

Hands-on Machine Learning with Python - Module 1

Hands-on webinar

Katrien Antonio & Jonas Crevecoeur & Roel Henckaerts

[hands-on-machine-learning-Python-module-1](#) | March, 2023

Prologue

Introduction

Course

🔗 <https://github.com/katrienantonio/hands-on-machine-learning-Python-module-1>

The course repo on GitHub, where you can find the data sets, lecture sheets, Google Colab links and Python notebooks.

Us

🔗 <https://katrienantonio.github.io/> & LinkedIn profile Jonas & LinkedIn profile Roel

✉️ katrien.antonio@kuleuven.be & jonas.crevecoeur@kuleuven.be & roel.henckaerts@kuleuven.be

🎓 (Katrien) Professor in insurance data science

🎓 (Jonas) PhD in insurance data science, now data science consultant at UHasselt and KULeuven

🎓 (Roel) PhD in insurance data science, now senior data analyst at [Prophecy Labs](#)

Why this course?

The goals of this course

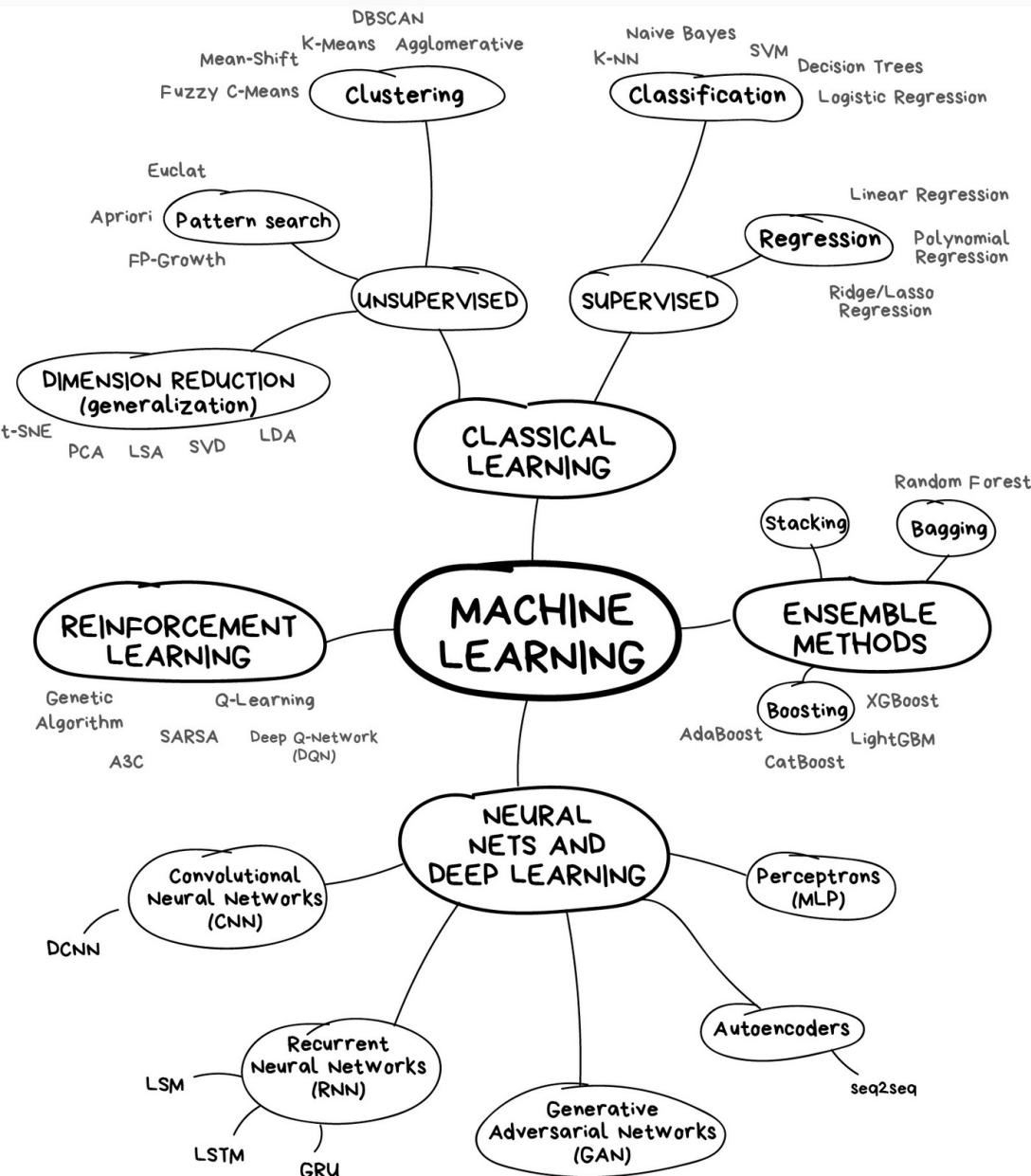
- develop practical **machine learning (ML) foundations in Python**
- **fill in the gaps** left by traditional training in actuarial science or econometrics
- focus on the use of ML methods for the **analysis of frequency + severity data**, but also **non-standard data** such as images
- **explore** a substantial range of **methods (and data types)** (from GLMs to deep learning), but - most importantly - **build foundation** so that you can explore other methods (and data types) yourself.

"In short, we will cover things that we wish someone had taught us in our undergraduate programs."

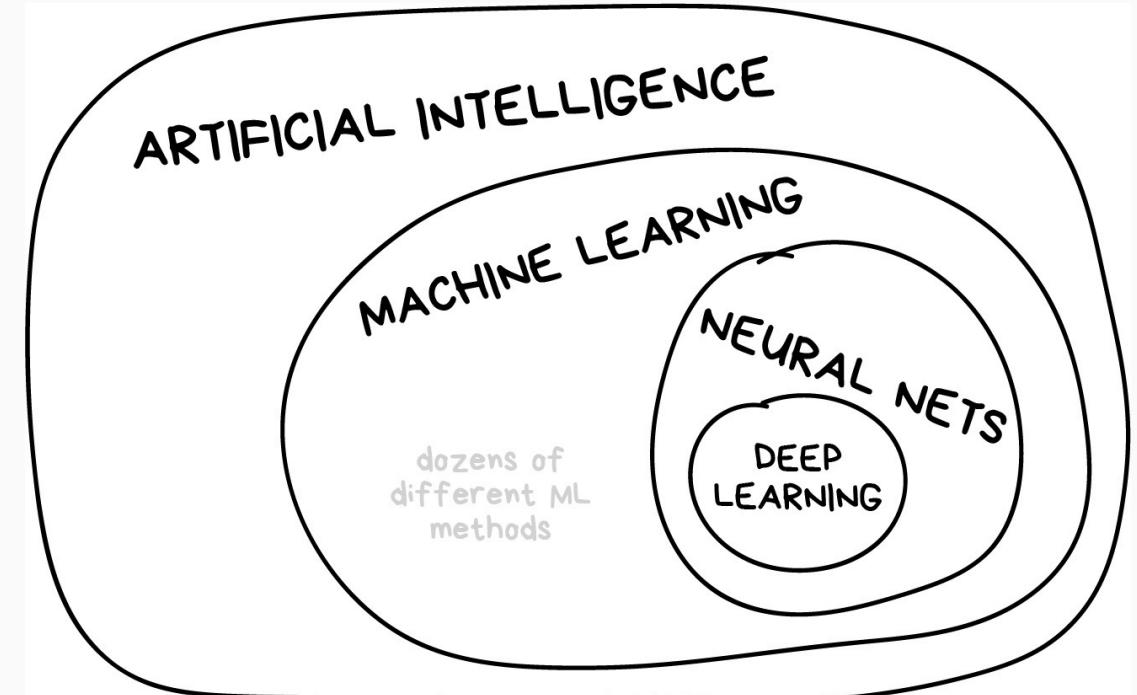
This quote is from the [Data science for economists course](#) by Grant McDermott.

Module 1's outline

- Prologue
- Knowing me, knowing you:
statistical and machine learning
 - Supervised and unsupervised learning
 - Regression and classification
 - Statistical modeling: the two cultures
- Model accuracy and loss functions
- Overfitting and bias-variance tradeoff
- Data splitting, Resampling methods
- Parameter tuning
 - with `scikit-Learn`
- Target and feature engineering
 - Data leakage
 - Pre-processing steps
 - Specifying blue-prints with `scikit-Learn`
 - Putting it all together: tuning and preprocessing via the pipeline module
- Regression models
 - Creating models in Python
 - GLMs with `statsmodels`
 - GAMs with `statsmodels`
 - Regularized (G)LMs with `scikit-learn` and `statsmodels`.

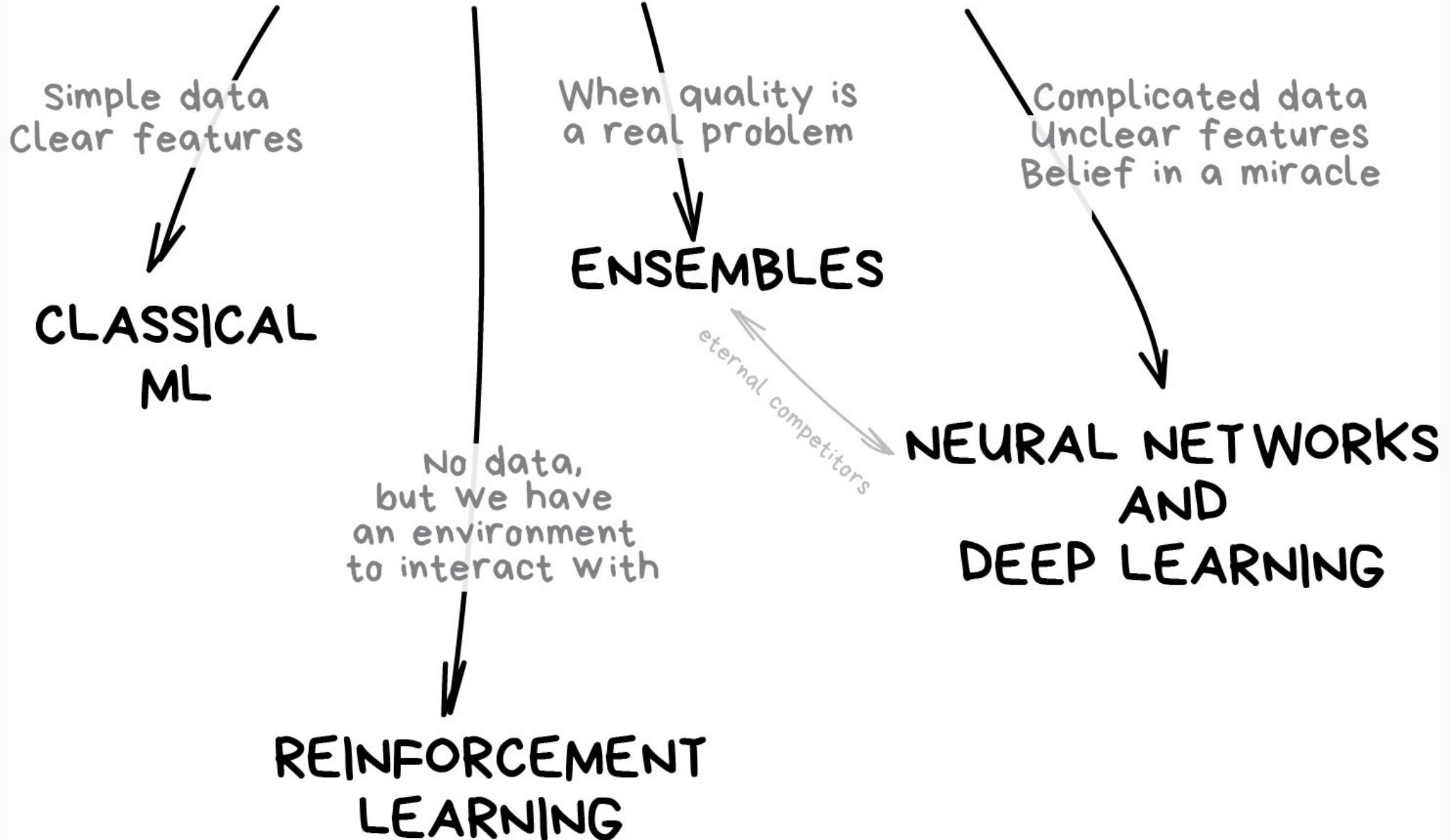


Some roadmaps to explore the ML landscape...



Source: Machine Learning for Everyone In simple words. With real-world examples. Yes, again.

THE MAIN TYPES OF MACHINE LEARNING



Knowing me, knowing you: statistical and machine learning

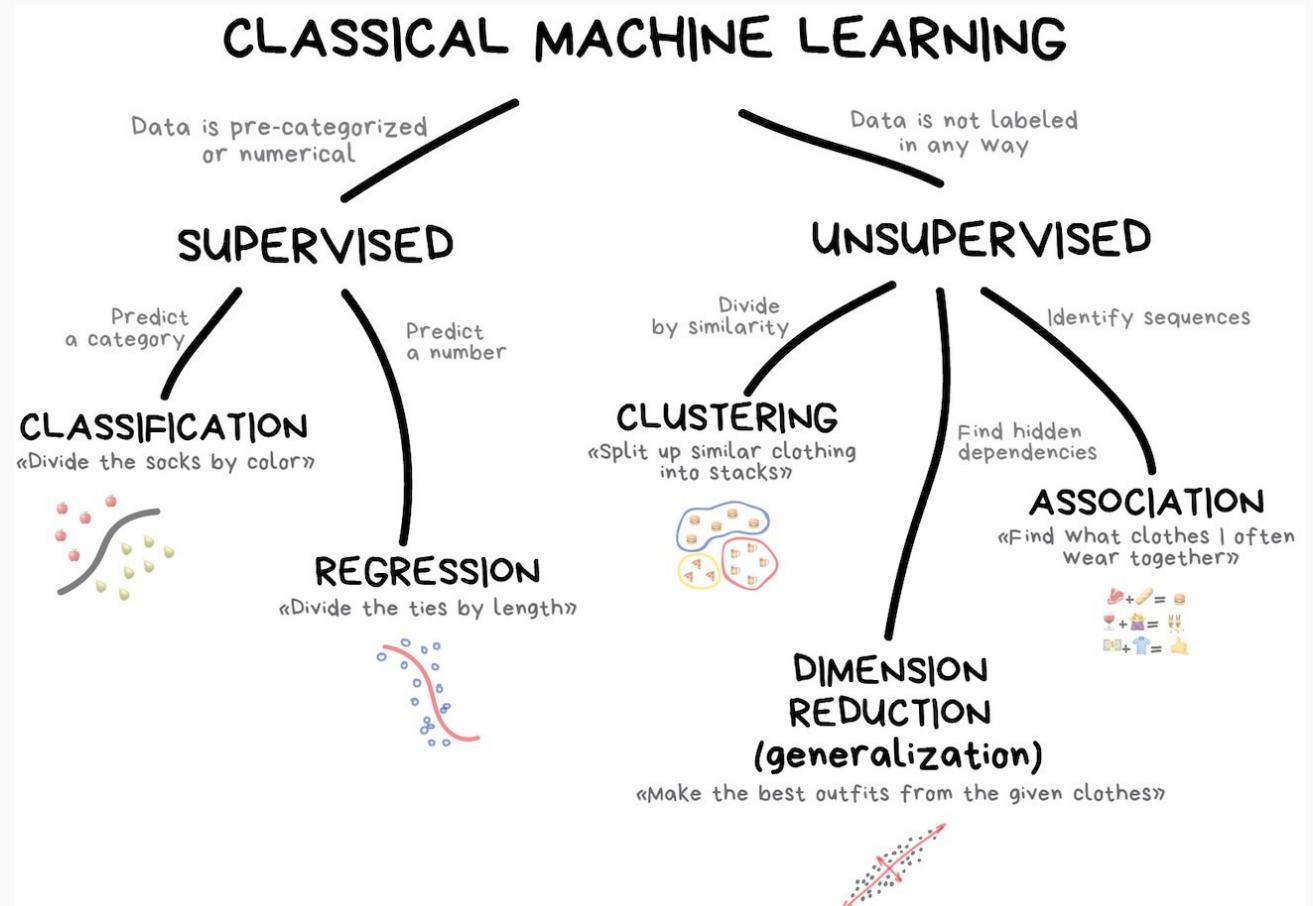
Supervised learning

Supervised learning builds ("learns") a model f (*the Signal*) such that the *outcome or target* Y can be written as

$$Y = f(x_1, \dots, x_p) + \epsilon$$

with *features* x_1, \dots, x_p and error term ϵ (*the Noise*).

Supervised learners construct **predictive models**.



Picture taken from [Machine Learning for Everyone](#). In simple words. With real-world examples. Yes, again

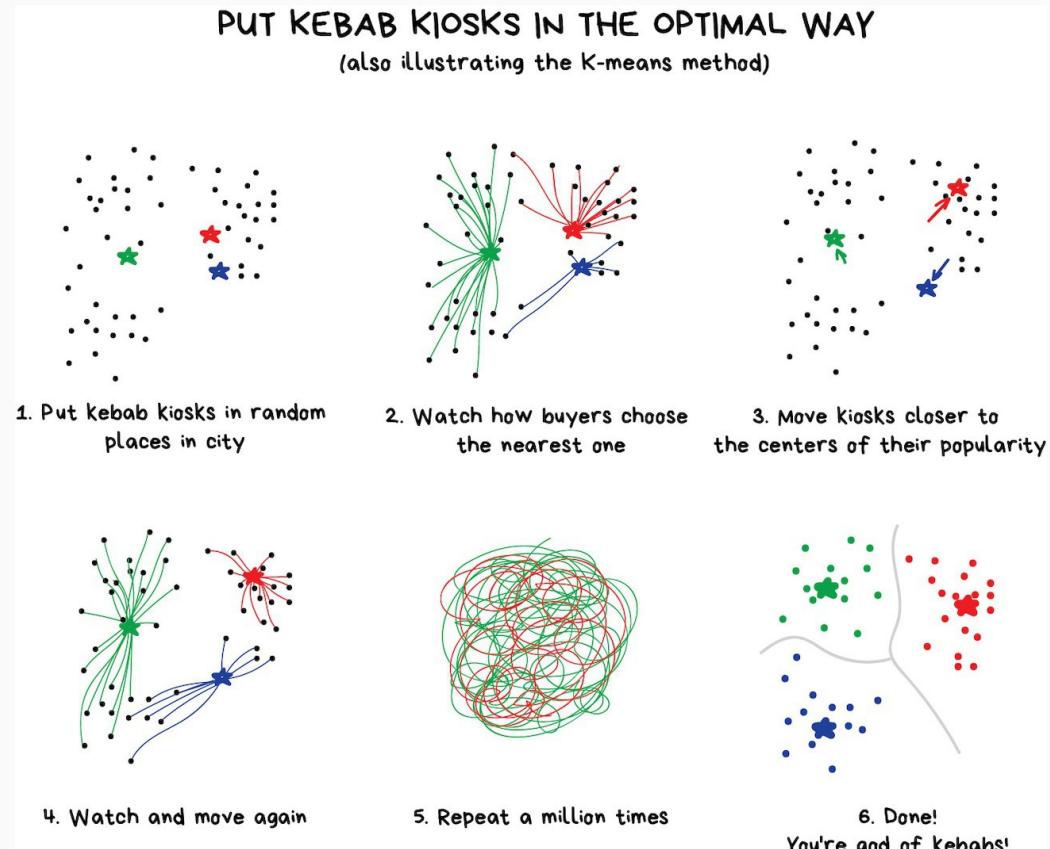
Unsupervised learning

With unsupervised learning there is **NO** outcome or target Y , only the feature vector $x = (x_1, \dots, x_p)$.

Let n denote the sample size and p the number of features.

Then, \mathbf{X} is the $n \times p$ matrix of features, with $x_{i,j}$ observation i on variable or feature j .

Unsupervised learners construct **descriptive models**, without any supervising output, letting the data "speak for itself".



Picture taken from Machine Learning for Everyone. In simple words. With real-world examples. Yes, again

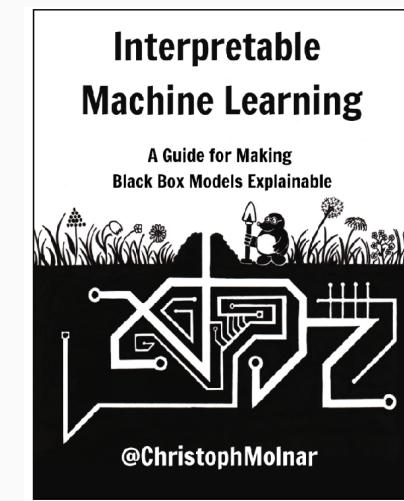
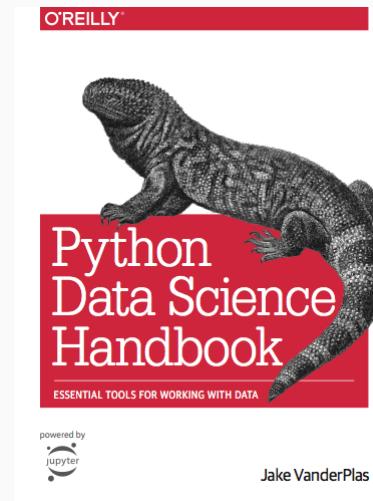
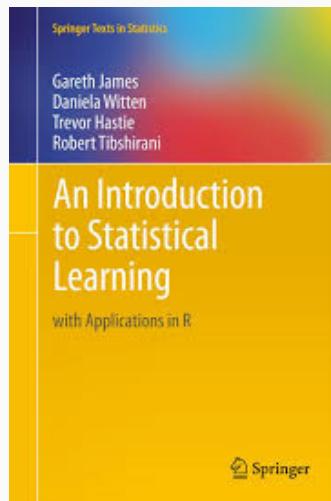
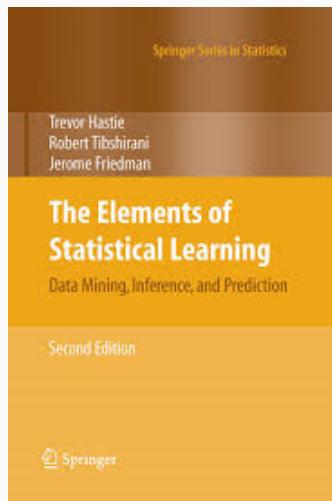
What's in a name?

Machine learning constructs algorithms that learn from data.

Statistical learning emphasizes statistical models and the assessment of uncertainty.

Data science applies mathematics, statistics, machine learning, engineering, etc. to extract knowledge from data.

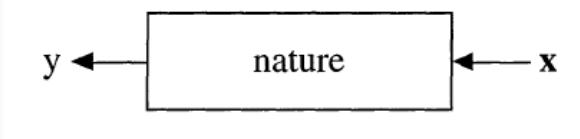
"Data Science is statistics on a Mac 🍏."



Source: Brandon M. Greenwell on [Introduction to Machine Learning in R](#).

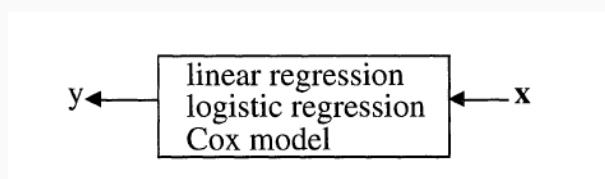
Statistical modeling: the two cultures

Consider a vector of input variables \mathbf{x} , being transformed into some vector of response variables \mathbf{y} via a black box algorithm.



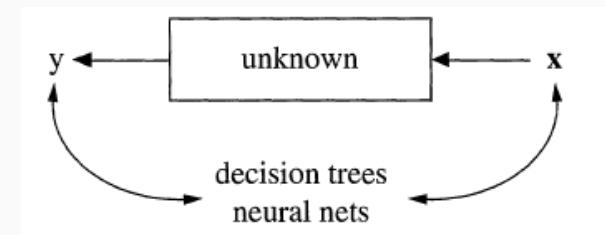
Statistical learning or data modeling culture

- assume statistical model, estimate parameter values
- validate with goodness-of-fit tests and residual inspection



Machine learning or algo modeling culture

- inside of the box is complex and unknown
- find algorithm $f(\mathbf{x})$ to predict \mathbf{y}
- measure performance by predictive accuracy



Source: Breiman (2001, Statistical Science) on *Statistical modeling: the two cultures*.

Newspeak from the two cultures

	Statistical learning	Machine learning
origin	statistics	computer science
$f(x)$	model	algorithm
emphasis	interpretability, precision and uncertainty	large scale applicability, prediction accuracy
jargon	parameters, estimation	weights, learning
CI	uncertainty of parameters	no notion of uncertainty
assumptions	explicit a priori assumption	no prior assumption, learn from the data

Source: read the blog [Why a mathematician, statistician and machine learner solve the same problem differently](#)



As discussed in the lecture, many problems in ML can be approached as a **regression**, **classification** or **clustering** problem.

Your turn

Q: consider the following **three problem settings** and **label them** as regression, classification or clustering.

1. In disability insurance: how do disability rates depend on the state of the economy (e.g. GDP)?
2. In MTPL insurance: predict whether a claim is attritional or large, *in casu* a claim that exceeds the threshold of 100 000 EUR?
3. How can we group customers based on the insurance products they bought from the company?

Exploring the Google Colab environment



We will now visit the Google Colab for the first time and explore the packages and data sets that will be used in today's session.

We will use the Ames Iowa housing data. There are 2,930 properties in the data set.

The `Sale_Price` (target or response) was recorded along with 80 predictors, including:

- location (e.g. neighborhood) and lot information
- house components (garage, fireplace, pool, porch, etc.)
- general assessments such as overall quality and condition
- number of bedrooms, baths, and so on.

More details in [De Cock \(2011, Journal of Statistics Education\)](#).

The raw data are at <https://vincentarelbundock.github.io/Rdatasets>.

Model accuracy and loss functions

Predictive modeling

How to use the observed data to learn or to estimate the unknown $f(\cdot)$?

$$y = f(x_1, x_2, \dots, x_p) + \epsilon.$$

How do I **estimate** $f(\cdot)$ - one way to phrase *all questions* that underly statistical & machine learning.

Take-aways

- main reasons we want to **learn about** $f(\cdot)$

prediction

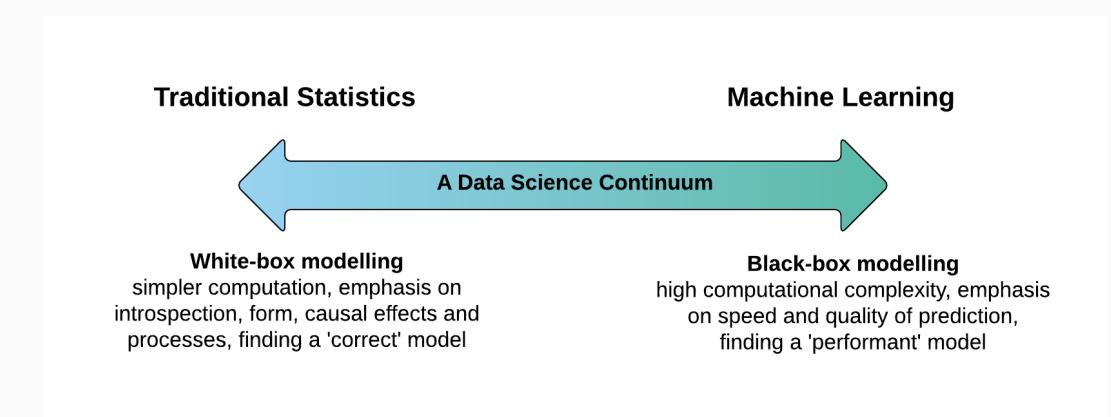
predict the target y as $\hat{f}(x)$

 - as black box setting?

inference

how does target y depend on features x ?

 - as white box setting?



Prediction errors

Why we're stuck with **irreducible error**

assume \hat{f} and x given, then

$$\begin{aligned} E[\{\mathbf{y} - \hat{\mathbf{y}}\}^2] &= E\left[\left\{\mathbf{f}(x) + \epsilon - \hat{\mathbf{f}}(x)\right\}^2\right] \\ &= \underbrace{\left[\mathbf{f}(x) - \hat{\mathbf{f}}(x)\right]^2}_{\text{Reducible}} + \underbrace{\text{Var}(\epsilon)}_{\text{Irreducible}} \end{aligned}$$

In **less math**:

- if ϵ exists, then x cannot perfectly explain y
- so even if $\hat{f} = f$, we still have irreducible error.

Thus, to form our **best predictors**, we will **minimize reducible error**.

Model accuracy

We assess **model** or **predictive accuracy** by evaluating how well predictions actually match observed data.

Use **loss functions**, i.e. metrics that compare predicted values to actual values.

Regression, use e.g. the **Mean Squared Error (MSE)**

$$\frac{1}{n} \sum_{i=1}^n (\textcolor{blue}{y}_i - \hat{f}(\textcolor{red}{x}_i))^2,$$

Recall: $\textcolor{blue}{y}_i - \hat{y}_i = \textcolor{blue}{y}_i - \hat{f}(\textcolor{red}{x}_i)$ is the prediction error.

Objective \odot : minimize!

Classification, use e.g. the **cross-entropy** or **log loss**

$$-\frac{1}{n} \sum_{i=1}^n (\textcolor{blue}{y}_i \cdot \log(p_i) + (1 - \textcolor{blue}{y}_i) \cdot \log(1 - p_i)).$$

Objective \odot : minimize!

Many other useful loss functions (e.g. deviance in regression, Gini index in classification).

Take-away 

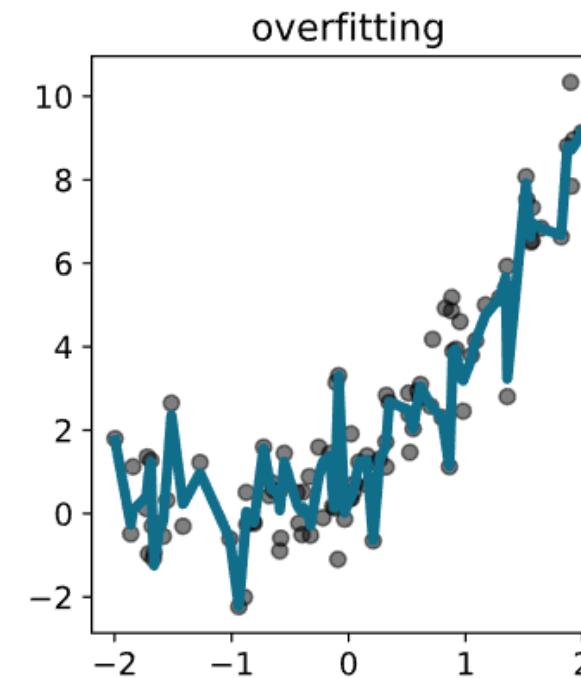
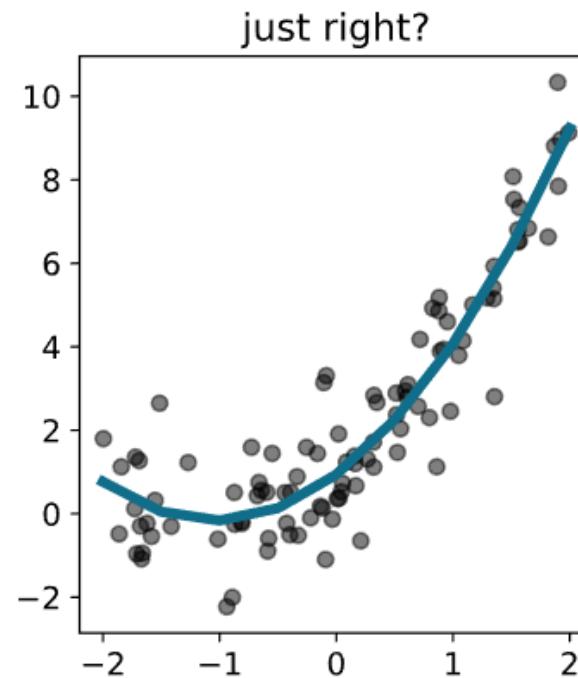
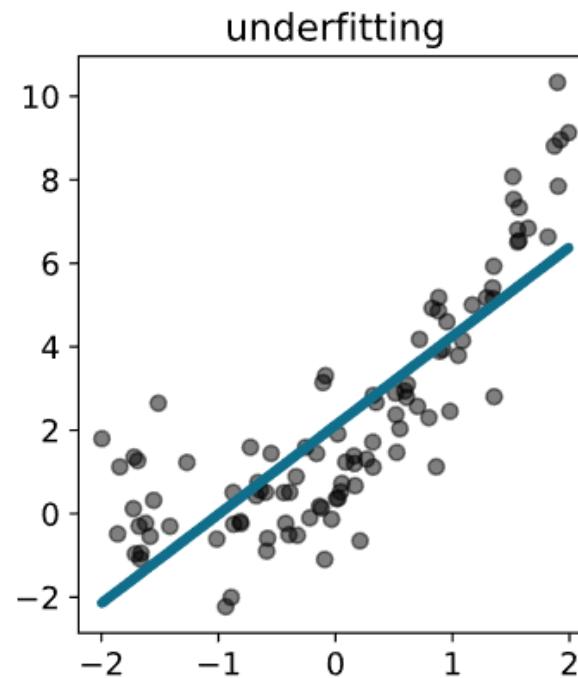
- a loss function emphasizes certain types of errors over others → pick a meaningful one!

Overfitting and bias-variance trade off

Overfitting

The [Signal and the Noise](#) discussion!

Which of the following three models (in green-blue-ish) will best generalize to new data?



Inspired by Brandon Greenwell's [Introduction to Machine Learning in R](#).

Overfitting (cont.)

With a small training error, but large test error, the model is **overfitting** or working too hard!

The expected value of the **test MSE**:

$$E\left(\textcolor{orange}{y}_0 - \hat{\textcolor{blue}{f}}(\textcolor{red}{x}_0)\right)^2 = \text{Var}(\hat{\textcolor{blue}{f}}(\textcolor{red}{x}_0)) + [\text{Bias}(\hat{\textcolor{blue}{f}}(\textcolor{red}{x}_0))]^2 + \text{Var}(\epsilon).$$

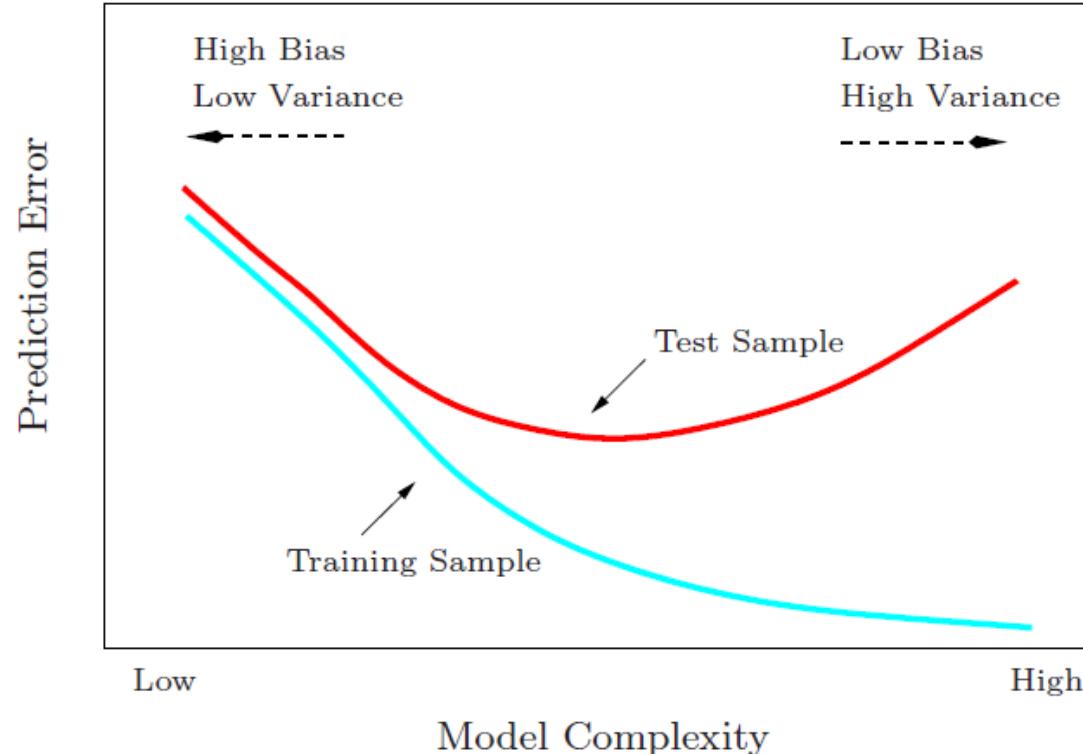
In general - with more flexible methods

- variance  and bias 
- their relative rate of change determines whether the test error increases or decreases

Take-aways

- U-shape curves of **test MSE** w.r.t model flexibility
- the **bias-variance tradeoff** is central to quality prediction.

Bias-variance trade off



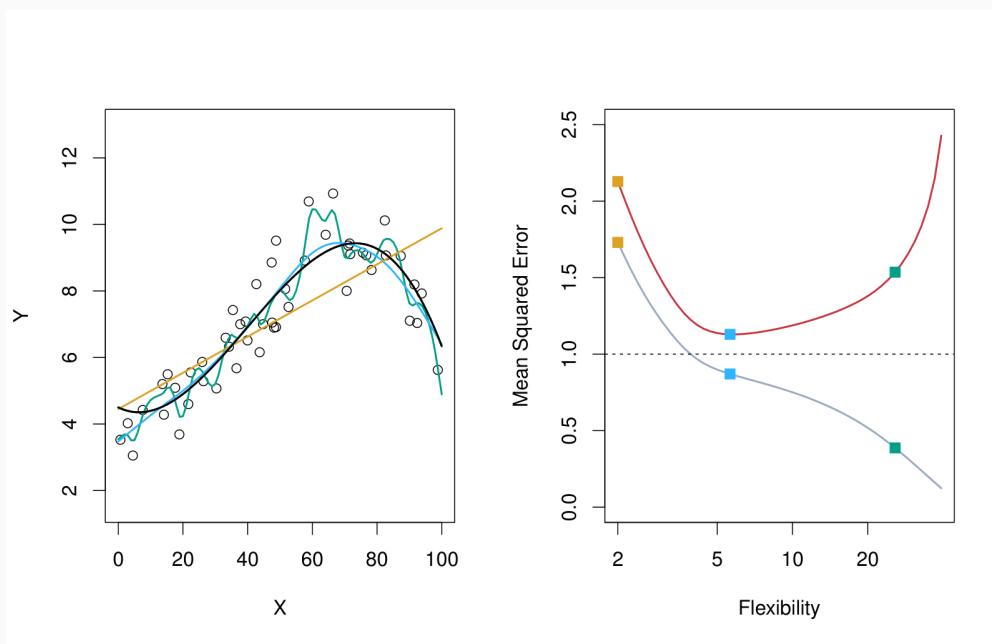
Source: James et al. (2013) on <http://faculty.marshall.usc.edu/gareth-james/ISL/>.



Your turn

Data are generated from: $y = f(x) + \epsilon$, with the black curve as the true f . The orange (linear regression), blue (smoothing splines) and green (smoothing splines) curves are three estimates for f , with increasing level of complexity.

Q: which model do you prefer (orange, blue, green) for each of the following examples? Why?



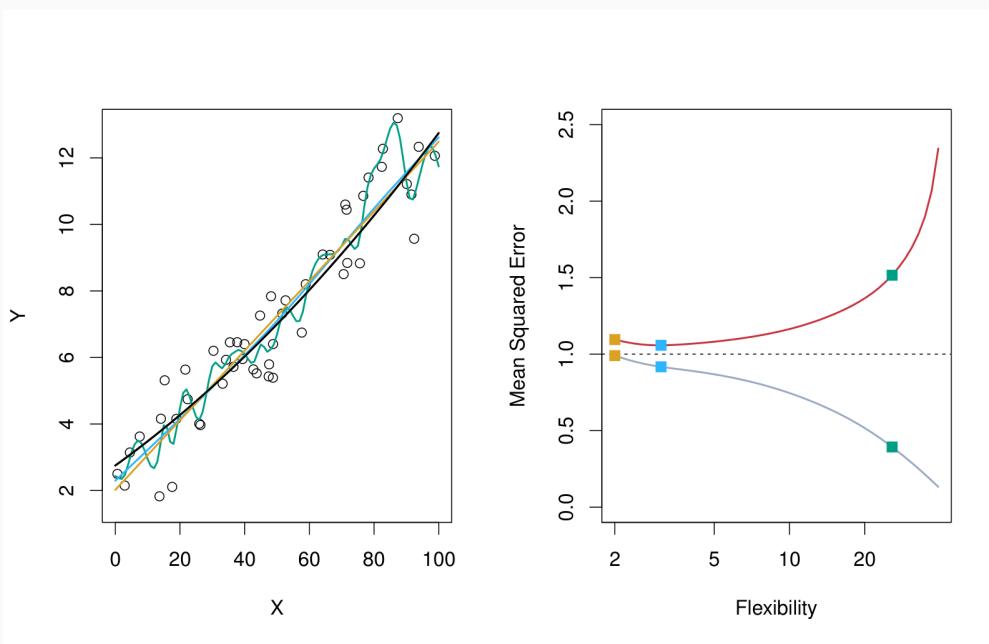
Example from James et al. (2013) on <http://faculty.marshall.usc.edu/gareth-james/ISL/>.



Your turn

Data are generated from: $y = f(x) + \epsilon$, with the black curve as the true f . The orange (linear regression), blue (smoothing splines) and green (smoothing splines) curves are three estimates for f , with increasing level of complexity.

Q: which model do you prefer (orange, blue, green) for each of the following examples? Why?



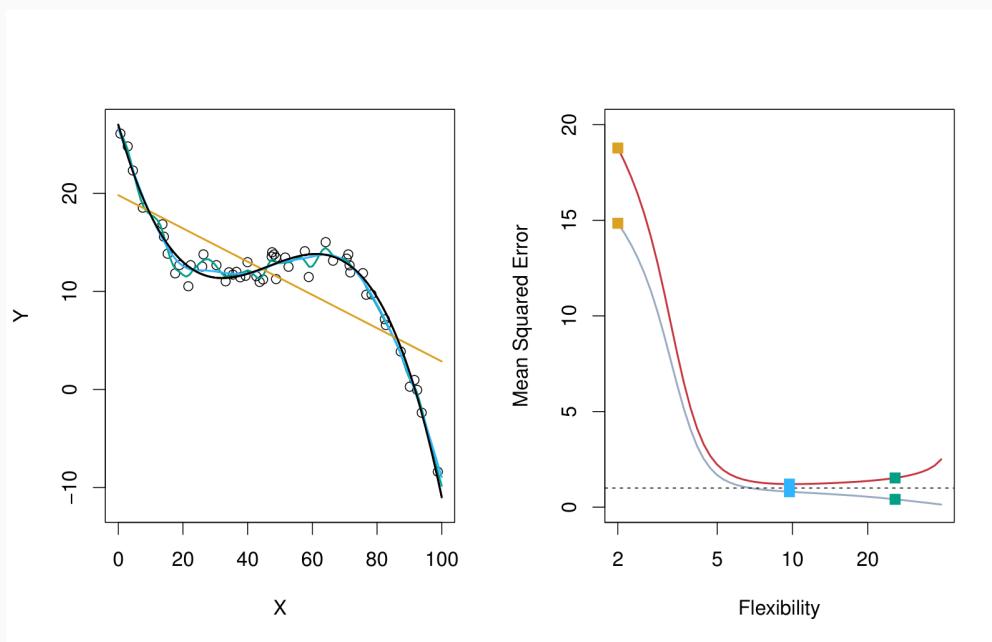
Example from James et al. (2013) on <http://faculty.marshall.usc.edu/gareth-james/ISL/>.



Your turn

Data are generated from: $y = f(x) + \epsilon$, with the black curve as the true f . The orange (linear regression), blue (smoothing splines) and green (smoothing splines) curves are three estimates for f , with increasing level of complexity.

Q: which model do you prefer (orange, blue, green) for each of the following examples? Why?



Example from James et al. (2013) on <http://faculty.marshall.usc.edu/gareth-james/ISL/>.



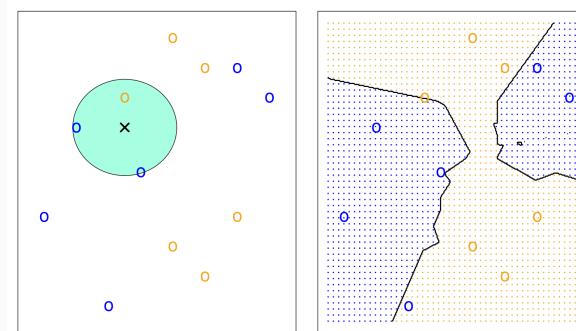
Your turn

The K -nearest neighbors (KNN) classifier

- take the K observations in the training data set that are 'closest' to test observation $\textcolor{red}{x}_0$, calculate

$$\Pr(\textcolor{orange}{Y} = j | \textcolor{red}{X} = \textcolor{red}{x}_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} \mathbb{I}(\textcolor{orange}{y}_i = j).$$

- KNN then assigns the test observation $\textcolor{red}{x}_0$ to the class j with the highest probability, e.g. with $K=3$ (from James et al., 2013)



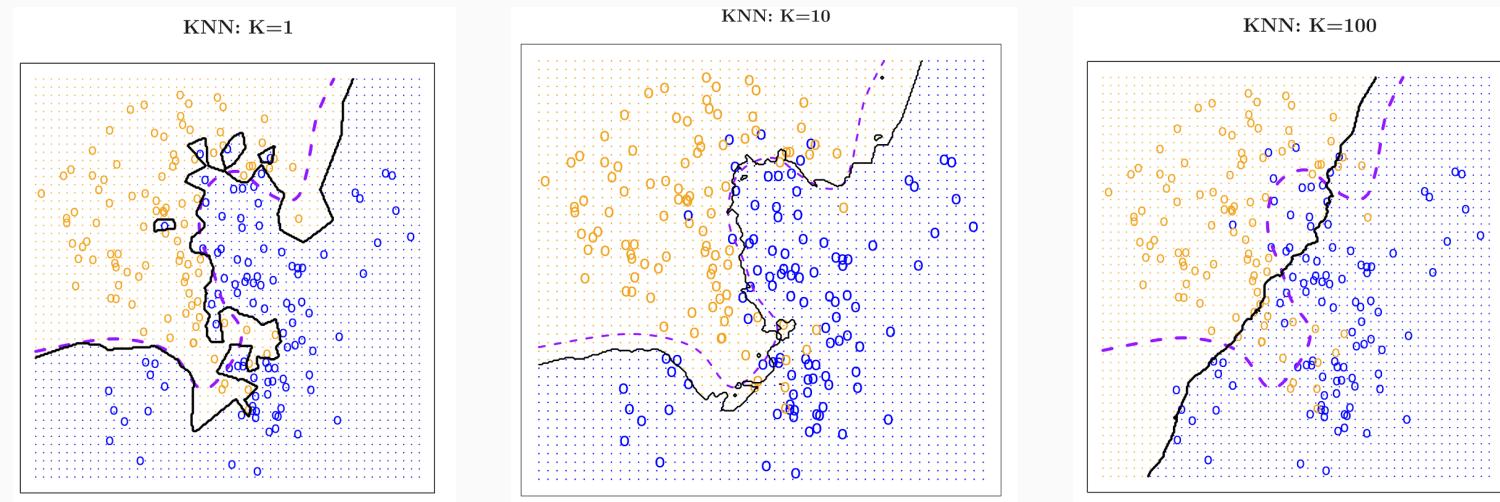
Q: is KNN a supervised learning or unsupervised learning method? Discuss.



Your turn

The K -nearest neighbors (KNN) classifier (cont.)

Now compare KNN with K equals 1, 10 and 100.



Q: which classifier do you prefer? Which of these classifiers is under-fitting, which one is over-fitting?

Data splitting and resampling methods with scikit learn

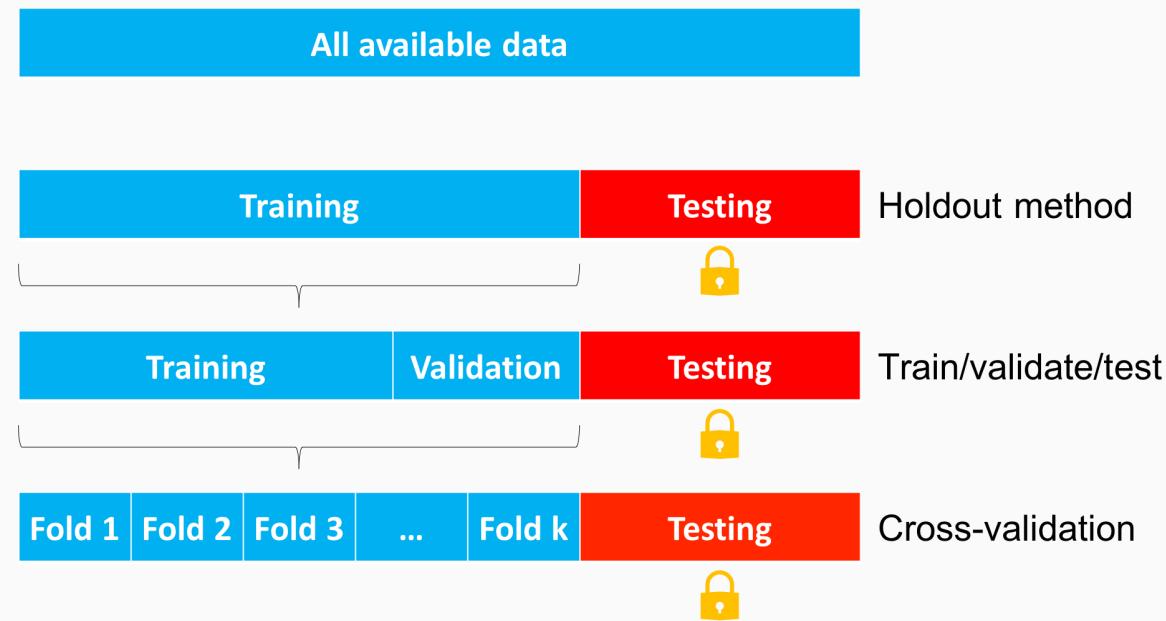
Data splitting

We fit our model on past data $\{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_n, \mathbf{y}_n)\}$ and get \hat{f} .

What we want: how does our model **generalize** to new, unseen data $(\mathbf{x}_0, \mathbf{y}_0)$, or: is $\hat{f}(\mathbf{x}_0)$ close to \mathbf{y}_0 ?

Training set

- to develop, to train,
to tune, to compare
different settings, ...



Test set

- to obtain unbiased
estimate of final
model's
performance.

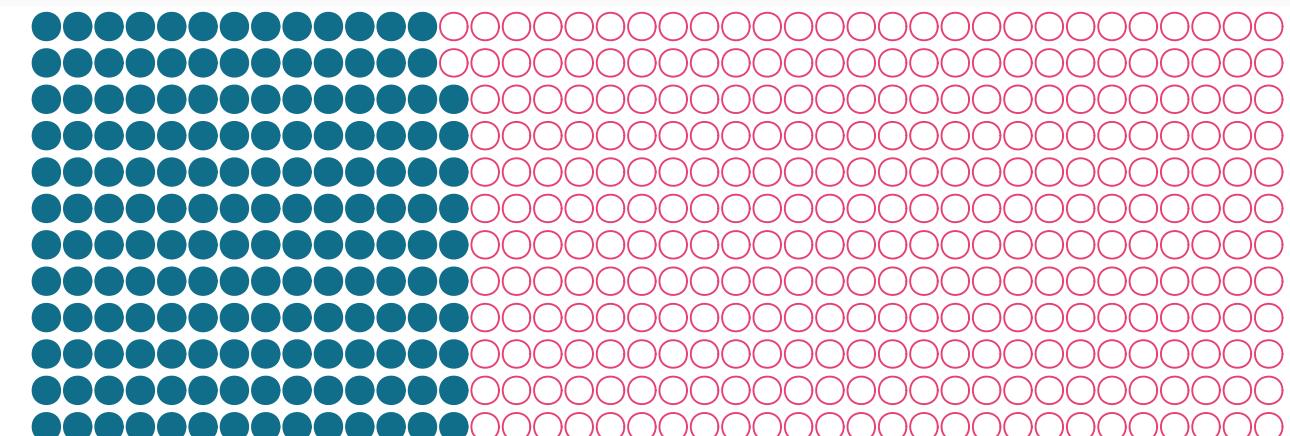
Picture taken from [Introduction to Machine Learning in R](#).

Resampling methods

In [Data splitting](#), we discussed *training* and *test* set. Let's now dive deeper into *resampling* methods.

Validation set (visual inspired by [Ed Rubin's course](#))

- we hold out a subset of the training data (e.g. 30%) and then evaluate the model on this held out validation set
- calculate the loss function on this validation set, as approximation of the true test error
-  high variability + inefficient use of data
- e.g., a **validation set (30%)** and **training set (70%)**.

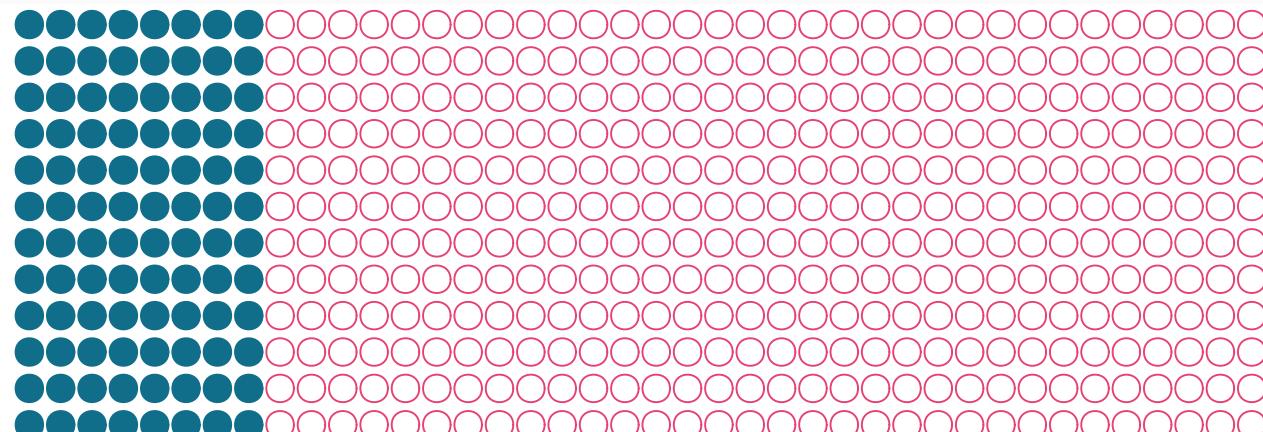


Resampling methods (cont.)

In [Data splitting](#), we discussed *training* and *test* set. Let's now dive deeper into *resampling* methods.

***k* fold cross validation** (visual inspired by [Ed Rubin's course](#))

- divide training data into k equally sized groups (e.g. **group 1** on the picture)
- iterate over the k groups, treating each as validation set once (and train model on the other $k-1$ groups) (e.g. get **MSE₁** corresponding to fold 1)
- average the folds' loss to estimate the true test error
-  greater accuracy (compared to validation set).

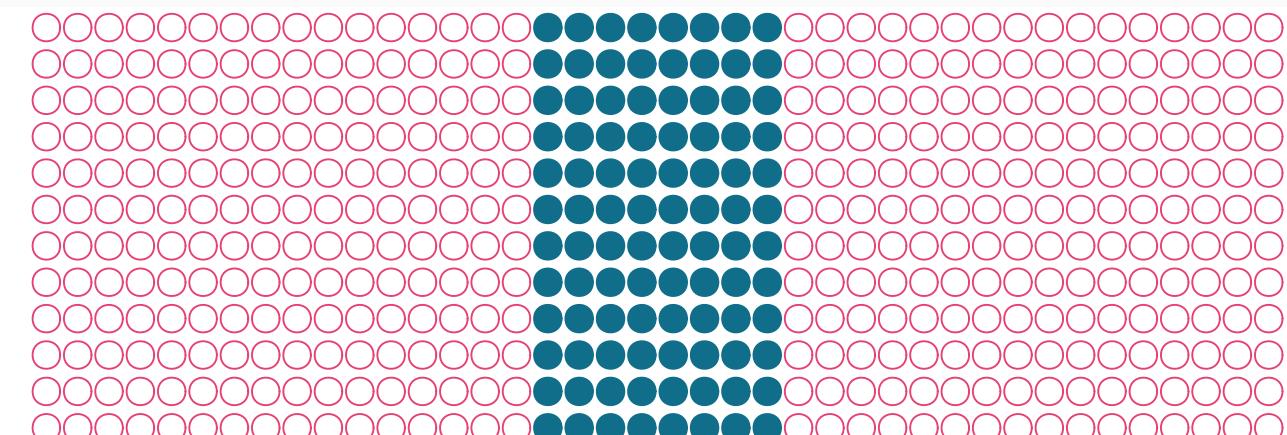


Resampling methods (cont.)

In [Data splitting](#), we discussed *training* and *test* set. Let's now dive deeper into *resampling* methods.

***k* fold cross validation** (visual inspired by [Ed Rubin's course](#))

- divide training data into k equally sized groups (e.g. **group 1** on the picture)
- iterate over the k groups, treating each as validation set once (and train model on the other $k-1$ groups) (e.g. get **MSE₁** corresponding to fold 1)
- average the folds' loss to estimate the true test error
-  greater accuracy (compared to validation set).



Resampling methods (cont.)

In [Data splitting](#), we discussed *training* and *test* set. Let's now dive deeper into *resampling* methods.

***k* fold cross validation** (picture from [Boehmke & Greenwell](#))

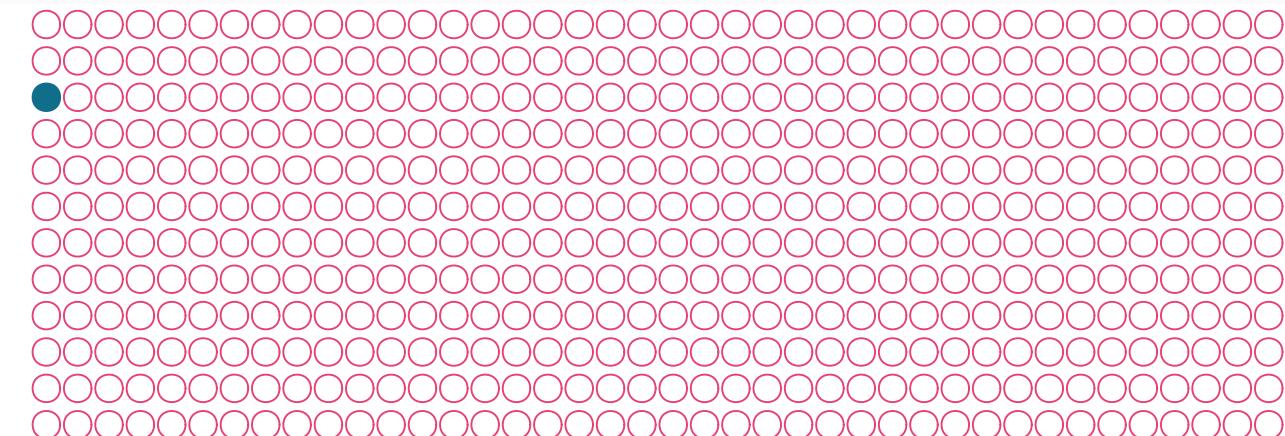


Resampling methods (cont.)

In [Data splitting](#), we discussed *training* and *test* set. Let's now dive deeper into *resampling* methods.

Leave-one-out cross validation (visual inspired by [Ed Rubin's course](#))

- each observation takes a turn as the validation set (e.g. get **MSE₃**)
- other $n-1$ observations are the training set
- average the folds' loss to estimate the true test error
- 🗺 very computationally demanding.



Working with pandas and scikit-learn



We will now explore basic instructions to:

- explore the structure of the data set with instructions from `{pandas}`
- visualize the data using `matplotlib` and `seaborn`
- split a data set into a training and test set with instructions from `{numpy}` and `{scikit learn}`.

Here:

- `{pandas}` is a package for data analysis and the manipulation of tabular data; the name is derived from the term **panel data**
- `{numpy}` is a package for scientific computation
- `{scikit-learn}` is for machine learning in Python, with functions for a.o. preprocessing, model selection, regression and classification.

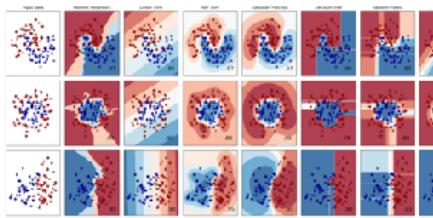
- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.

Algorithms: SVM, nearest neighbors, random forest, and more...

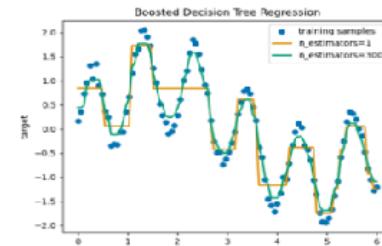
[Examples](#)

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, nearest neighbors, random forest, and more...

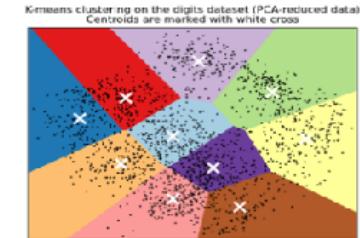
[Examples](#)

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, and more...

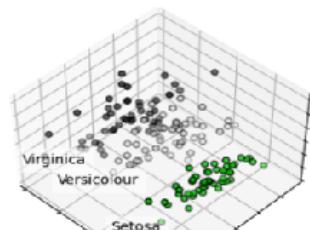
[Examples](#)

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency

Algorithms: PCA, feature selection, non-negative matrix factorization, and more...

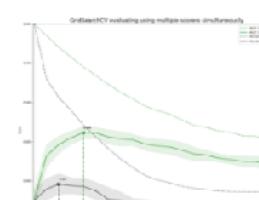


Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning

Algorithms: grid search, cross validation, metrics, and more...

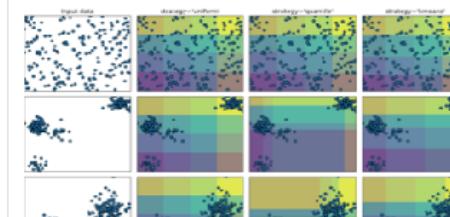


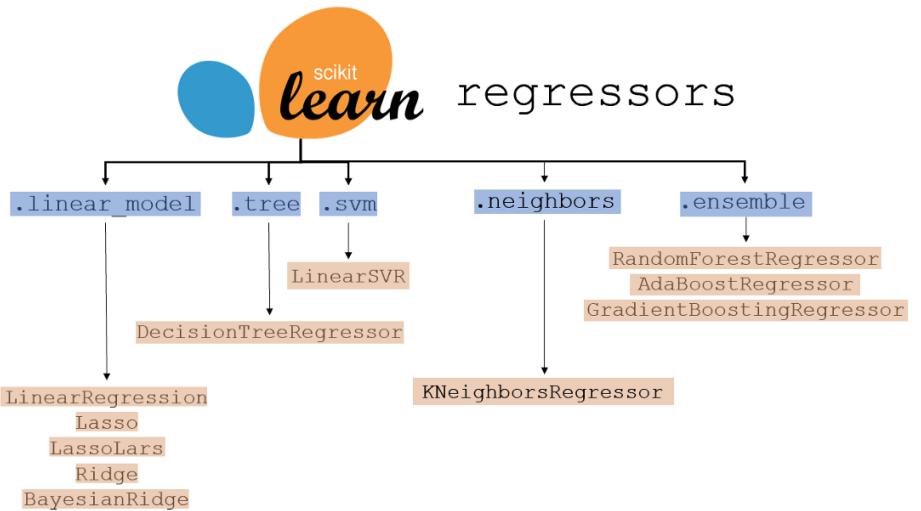
Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.

Algorithms: preprocessing, feature extraction, and more...



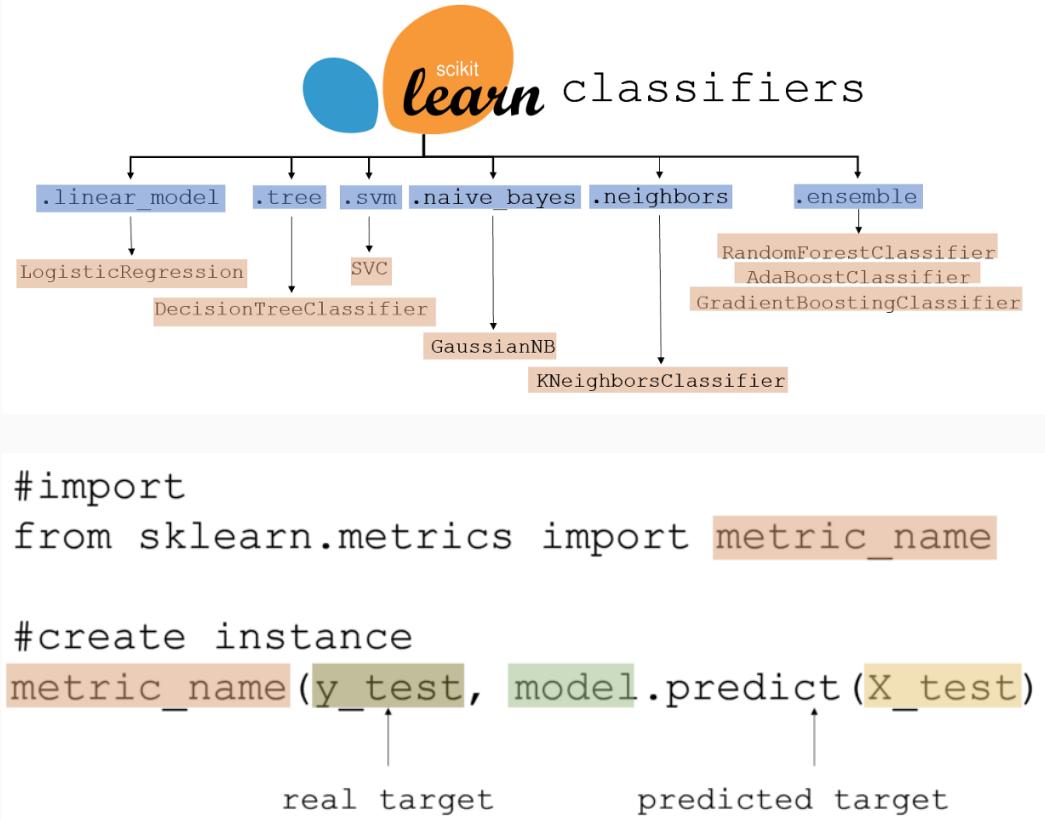


```

import
from sklearn.branch import model_name

#create instance
model = model_name()

#fit model
model.fit(X_train, y_train)
  
```



Pictures taken from [KD Nuggets ultimate scikit-learn ML cheatsheet](#).

Resampling methods in scikit-learn



We return to the Google Colab and set up 5-fold cross validation using the `scikit-learn` library.

Parameter tuning with scikit-learn

Tuning parameters

Finding the optimal level of flexibility highlights the bias-variance tradeoff.

Bias : the error that comes from inaccurately estimating \hat{f} .

Variance : the amount \hat{f} would change with a different training sample.

Take-aways  : high variance models more prone to overfitting

- use **resampling methods** to reduce this risk
- hyperparameters (or *tuning parameters*) control complexity, and thus the bias-variance trade-off
- identify their optimal setting, e.g. with a *grid search*
- no analytic expression for these hyperparameters.

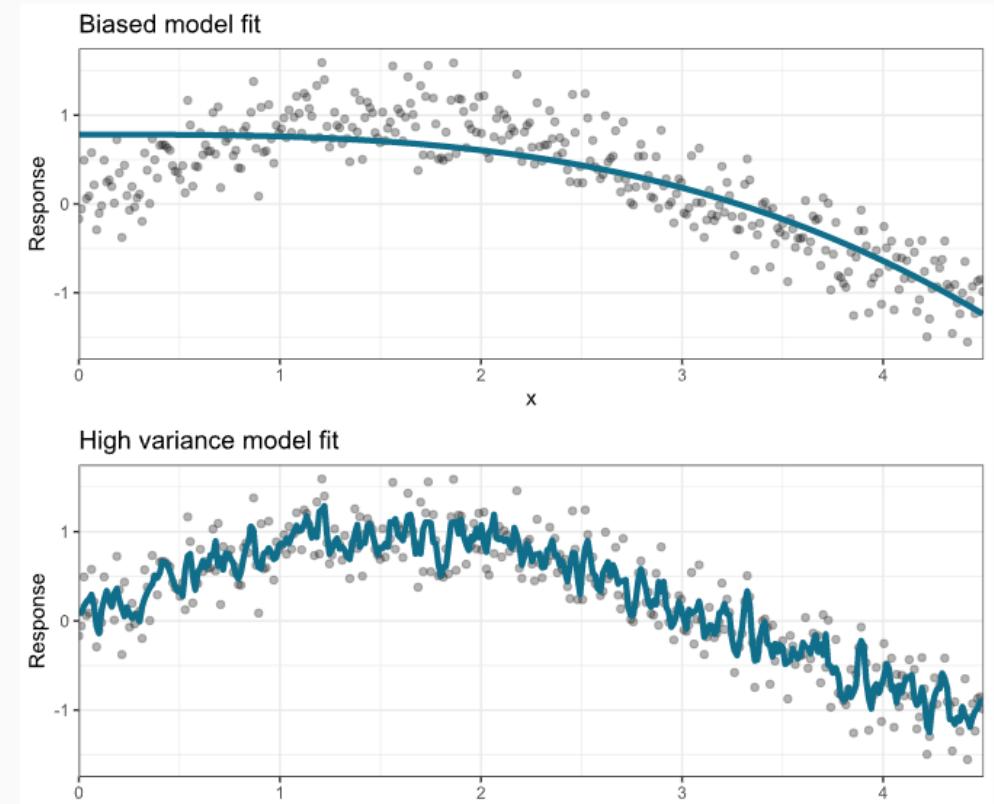
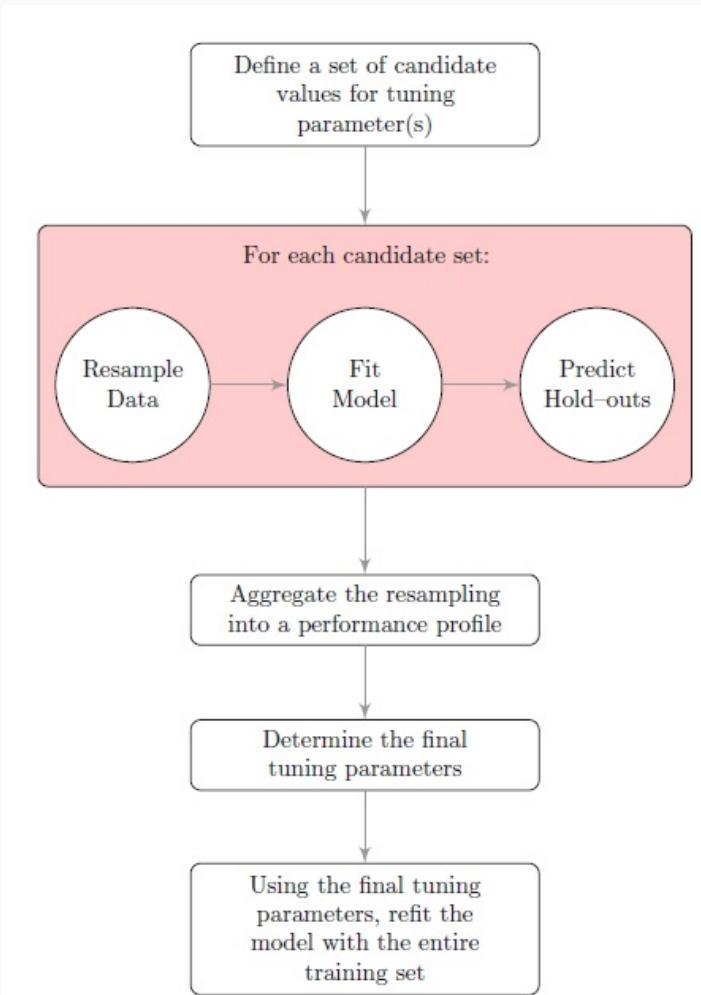


Illustration and code from Boehmke & Greenwell (2019, Chapter 2) on [Hands-on machine learning with R](#).

Tuning parameters via grid search



Model training & validation phase

- define a set of candidate values (a *grid*)
- assess model utility across the candidates (use clever *resampling*)
- choose the optimal settings (optimize *loss*)
- refit the model on entire training data with final tuning parameters
- evaluate performance of the model on the test data (under).

Model selection

- repeat the above steps for different models
- compare performance of these models that will generalize to new data (via test data, under).

Flow chart from Kuhn & Johnson (2013) on *Applied predictive modeling*.

Putting it all together

During the tuning process we inspect plots like the one on the right.

Take-aways

Less is more:

- we prefer simple over more complex
- choose tuning parameters based on the numerically optimal value **OR**
- choose a simpler model that is within a certain tolerance of the numerically best value
- use the '**one-standard-error**' rule.

With the selected tuning parameters, we refit the model on the complete training set and use it to predict the test set (under ).

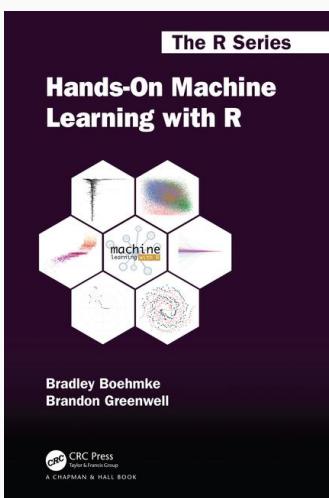
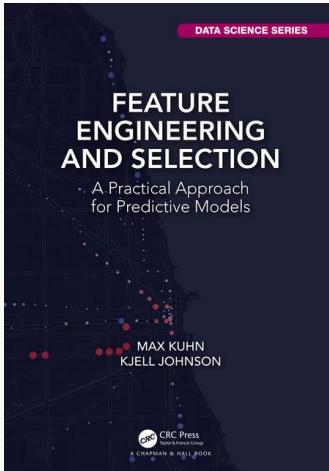
Tuning parameters with scikit-learn



Let's return to our Colab.

Target and feature engineering: data pre-processing steps

What is feature engineering?



Feature engineering:

- applies **pre-processing steps** to predictor (features) variables
- **creates new input features** from your existing ones (e.g. network features derived from a social network in a fraud detection model).

Target engineering:

- transforms the response variable (or target) to improve the performance of a predictive model.

The goal is to **make models more effective**.

See Kuhn & Johnson (2019) on **Feature Engineering and Selection: A Practical Approach for Predictive Models** for a detailed discussion.

Take-aways : different models have different sensitivities to the type of target and feature values in the model.

Table A.1: A summary of models and some of their characteristics

Model	Allows $n < p$	Pre-processing	Interpretable	Automatic feature selection	# Tuning parameters	Robust to predictor noise	Computation time
Linear regression [†]	✗	CS, NZV, Corr	✓	✗	0	✗	✓
Partial least squares	✓	CS	✓	○	1	✗	✓
Ridge regression	✗	CS, NZV	✓	✗	1	✗	✓
Elastic net/lasso	✗	CS, NZV	✓	✓	1–2	✗	✓
Neural networks	✓	CS, NZV, Corr	✗	✗	2	✗	✗
Support vector machines	✓	CS	✗	✗	1–3	✗	✗
MARS/FDA	✓		○	✓	1–2	○	○
K -nearest neighbors	✓	CS, NZV	✗	✗	1	○	✓
Single trees	✓		○	✓	1	✓	✓
Model trees/rules [†]	✓		○	✓	1–2	✓	✓
Bagged trees	✓		✗	✓	0	✓	○
Random forest	✓		✗	○	0–1	✓	✗
Boosted trees	✓		✗	✓	3	✓	✗
Cubist [†]	✓		✗	○	2	✓	✗
Logistic regression*	✗	CS, NZV, Corr	✓	✗	0	✗	✓
{LQRM}DA*	✗	NZV	○	✗	0–2	✗	✓
Nearest shrunken centroids*	✓	NZV	○	✓	1	✗	✓
Naïve Bayes*	✓	NZV	✗	✗	0–1	○	○
C5.0*	✓		○	✓	0–3	✓	✗

[†]regression only *classification only

Symbols represent affirmative (✓), negative (✗), and somewhere in between (○)

Source: Kuhn & Johnson (2013) on Applied predictive modeling.

Feature engineering steps

Examples of common pre-processing steps:

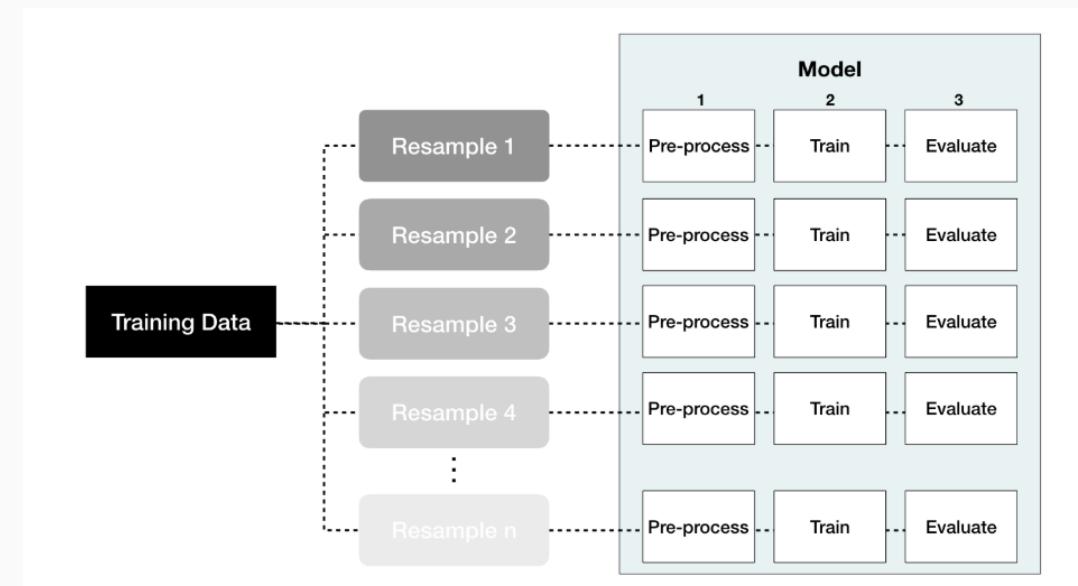
- Some models (e.g. KNN, Lasso, neural networks) require that the predictor variables are on the same scale. **Centering (C)** and **scaling (S)** the predictors can be used for this purpose.
- Other models are very sensitive to correlations between the predictors and filters or PCA signal extraction can improve the model.
- Some models find **(near) zero-variance (NZV)** predictors problematic, and these should be removed before fitting the model.
- In other cases, the data should be **encoded** in a specific way to make sure all predictors are numeric (e.g. one-hot encoding of factor variables in neural networks).
- Many models cannot cope with **missing data** so **imputation strategies** might be necessary.
- Development of new features that represent something important to the outcome.
- (add your own example here!)

This list is inspired by Max Kuhn (2019) on [Applied Machine Learning](#).

A blueprint for feature engineering

Take-aways : a proper implementation

- draft a **blueprint** of the necessary pre-processing steps, and their order
- Boehme & Greenwell (2019) suggest
 1. Filter out zero or near-zero variance features.
 2. Perform imputation if required.
 3. Normalize to resolve numeric feature skewness.
 4. Standardize (center and scale) numeric features.
 5. Perform dimension reduction (e.g., PCA) on numeric features.
 6. One-hot or dummy encode categorical features.
- avoid **data leakage** in the pre-processing steps when applied to resampled data sets!



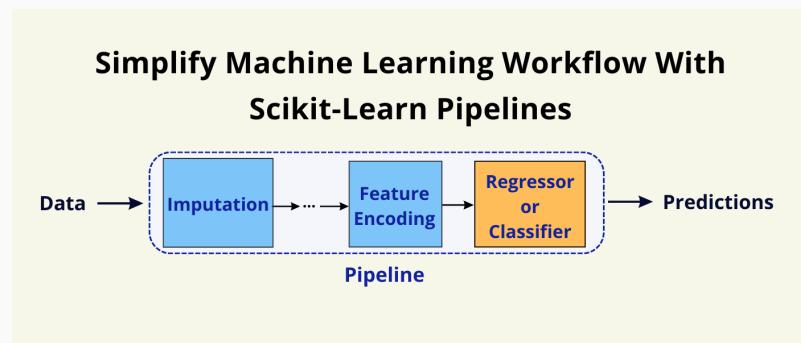
Pipelines



We'll use `pipeline` from the scikit-learn package to build a streamlined blueprint of pre-processing steps.

The main idea is to **preprocess multiple data sets** and to avoid data leakage using a single `pipeline` structure.

Before we start, keep the following **fundamentals** of `pipeline` in mind!



Creating a `pipeline` takes the following steps:

- build a pre-processing plan by setting up pipelines for distinct groups of variables
- join all the pipelines into one by specifying the family's pipeline and their variables
- add the model to the pipeline, e.g. a linear regression model
- save and load in the pipeline for future use.

Source: Mahmoud Youssef's github

Thanks!



Slides created with the R package `xaringan`.

Course material available via

 <https://github.com/katrienantonio/hands-on-machine-learning-Python-module-1>