Models

Applied Machine Learning with R

www.therbootcamp.com

@therbootcamp

January 2019

There is no free lunch

Theorem

Given a finite set \(V\) and a finite set \(S\) of real numbers, **assume that \(f:V\to S\) is chosen at random** according to uniform distribution on the set \(S^{V}\) of all possible functions from \(V\) to \(S\). For the problem of optimizing \(f\) over the set \(V\), **then no algorithm performs better than blind search.**

Wolpert & Macready, 1997, No Free Lunch Theorems for Optimization

Know your problem

Bias-variance trade-off

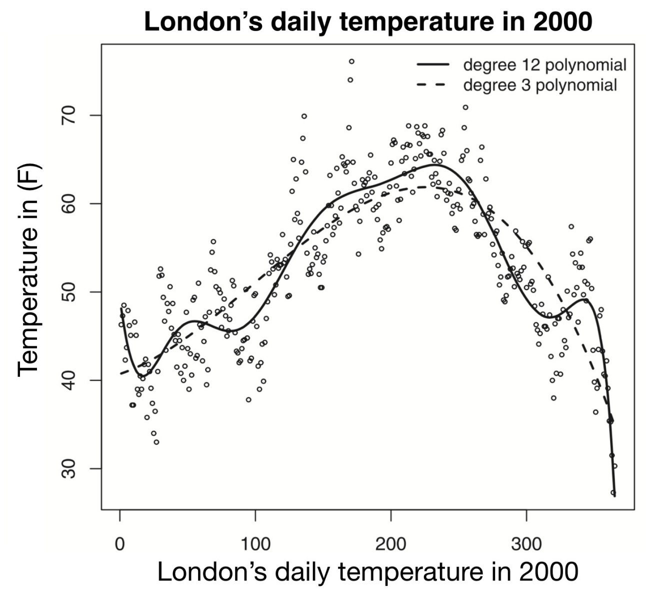
Error = **Bias** + **Variance**

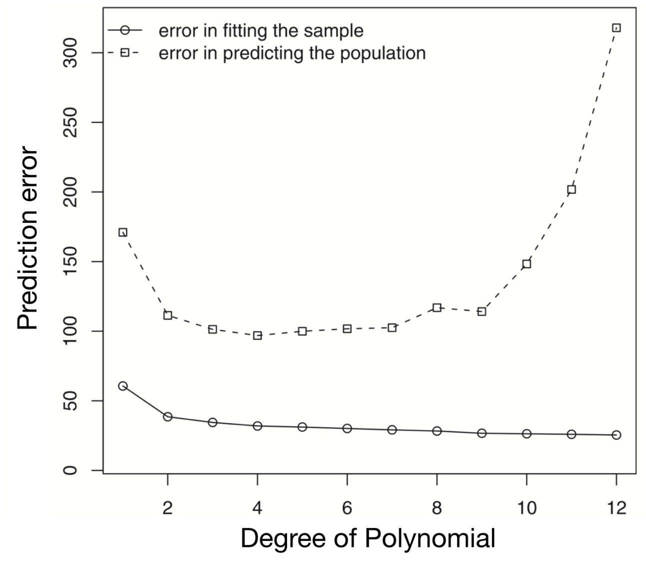
Simply put...

Bias arises from strong **model assumptions** not being met by the environment.

Variance arises from high **model flexibility** fitting the noise in the data (i.e., overfitting).

→ **Make strong assumptions** (use simple models), if possible.



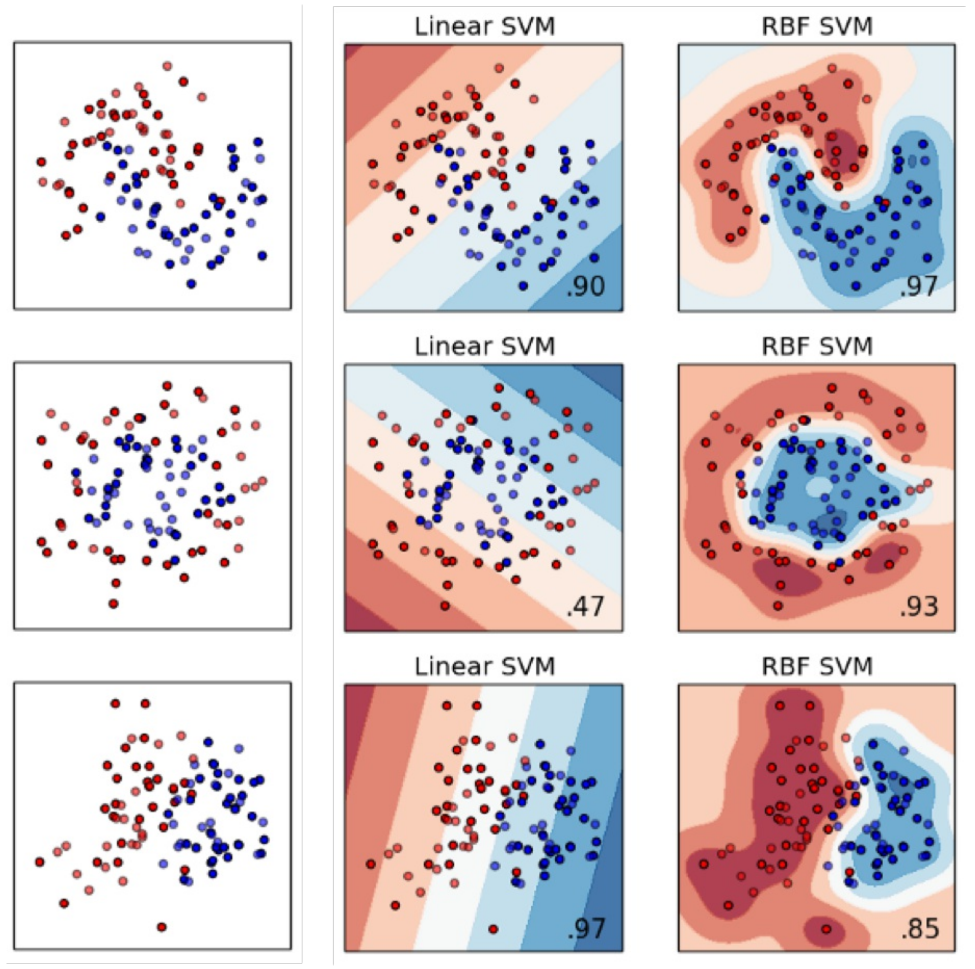


Linear or non-linear

One important model assumptions concerns linearity.

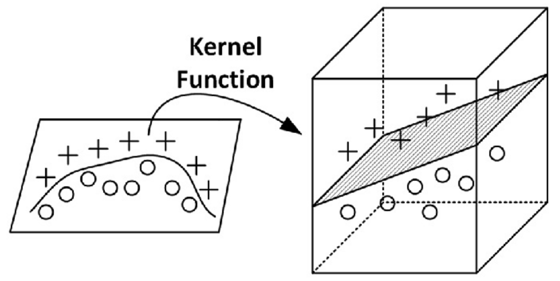
Linear models (`lm`, `glm`) make strong model assumptions. They are more often wrong, but also ceteris paribus **less prone to overfitting**.

Non-linear Models (everything else) make weaker model assumptions, leaving the exact relationship (more) open. They are are closer to the truth, but also ceteris paribus **more prone to overfitting**.



Kernel trick

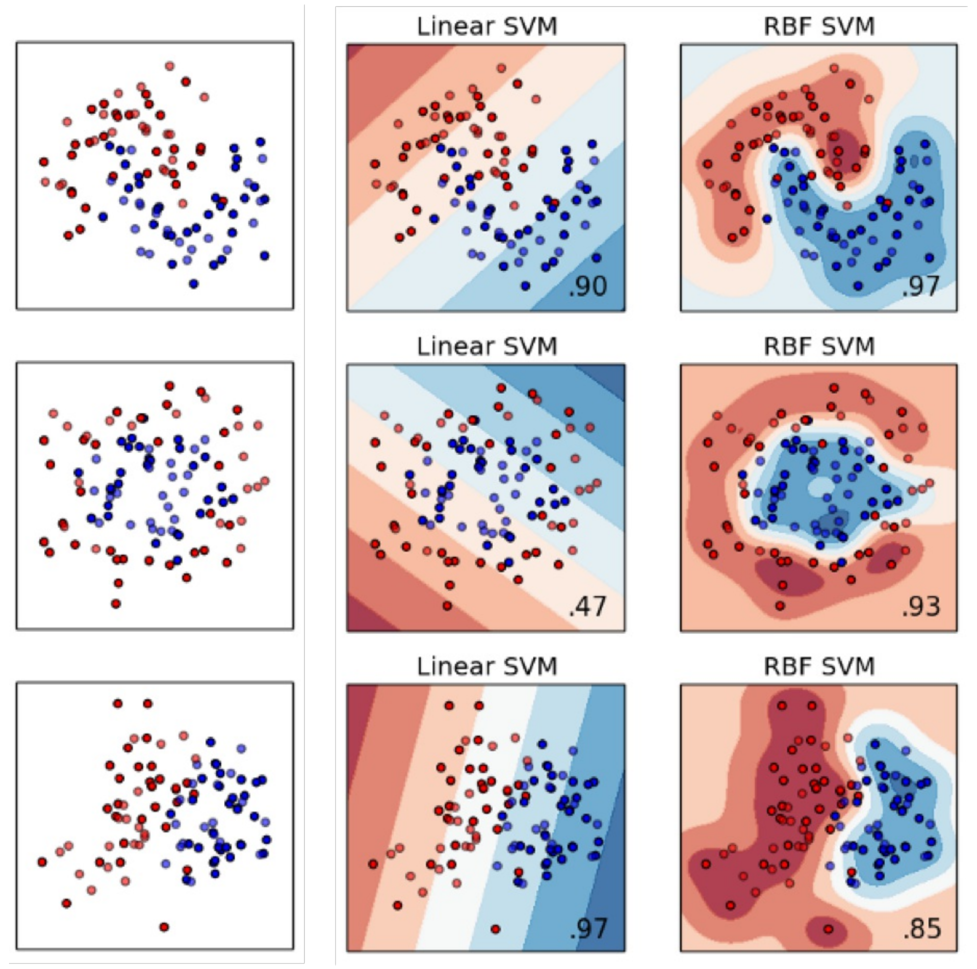
Transforms "input space" into new "feature space" to allows for object separation.



Used in **Support Vector Machines** (e.g., `method = "svmRadial"`) often using a **radial basis function** (rdf).

$$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$$

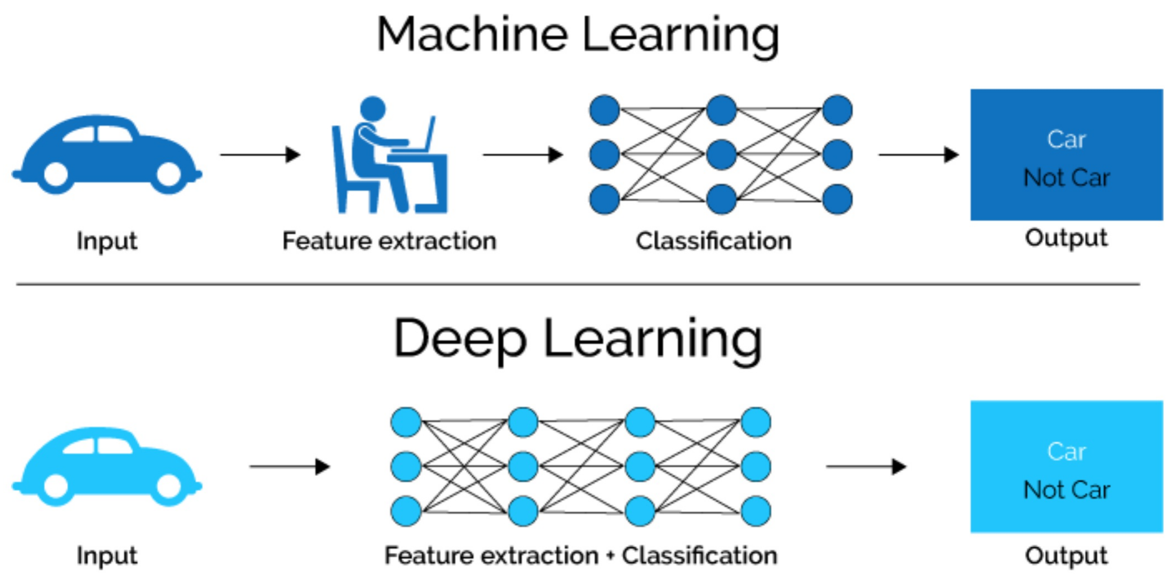
Kernels **re-represent objects** in terms of other objects!



Automatic feature engineering

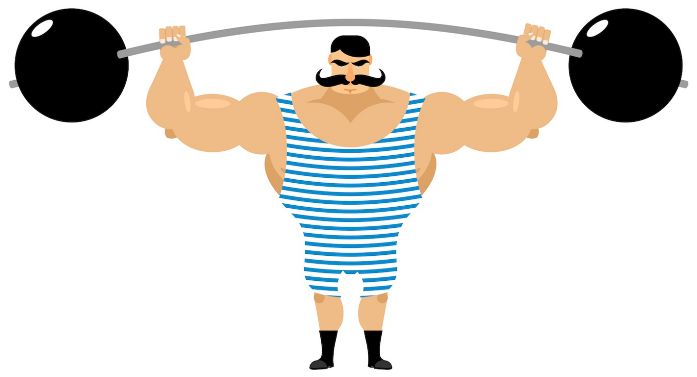
Deep learning aka neural networks and, especially, **convolutional neural networks**, excel because they generate their features.

Neural networks are not the focus of `caret` and this course. Powerful implementations based on **Google's Tensorflow** library are provided by `tensorflow`.



Robustness

To produce **robust predictions** that **suffer less from variance** ML models use a variety of **tricks**.



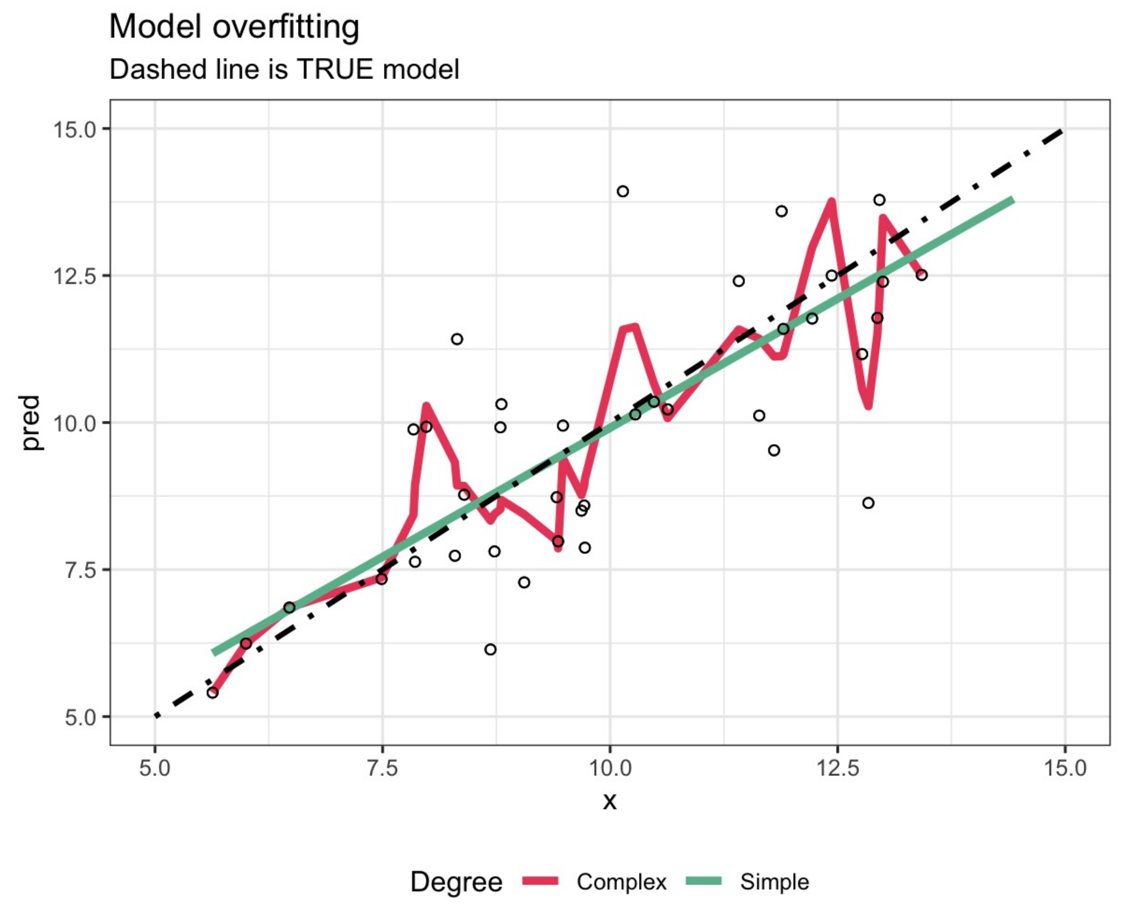
Approach	Implementation	Examples
Tolerance	Decrease error tolerance	`svmRadial`
Regularization	Penalize for complexity	`lasso`, `ridge`, `elasticnet`
Ensemble	Bagging	`treebag`, `randomGLM`, `randomForest`
Ensemble	Boosting	`adaboost`, `xgbTree`
Feature selection	Regularization	`lasso`
Feature selection	Importance	`random forest`

Regularization

Regularization is the process of adding model terms, usually **penalties for complexity**, in order to prevent overfitting (or solve a problem in the first place).

Loss = **Misfit** + **Penalty**

Name	Penalty	\`caret\`
AIC/BIC	$\|\|\beta\|\|_0$	-
Lasso	$\|\|\beta\|\|_1$	\`method = "lasso"\`
Ridge	$\|\|\beta\|\|_2$	\`method = "ridge"\`
Elastic Net	$\|\|\beta\|\|_2$	\`method = "elasticnet"\`



Bagging

Aggregate predictions from multiple fits to resampled data.

Especially beneficial for models that produce relatively unstable solutions, e.g., regression trees. `rpart` → `treebag`.

Random forest adds sampling of features to reduce dependencies across trees.

Algorithm

1. **Resample** data (with replacement)
2. **Fit** model to resampled data
3. **Average** predictions

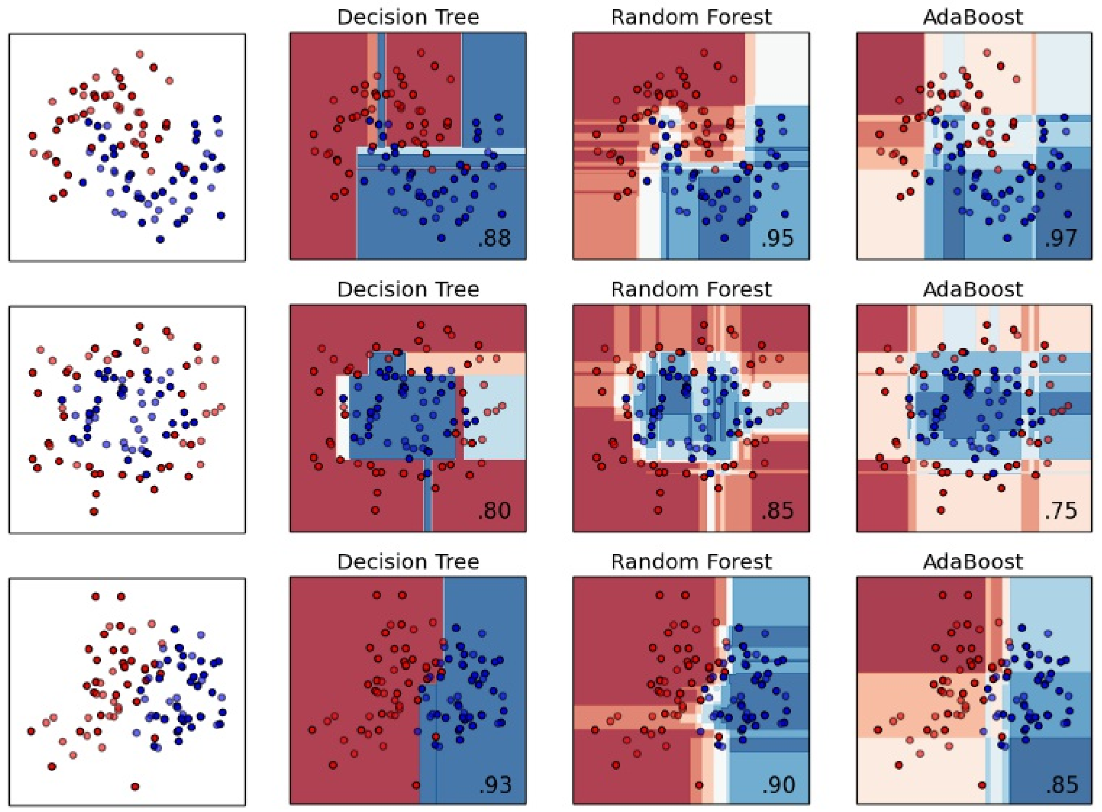
Boosting

Iterative algorithm that adaptively increases the weight given to previously misclassified samples.

New versions of the classic *adaboost* algorithm, e.g., `xgbTree`, **belong to the best ML models out there**.

Algorithm

1. Assign **equal weight** to samples
2. **Fit** simple model
3. **Increase weight of misfit samples** by model misfit for next iteration
4. **Average predictions weighted by model misfit**



Automatic feature selection

Many models reduce complexity by automatically relying on a subset of good features.

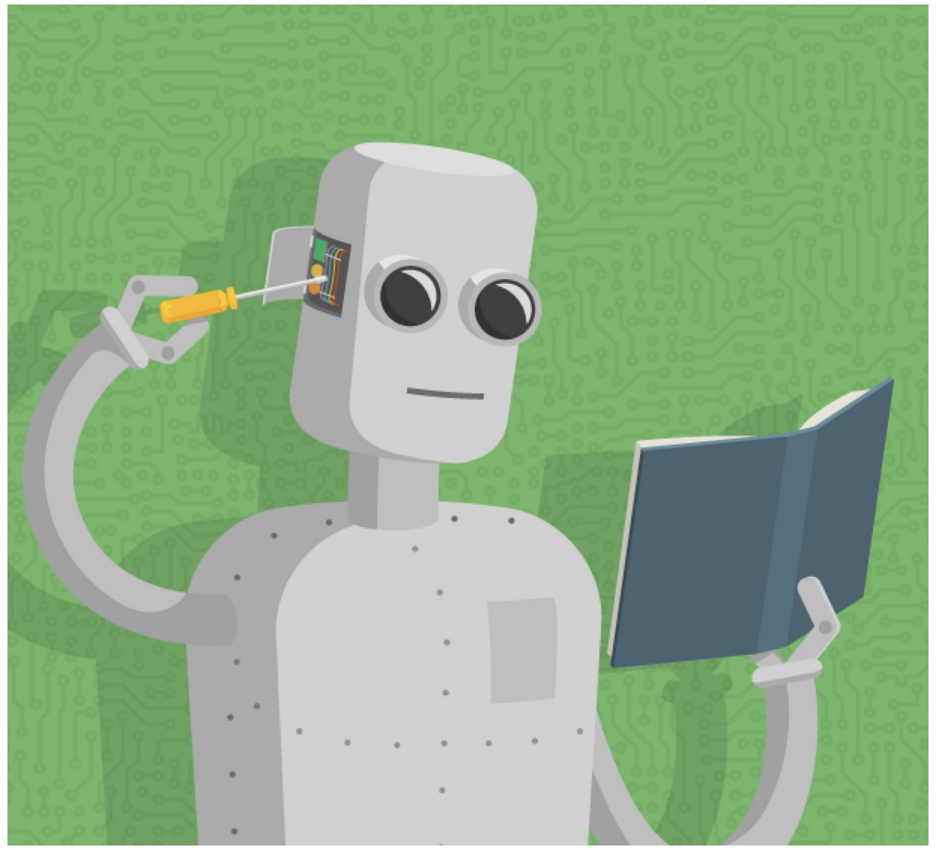
Two examples

LASSO

Regularization, in particular via `lasso`, frequently **estimates `beta = 0`** and, thus, essentially deselects that feature.

Random forests

As random forests select at any node the best of `mtry`-many randomly selected features, **unpredictive features may never come to action**. This is especially true for large `mtry`.



Excursus: Unsupervised learning

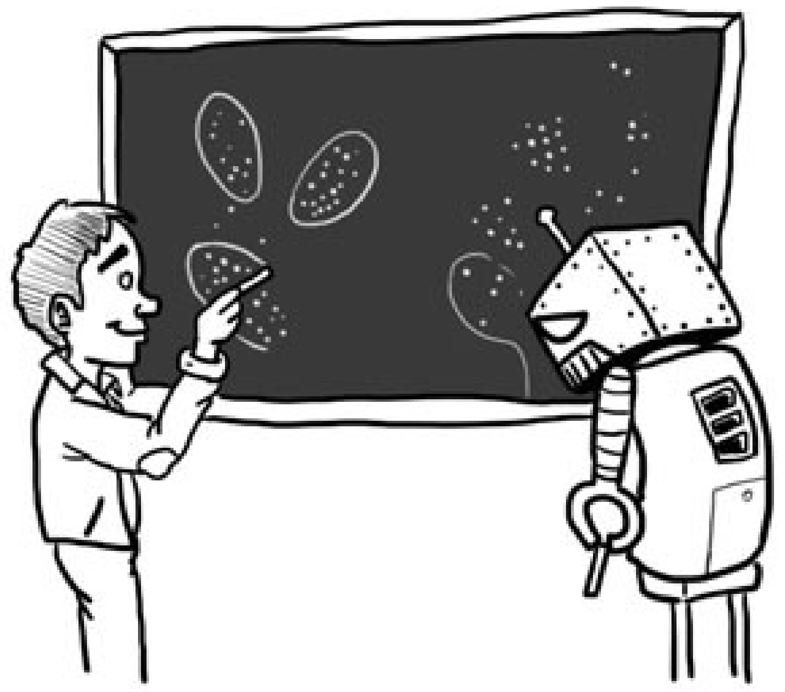
Unsupervised learning aims to **identify structure in the absence of labels**, i.e., a Criterion.

There is **no ground truth**, rendering unsupervised learning problems essentially **impossible to "solve"**, i.e., you never quite know how good a solution is.

Common questions

Are there **groups of cases** (clusters), which case belongs to which group, and how many groups are there? → `k-means` or `hierarchical clustering`

Are there **groups of features**, which features belongs to which group, and how many groups are there? → `pca` or `svd`



Excurse: Clustering

Clustering algorithms attempt to find distributed membership to \(k\) groups (clusters) such that **groups are maximally homogeneous**.

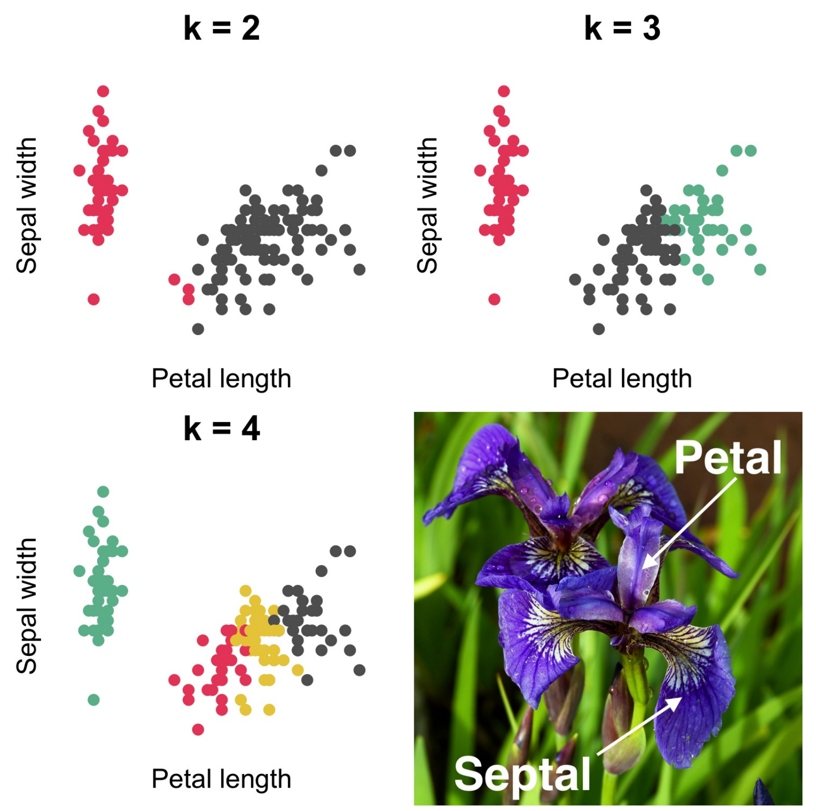
k-means

Assign cases to the closest cluster means, while iteratively shifting them around to **minimize within-group variance**.

hierarchical clustering

Place every case in one group. **Join clusters according to a pre-specified distance function** until the desired number of \(k\) clusters is reached.

R packages **cstab**

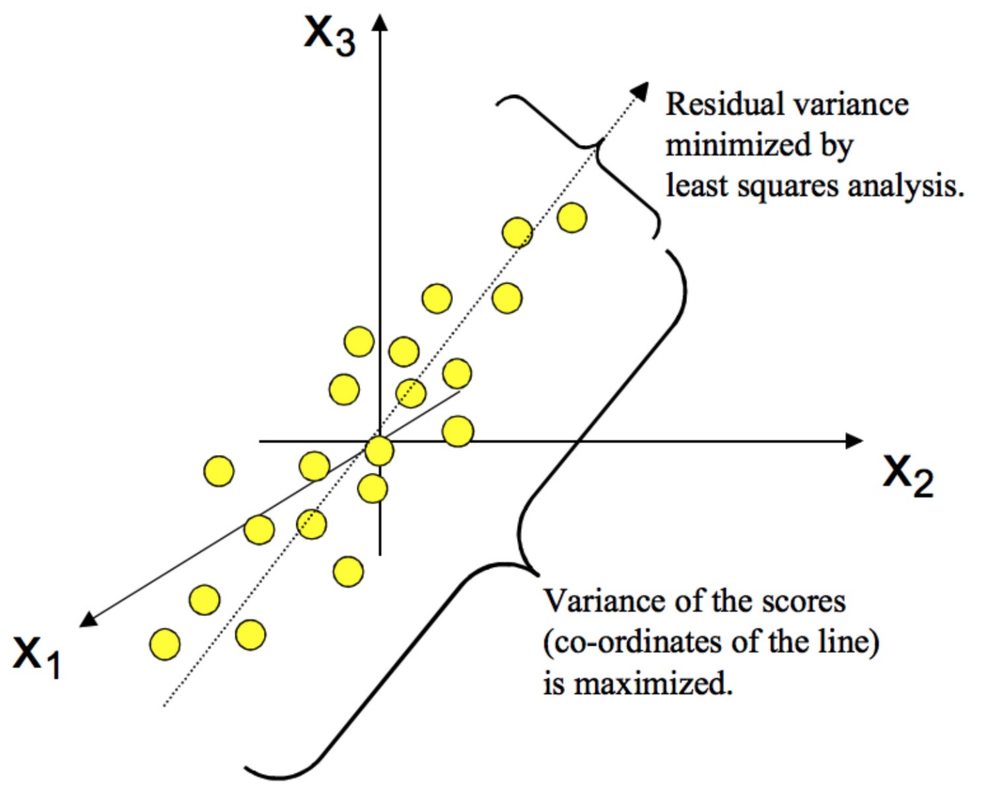


Excurse: Dimensionality reduction

Principal component analysis (PCA) is an **unsupervised** algorithm that re-represents the data in a **new feature space**.

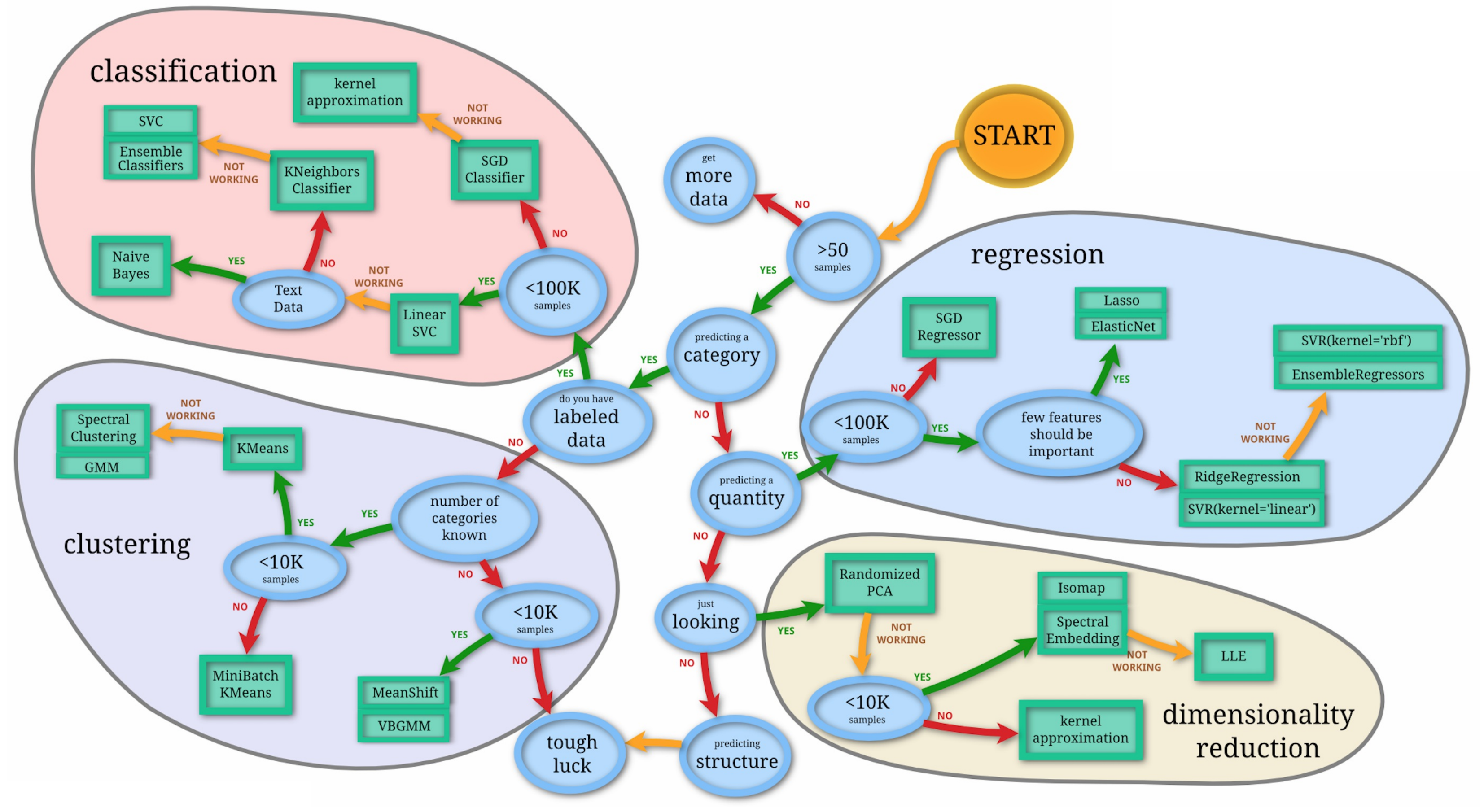
The new features aka **principal components are greedy** in that they attempt to explain and, thus, group together as many features as they can (leaving as few as possible for the next component).

Singular value decomposition (SVD) is the general mathematical approach underlying PCA and other methods.



x_3
Residual variance minimized by least squares analysis.
x_2
x_1
Variance of the scores (co-ordinates of the line) is maximized.

Some help in choosing models

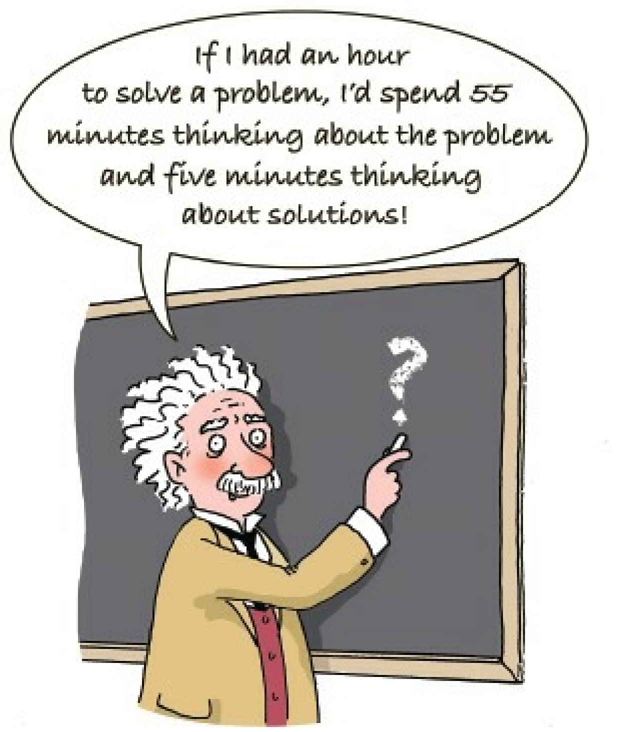


Source: scikit-learn

Remember

Pedro Domingos

Xavier Conort


If I had an hour to solve a problem, I'd spend 55 minutes thinking about the problem and five minutes thinking about solutions!

Practical