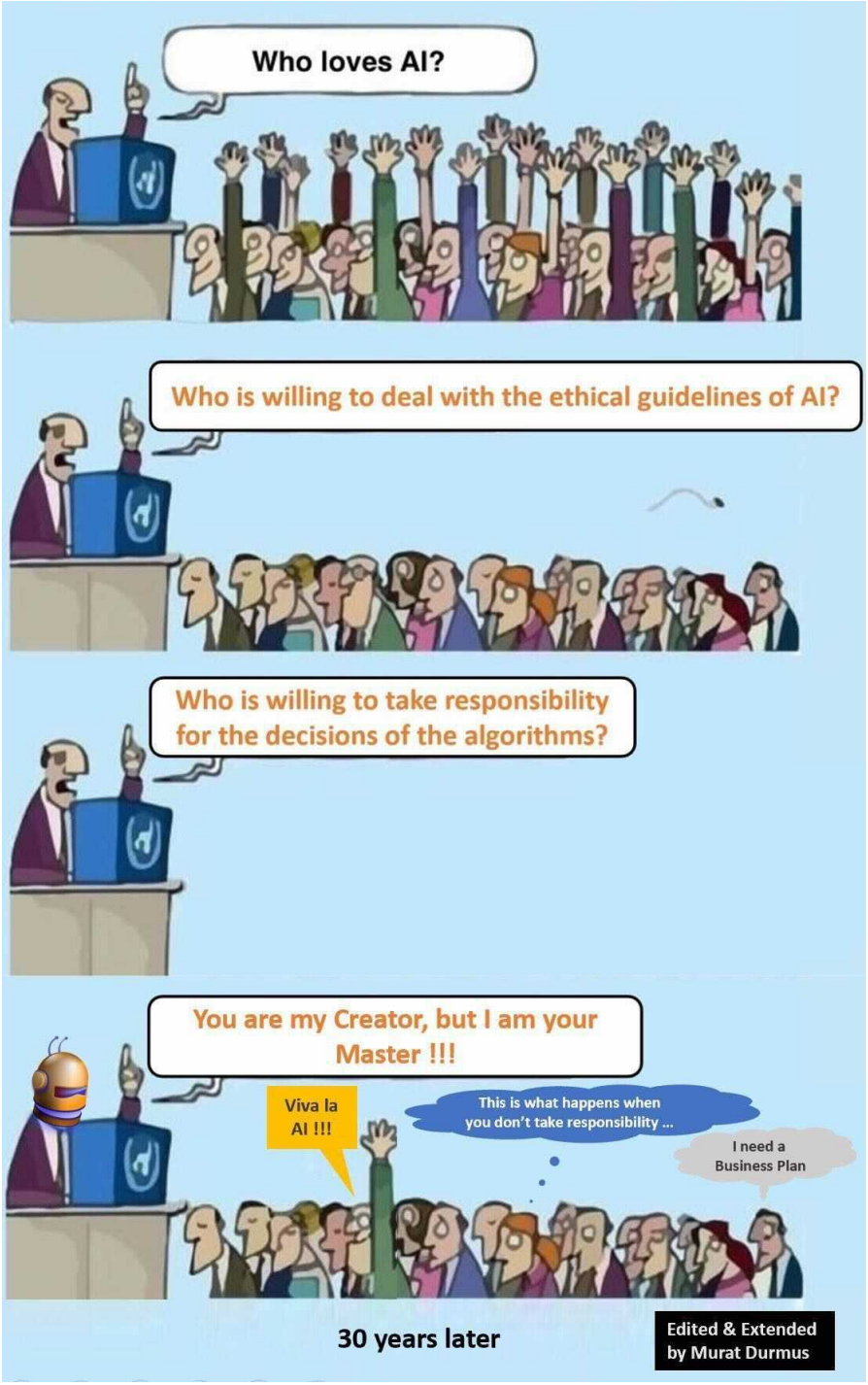
Lack of transparency

Lack of accountability

No hard business reason to prioritize fairness

Image: Rob Osborne

How will we end this?



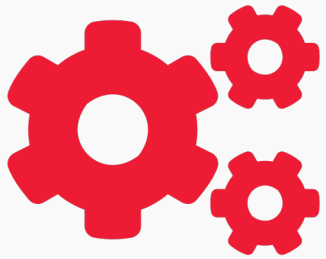
Ingredients of an algorithmic decision

TECHNOLOGY

- Data
- Algorithm
- …

PEOPLE

- Data Analyst/Scientist
- Business Owner
- End User
- …



PROCESS

- Model Training
- Evaluation
- Application
- …

How can we change these to mitigate algorithmic bias?

Process

What mechanisms can help us build fair models?

CHALLENGES

Hard to define

Hard to measure

Lack of transparency

Lack of accountability

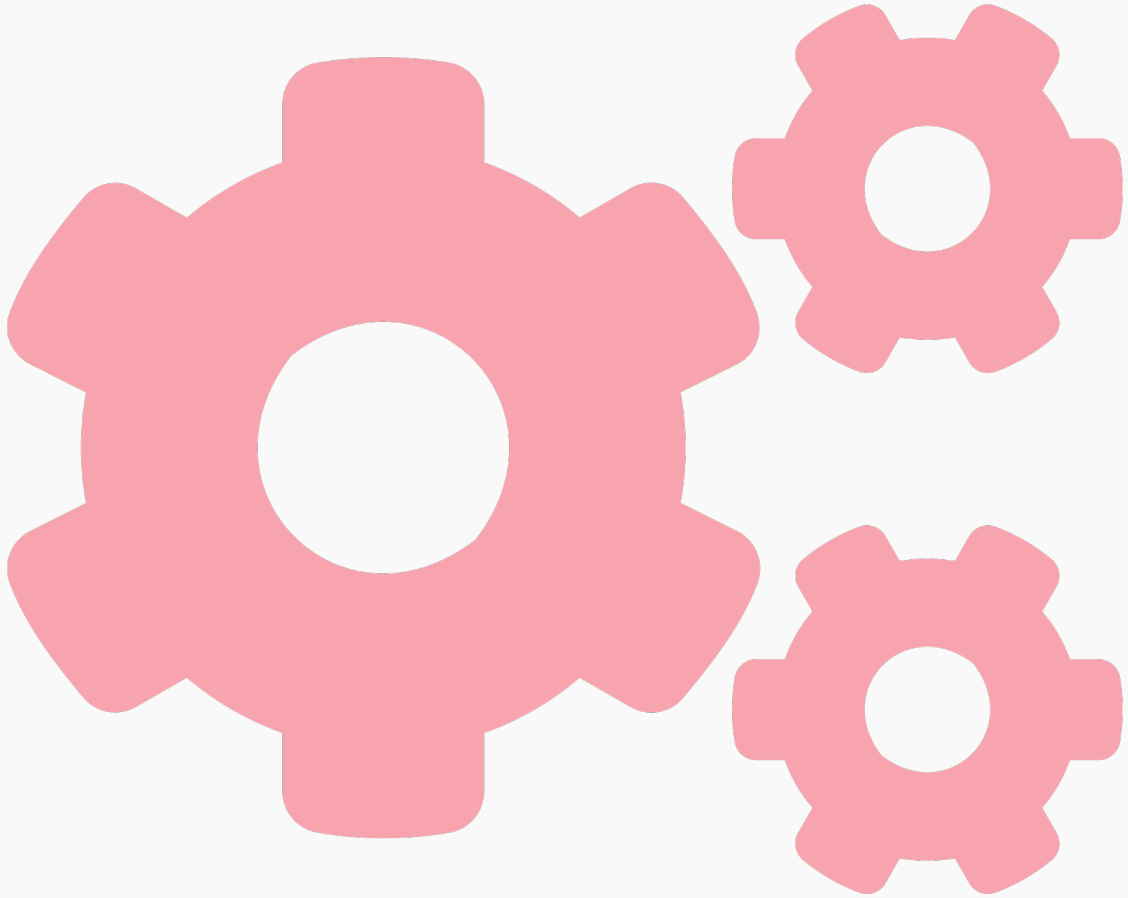
PROPOSED APPROACH

Fairness Statement

A commitment to defined and measurable fairness objectives

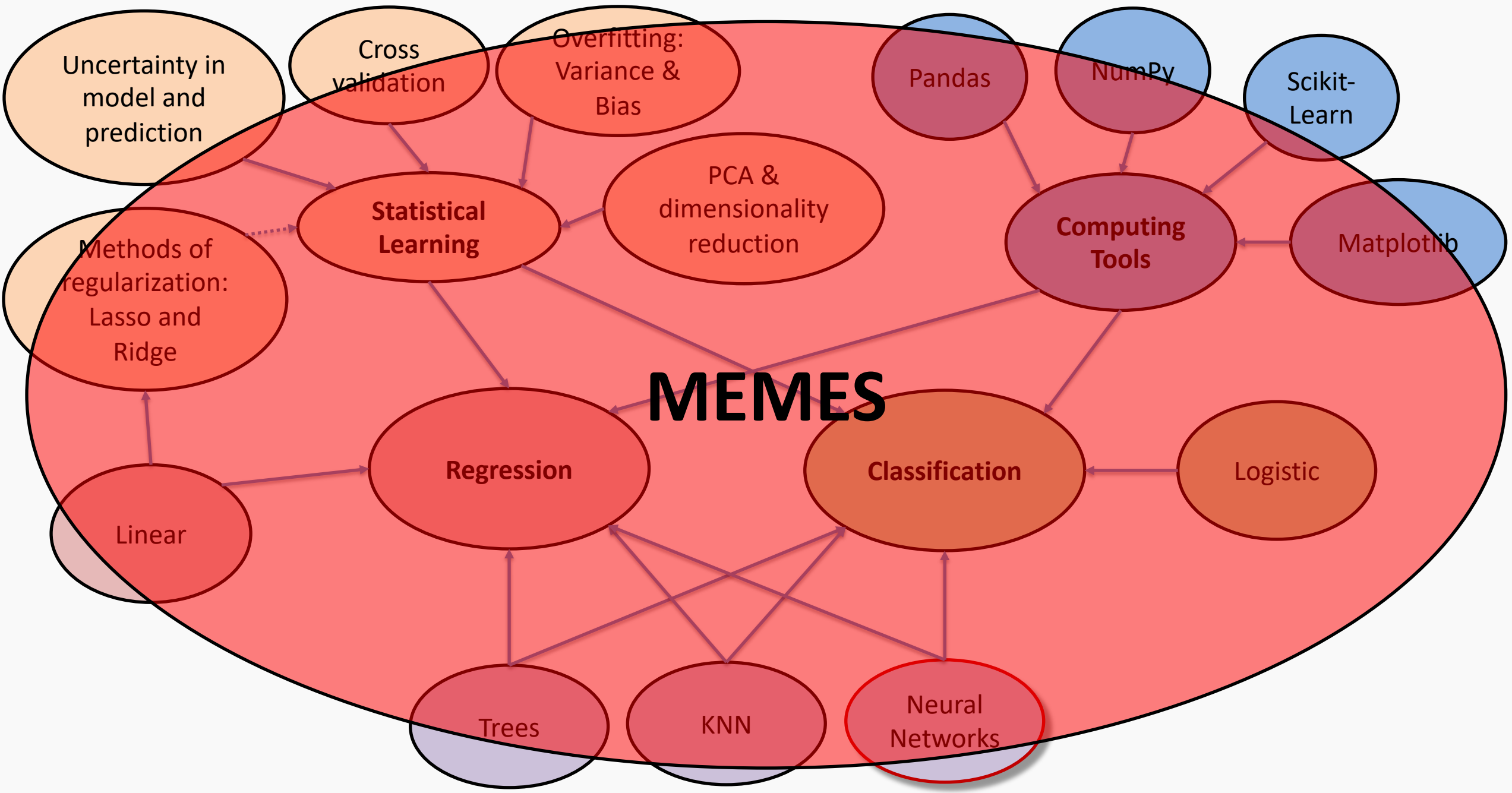
Algorithmic Practice Audit

An independent, third party review of processes and outcomes

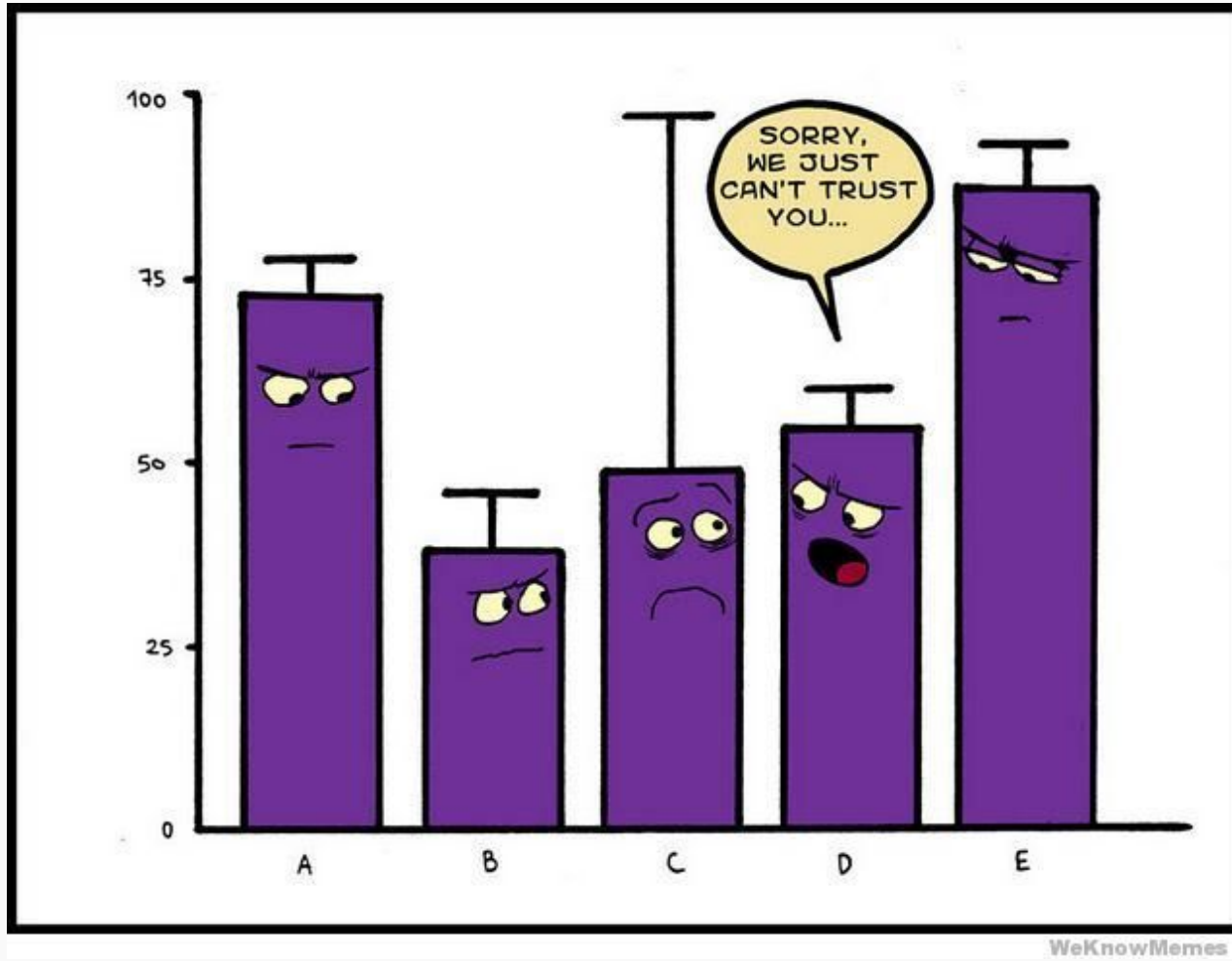


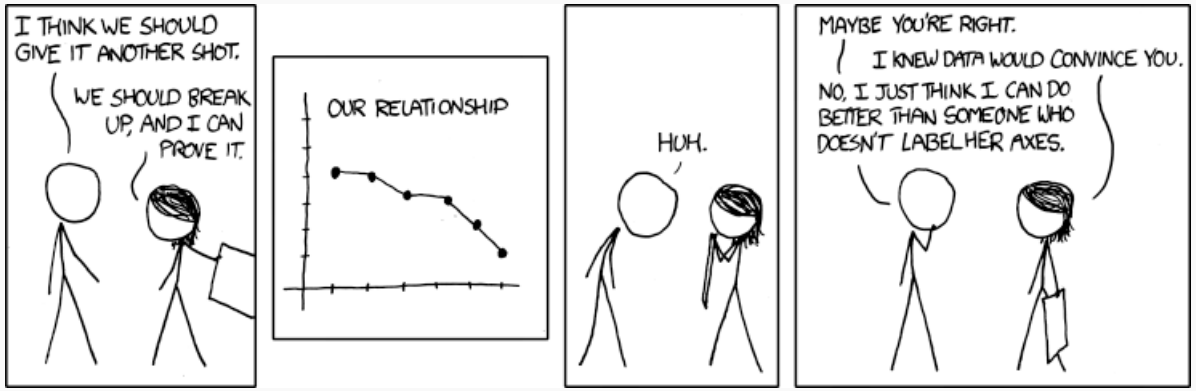
What will you do to create fair algorithms?

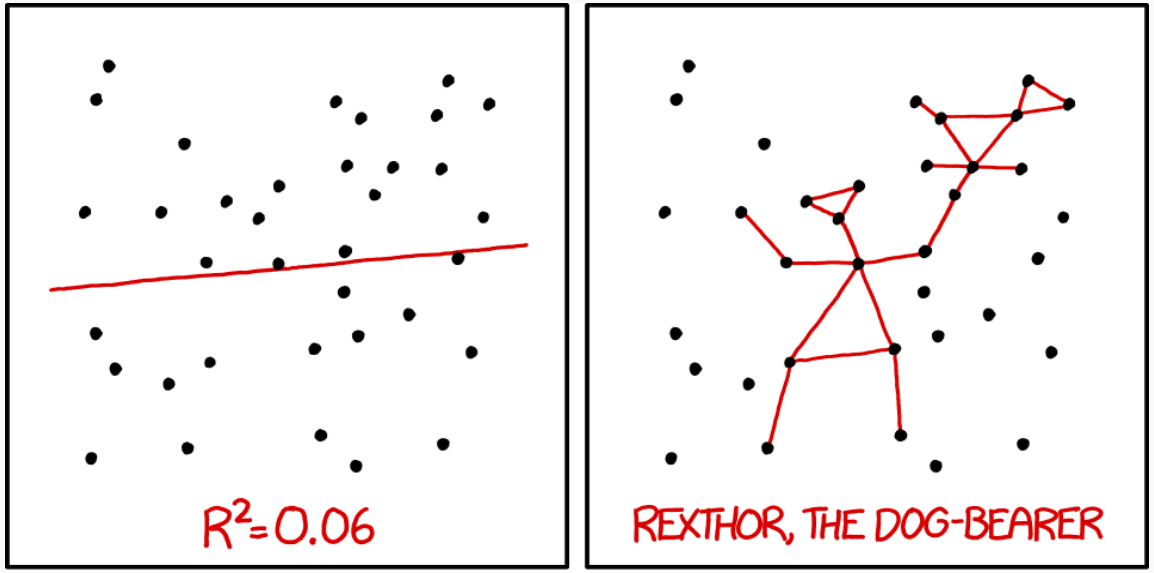
TECHNOLOGY	Are you following existing technical best practices, and using classes of fair algorithms?
PEOPLE	Are your data and tech teams representative of your customers and stakeholders?
PROCESS	Do you have mechanisms to ensure algorithmic fairness?


Uncertainty in model and prediction
Cross validation
Overfitting: Variance & Bias
Pandas
NumPy
Scikit-Learn
Statistical Learning
PCA & dimensionality reduction
Computing Tools
Matplotlib
Methods of regularization: Lasso and Ridge
MEMES
Regression
Classification
Logistic
Linear
Trees
KNN
Neural Networks

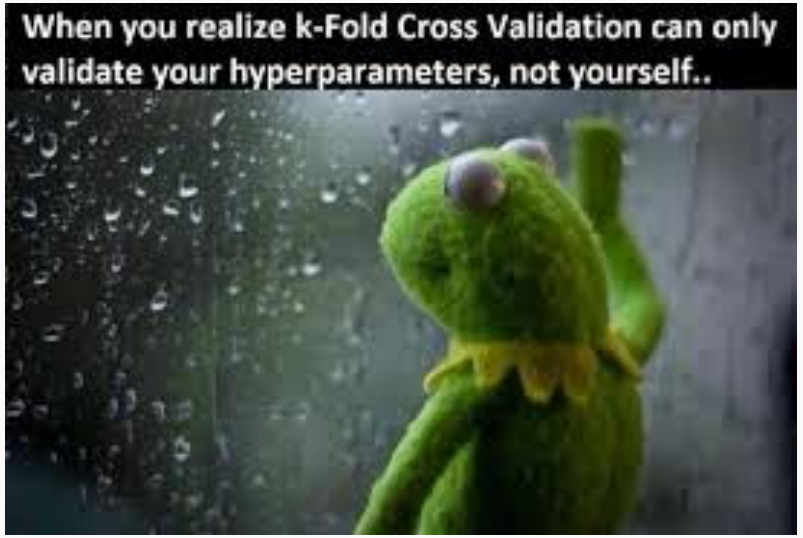

I finally remember what Zoom meetings remind me of.


100
75
50
25
0
A
B
C
D
E
SORRY, WE JUST CAN'T TRUST YOU...
WeKnowMemes



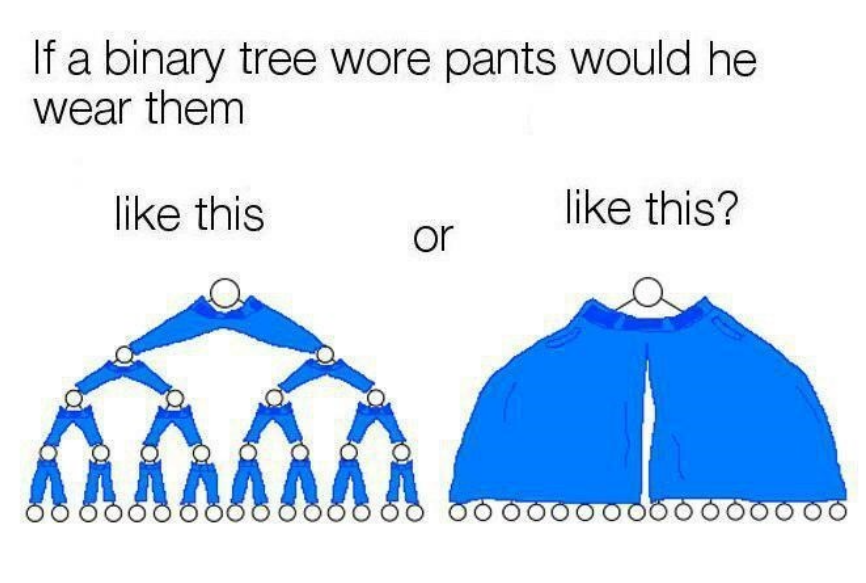


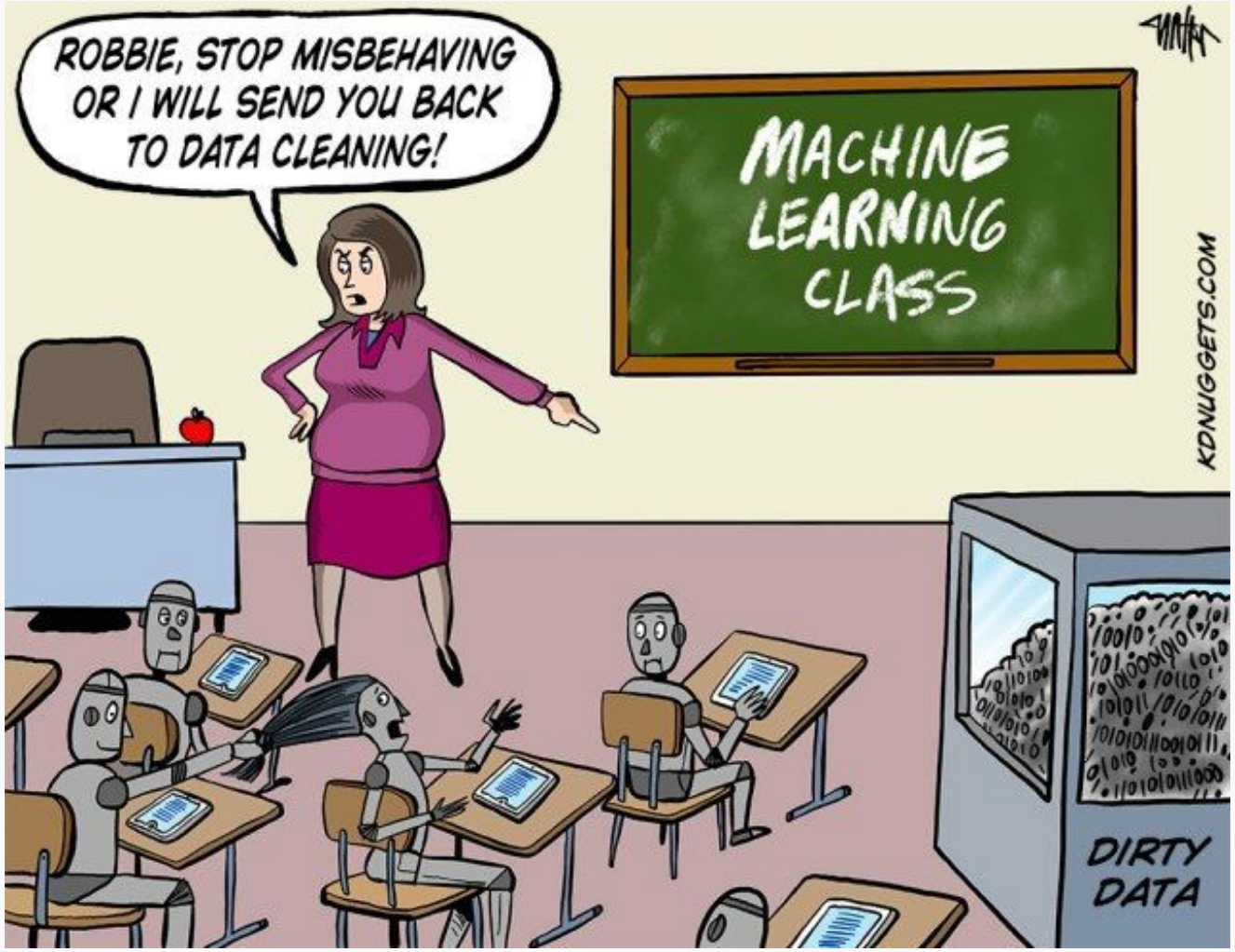
I DON'T TRUST LINEAR REGRESSIONS WHEN IT'S HARDER TO GUESS THE DIRECTION OF THE CORRELATION FROM THE SCATTER PLOT THAN TO FIND NEW CONSTELLATIONS ON IT.



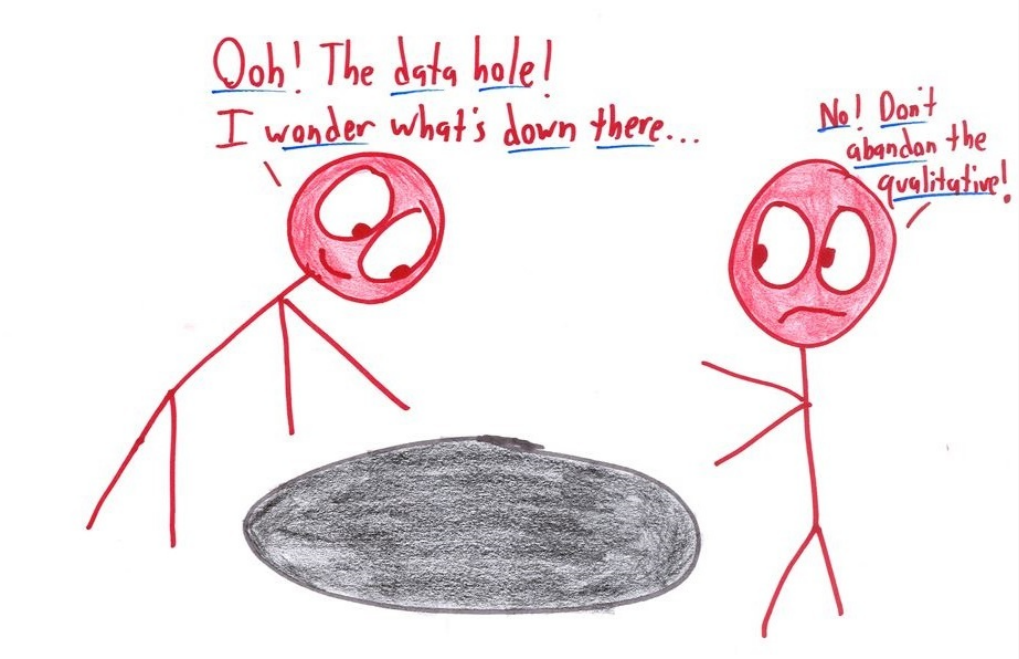
The secret is stackoverflow

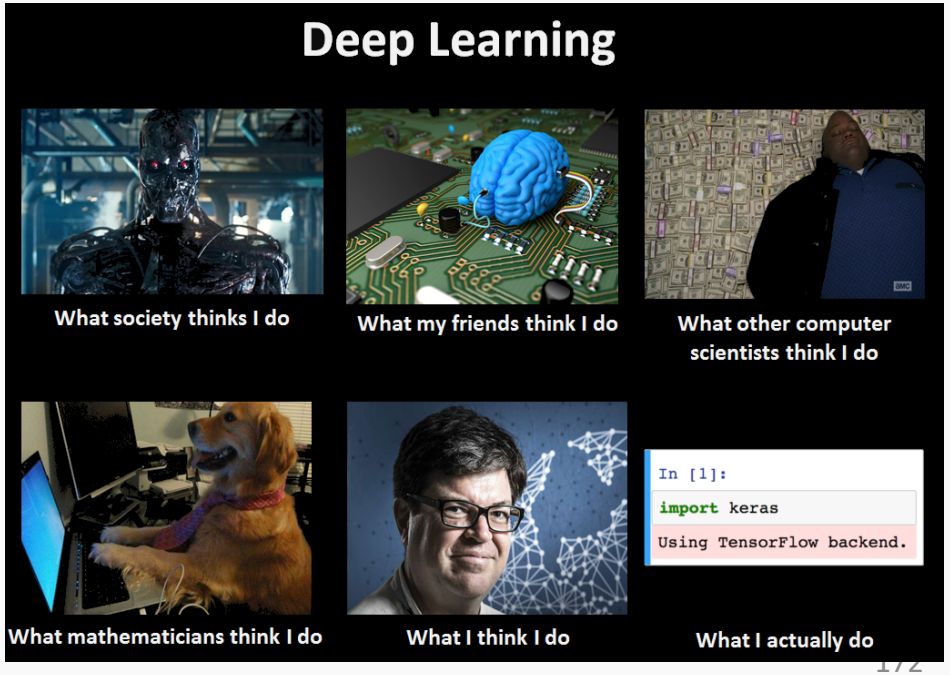

Tableau
66.6K views
0:01 / 1:29

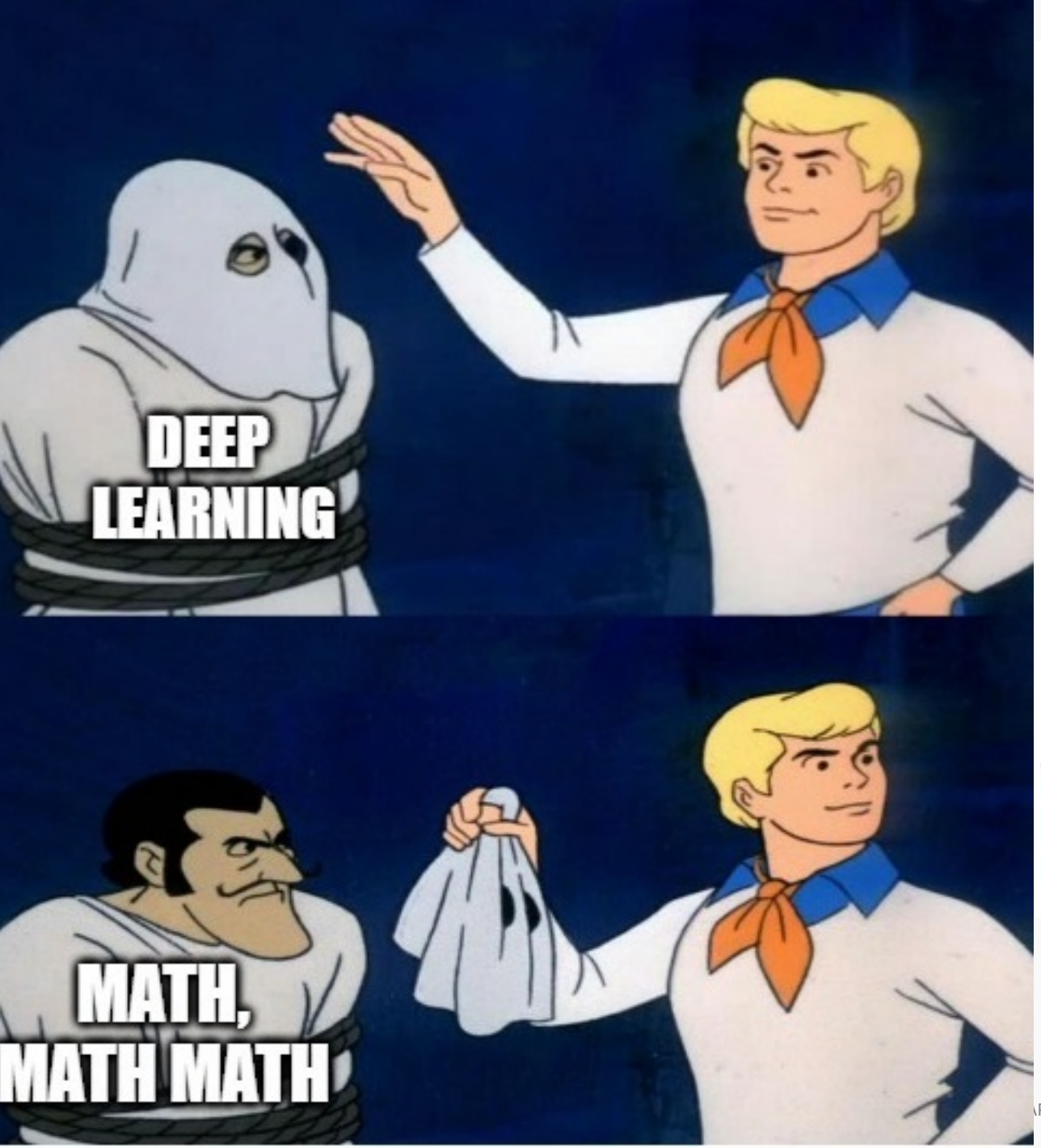

If a binary tree wore pants would he wear them
like this
or
like this?

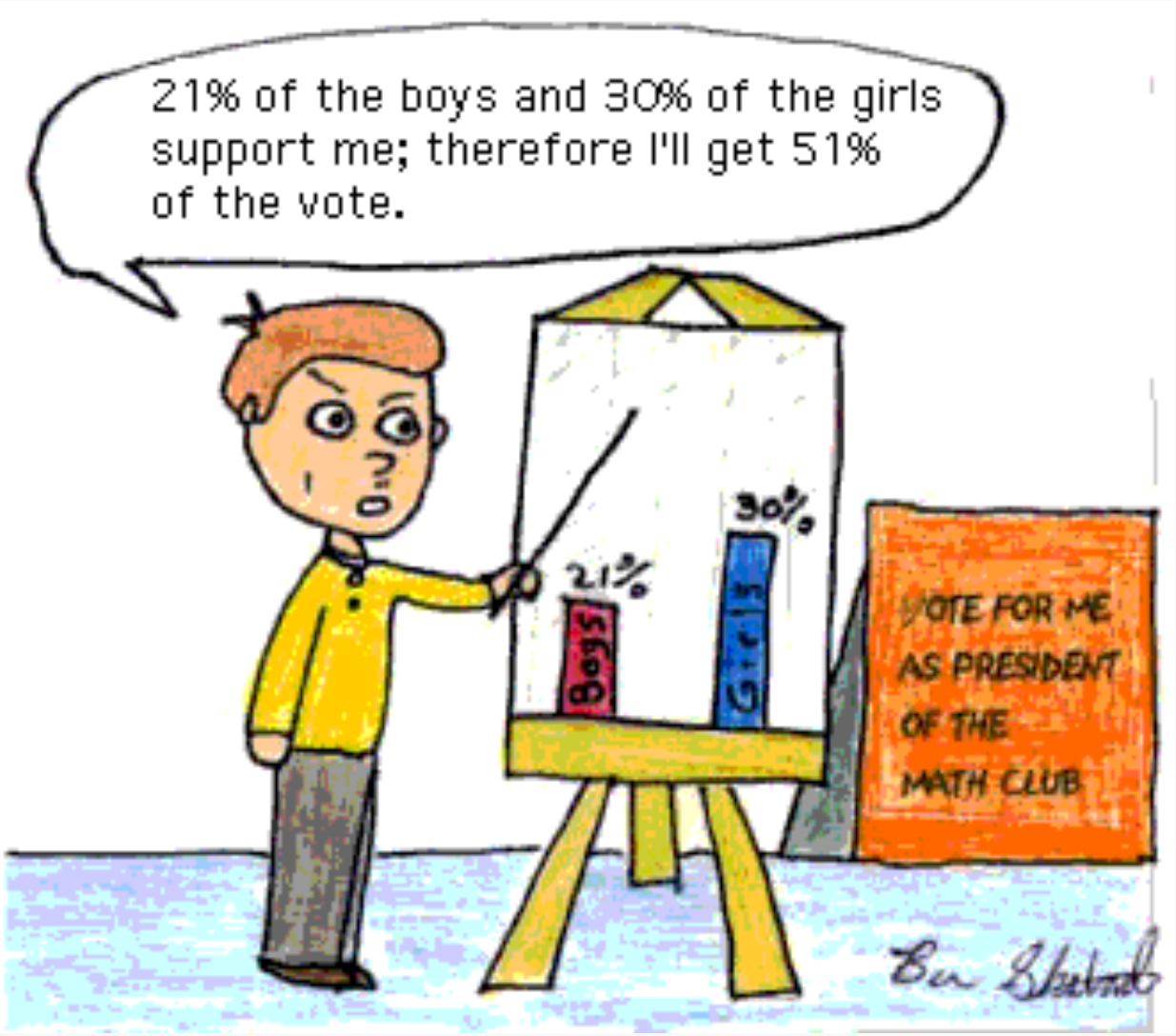

ROBBIE, STOP MISBEHAVING OR I WILL SEND YOU BACK TO DATA CLEANING!
MACHINE LEARNING CLASS
KDNUGGETS.COM
DIRTY DATA

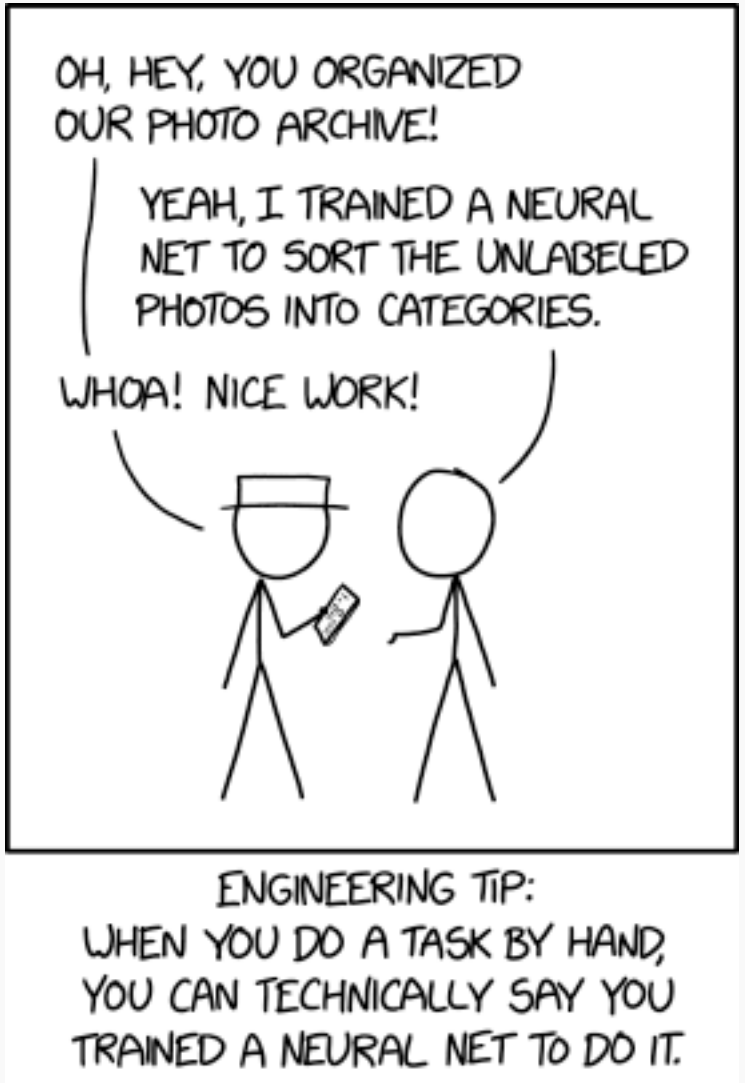

I VOTED.


Ooh! The data hole!
I wonder what's down there...
No! Don't abandon the qualitative!


Deep Learning
What society thinks I do
What my friends think I do
What other computer scientists think I do
What mathematicians think I do
What I think I do
In [1]:
import keras
Using TensorFlow backend.
What I actually do


DEEP LEARNING
MATH, MATH MATH


21% of the boys and 30% of the girls support me; therefore I'll get 51% of the vote.
21%
30%
Boys
Girls
VOTE FOR ME AS PRESIDENT OF THE MATH CLUB



How to bully machine learning training

Courses Related to Data Science

- **CS 109B: Advanced Topics in Data Science**
  - **https://harvard-iacs.github.io/2020-CS109B/**
- CS 171: Visualizations
- CS 181/281: Machine Learning
- CS 182: Artificial Intelligence (AI)
- CS 205: Distributive Computing
- Stat 110/210: Probability Theory
- Stat 111/211: Statistical Inference
- Stat 139: Linear Models
- Stat 149: Generalized Linear Models
- Stat 195: Intro to Statistical Machine Learning

This list is not exhaustive!

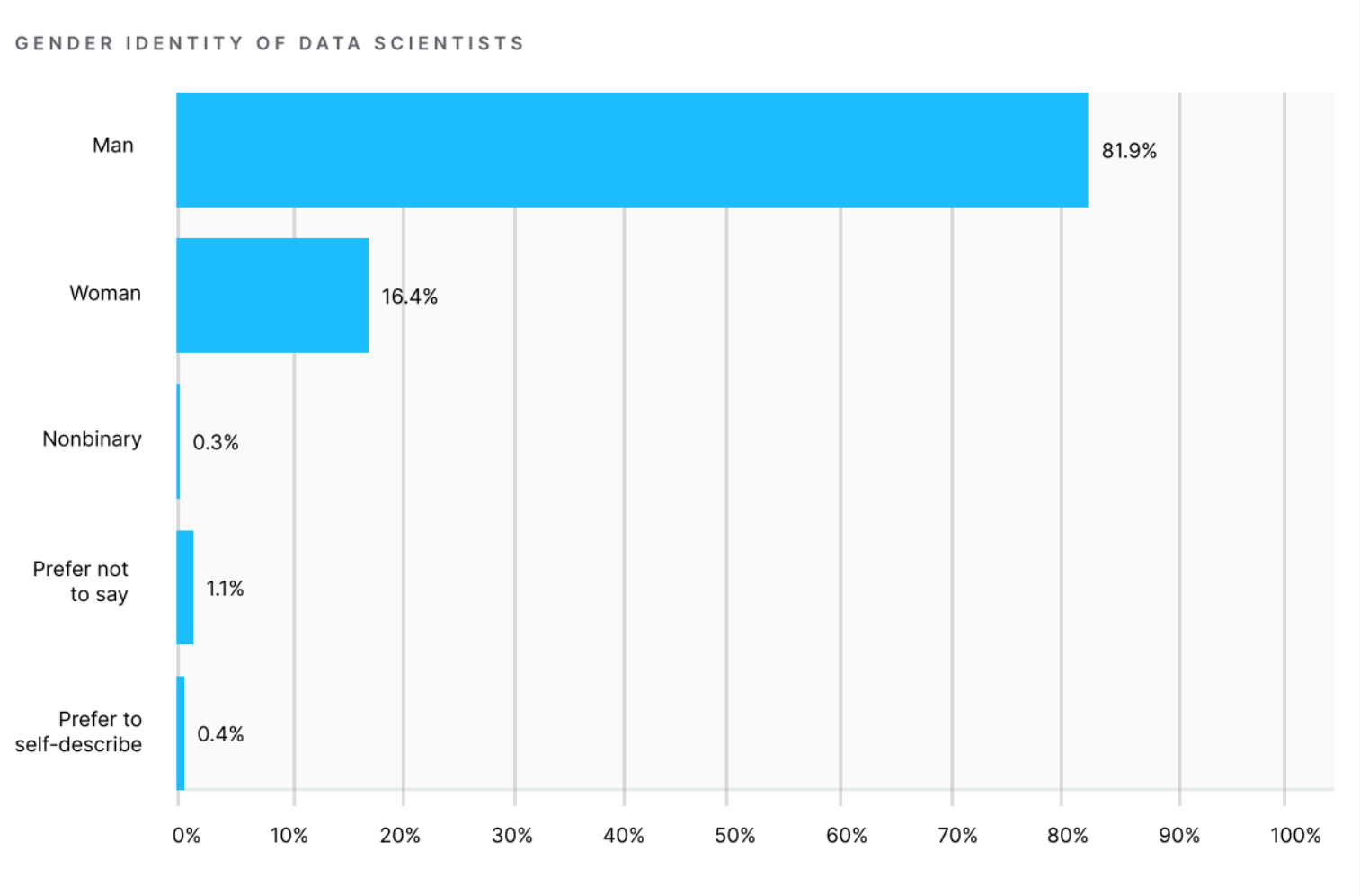
State of Machine Learning and Data Science 2020

Kaggle enterprise executive summary report

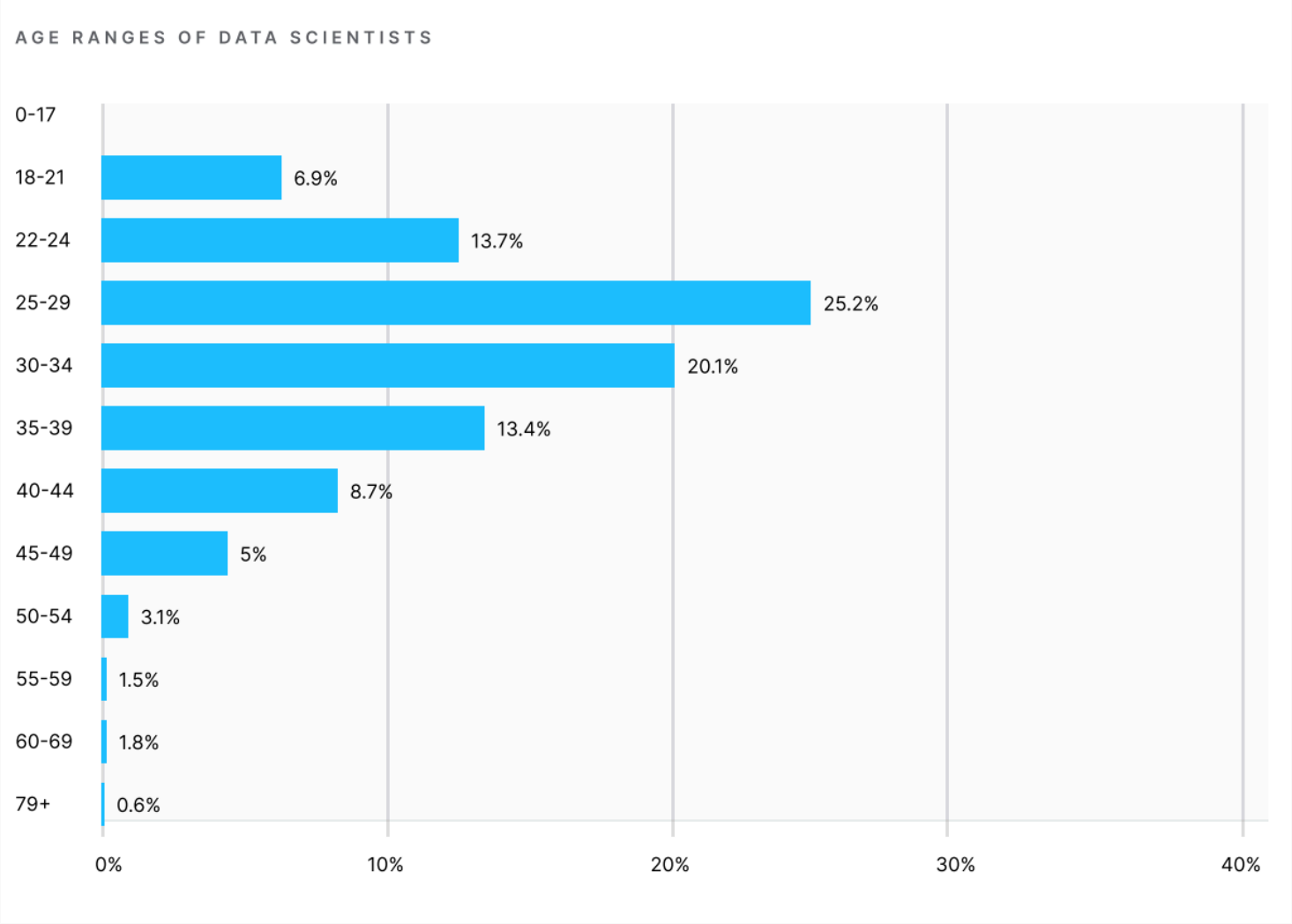
Kaggle surveyed its community of data enthusiasts to share trends within a quickly growing field.

Based on responses from 20,036 Kaggle members, they've created a report focused on the 13% (2,675 respondents) who are currently employed as data scientists.

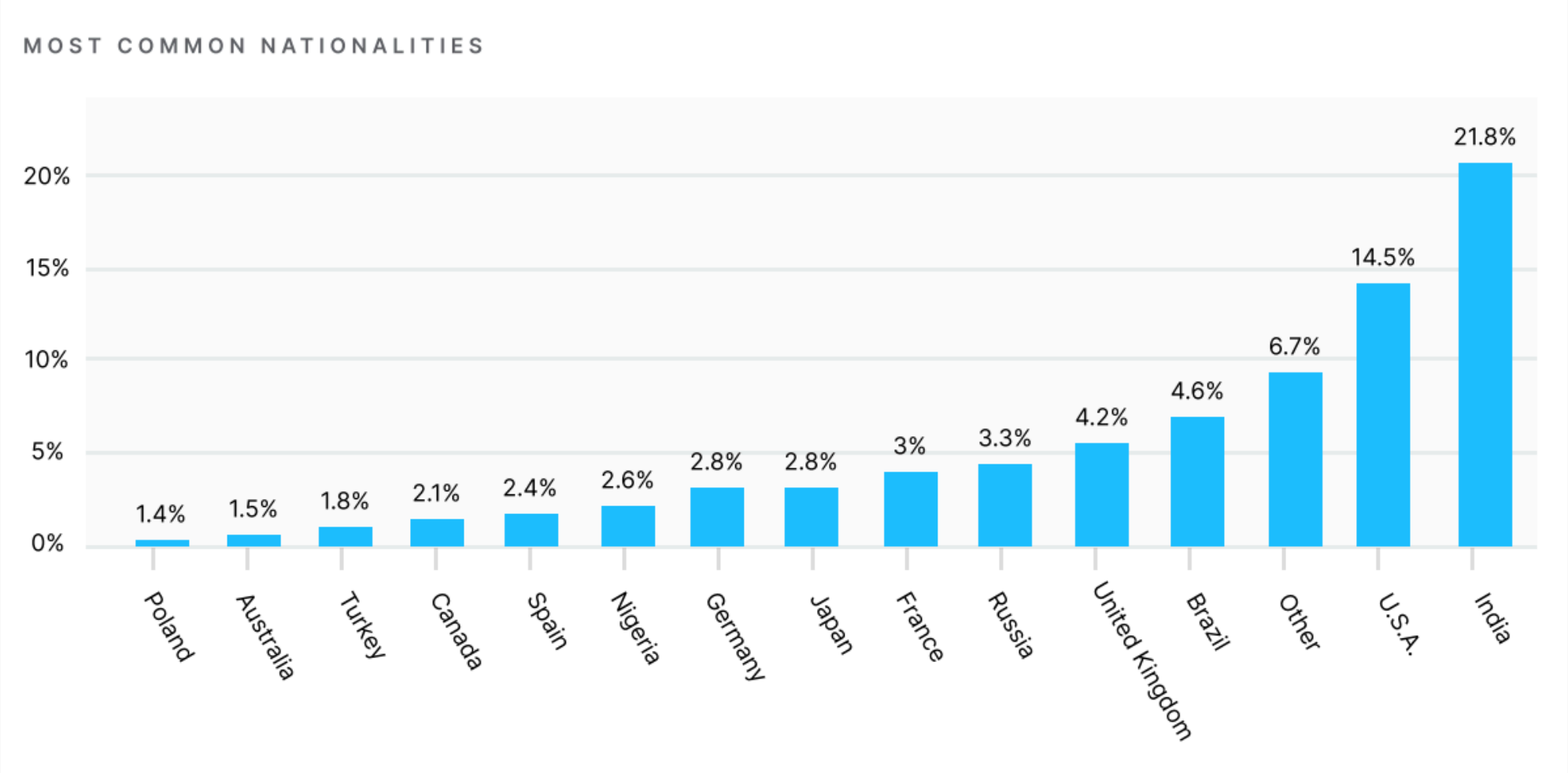
Key findings: Gender



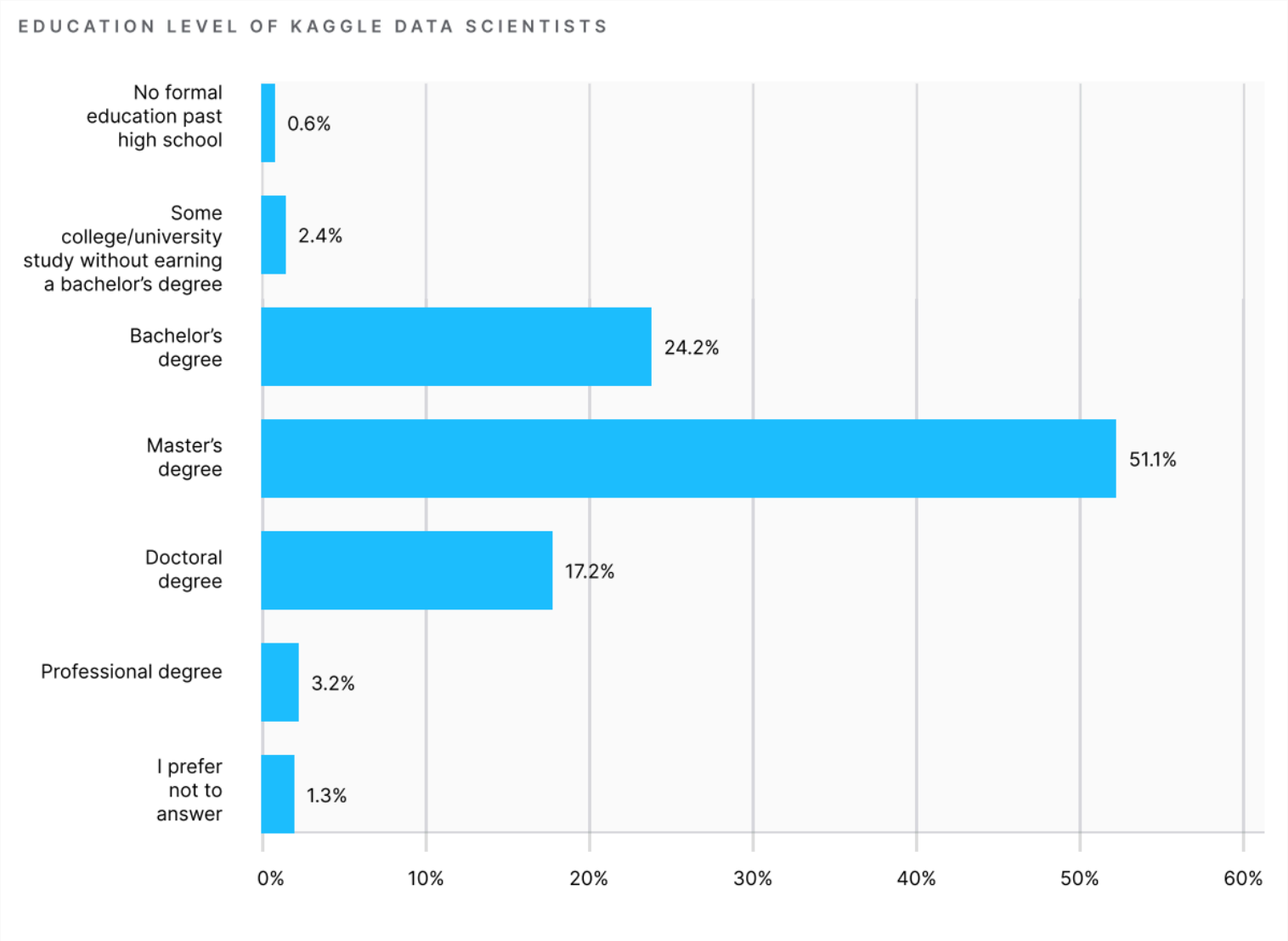
Key findings: Age



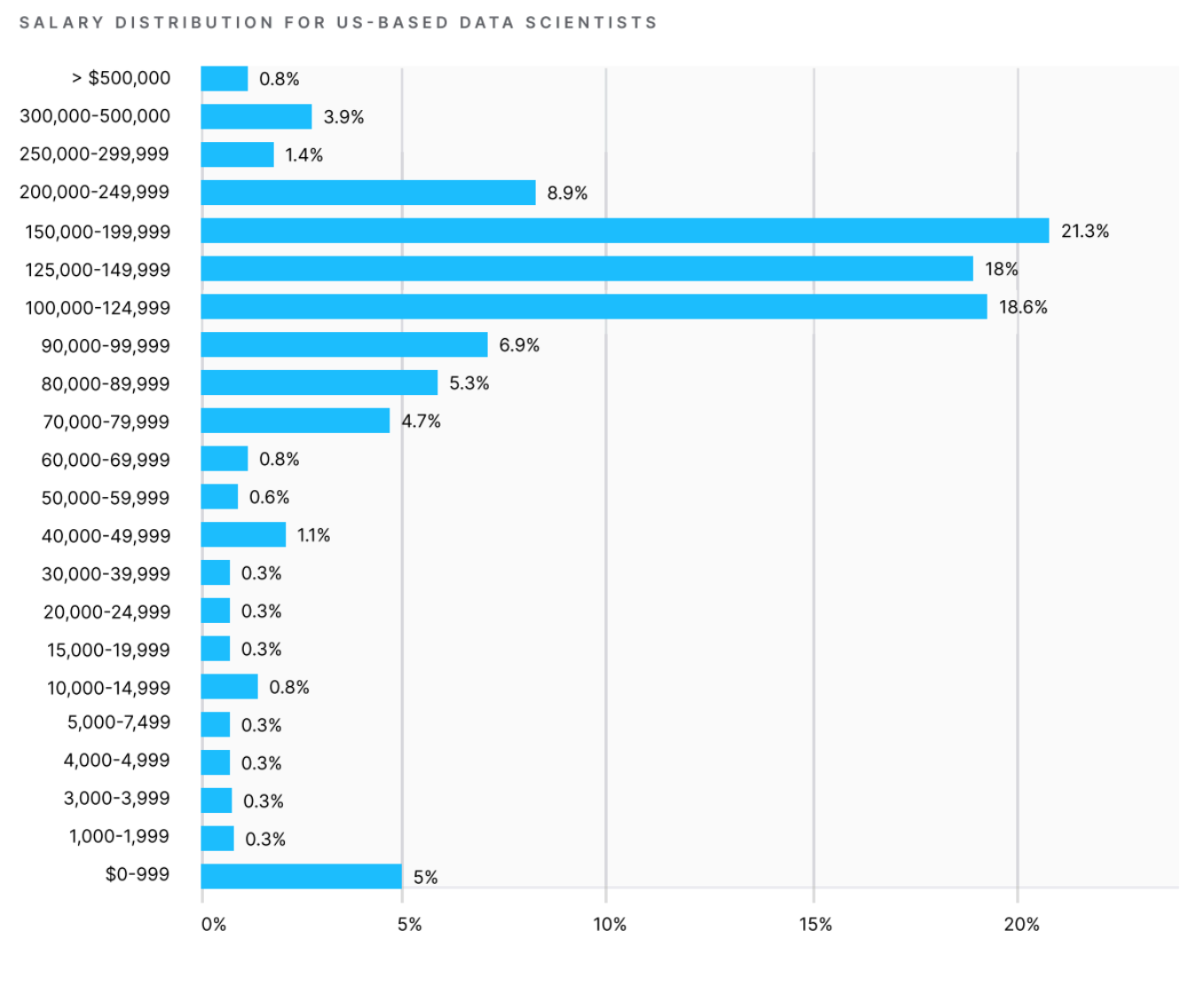
Key findings: Nationalities



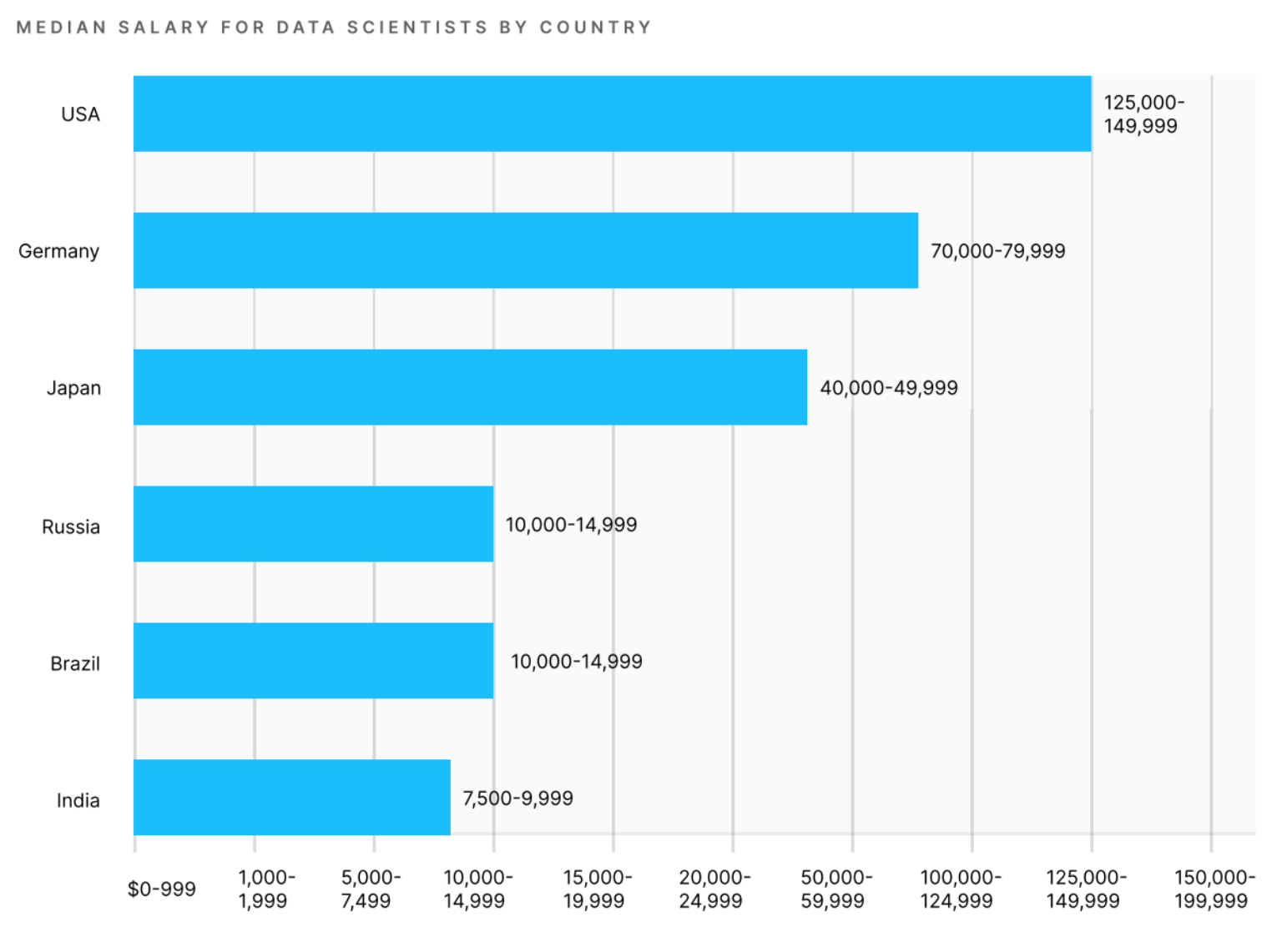
Key findings: Education



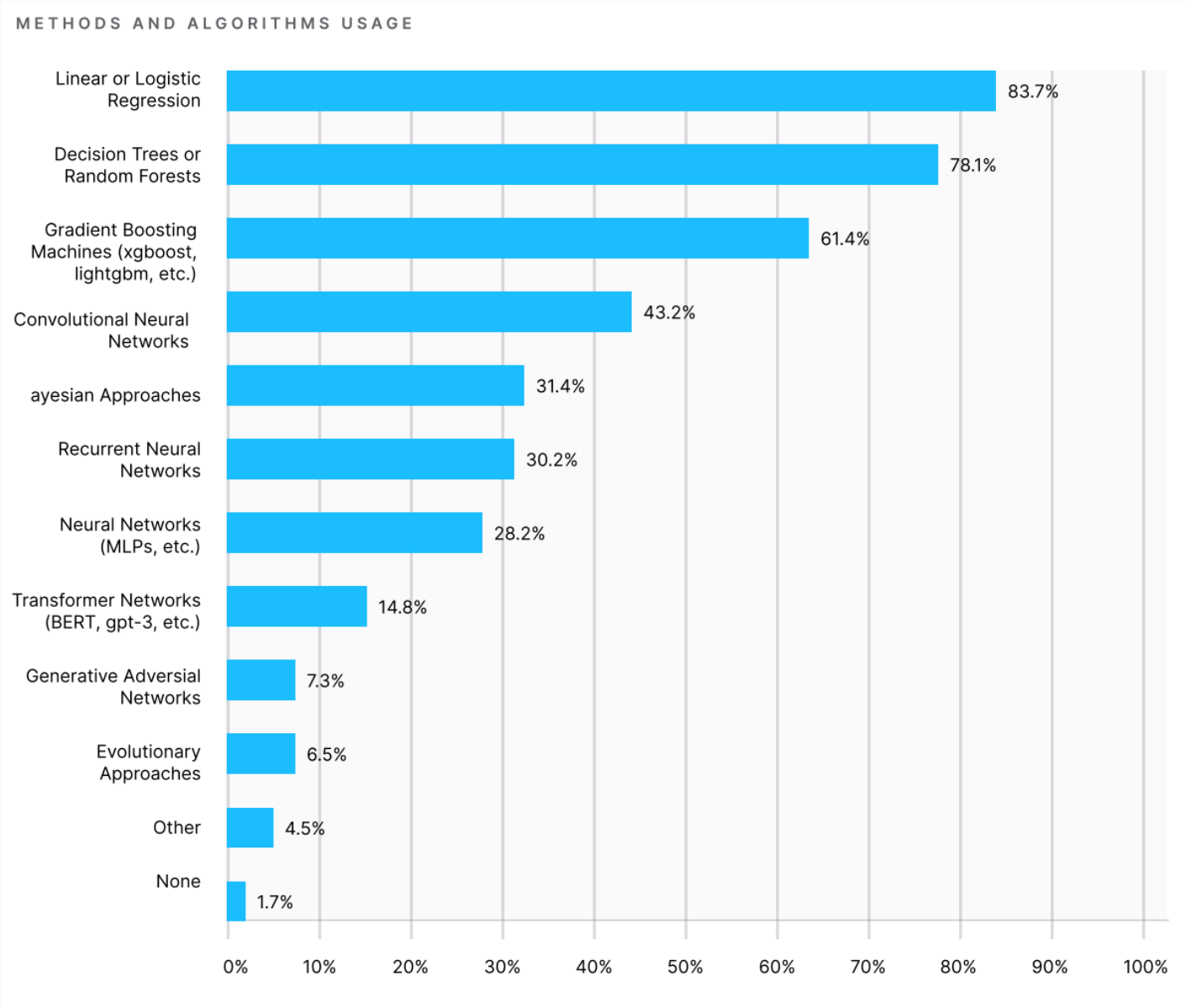
Key findings: Salary



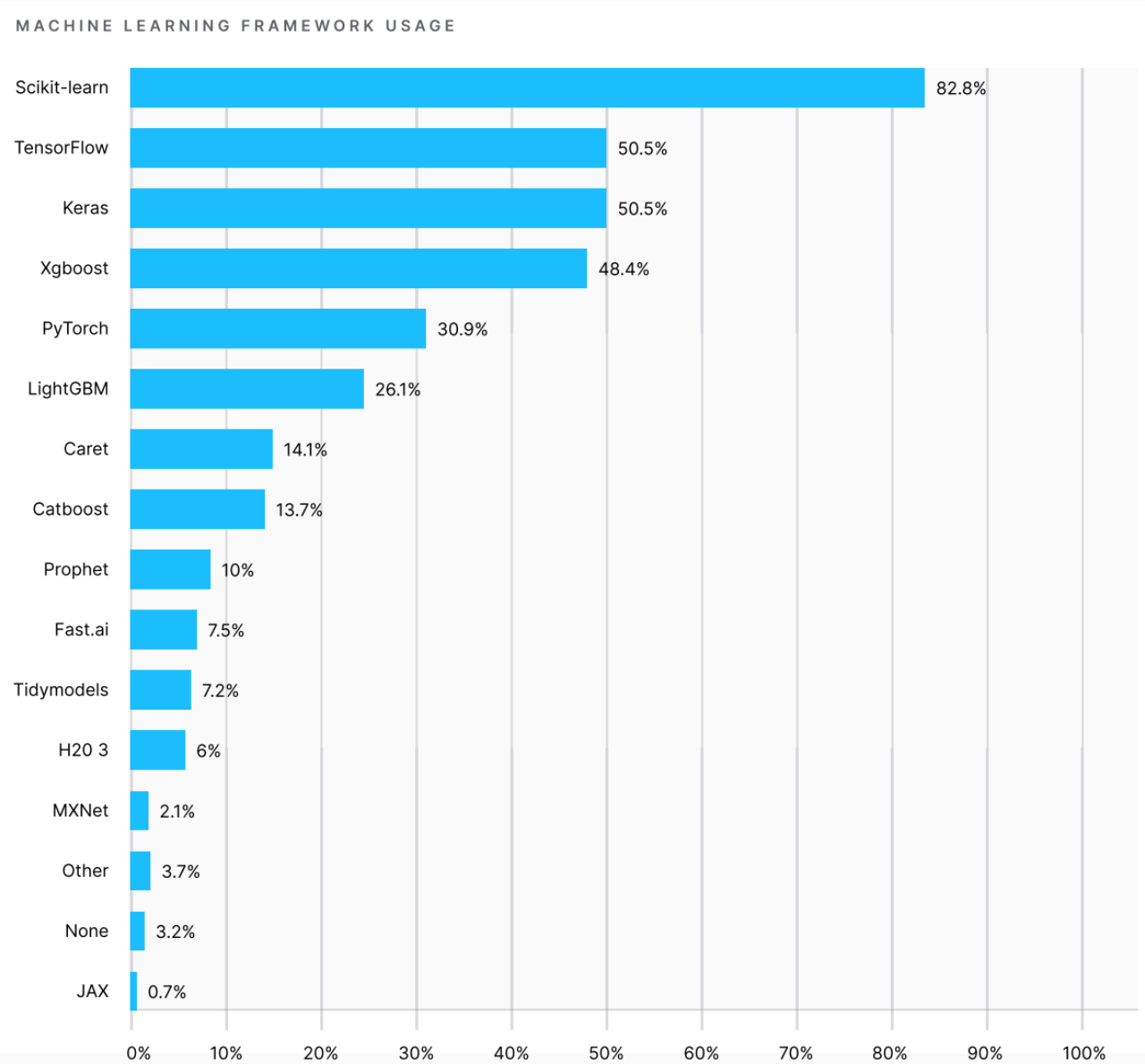
Key findings: Salary by Country



Key findings: Methods and Algorithms



Key findings: ML Frameworks



Thank You!