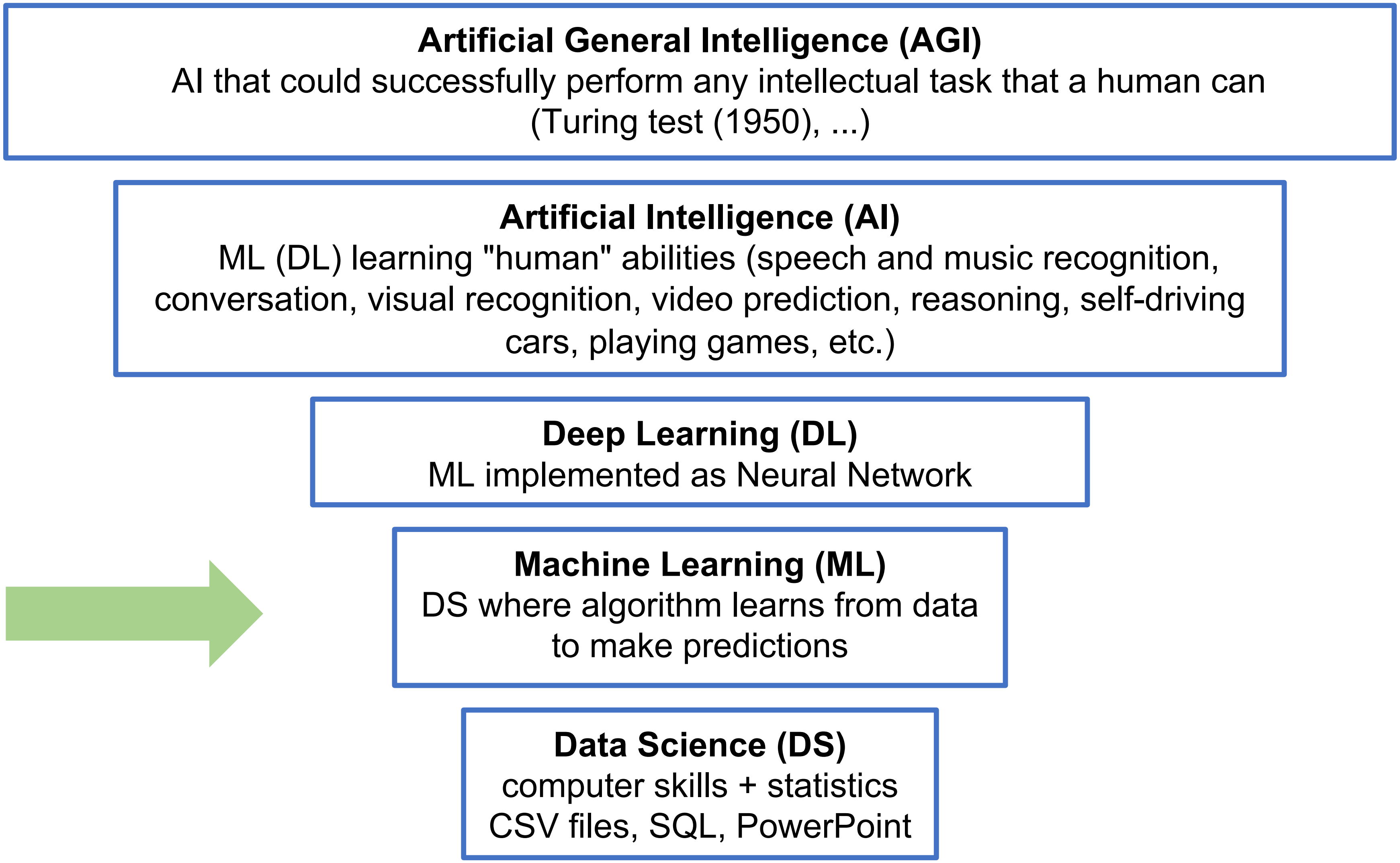
Classical Machine Learning Tools



Artificial General Intelligence (AGI)
AI that could successfully perform any intellectual task that a human can
(Turing test (1950), ...)

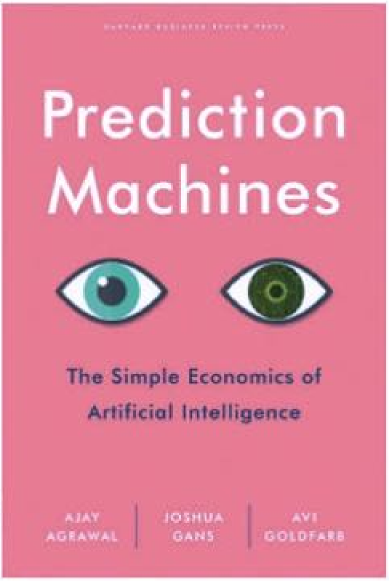
Artificial Intelligence (AI)
ML (DL) learning "human" abilities (speech and music recognition, conversation, visual recognition, video prediction, reasoning, self-driving cars, playing games, etc.)

Deep Learning (DL)
ML implemented as Neural Network

Machine Learning (ML)
DS where algorithm learns from data to make predictions

Data Science (DS)
computer skills + statistics
CSV files, SQL, PowerPoint

Machine Learning Systems = Prediction Machines



Ajay Agrawal
University of Toronto

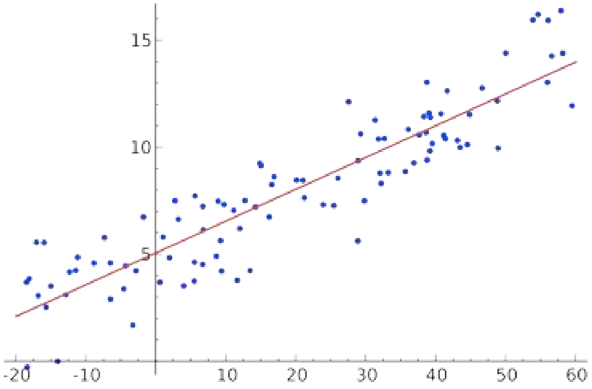
ML = "Cheap Predictions"

AI = "Cheap Predictions"

OK, so what is Machine Learning?

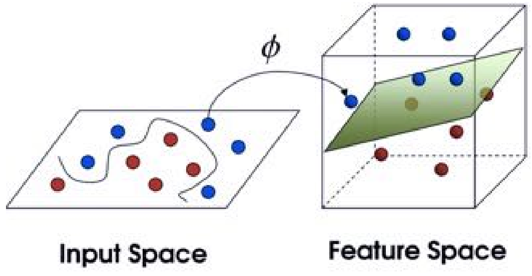
- A computer program
- used to describe data in a compressed way
- to make predictions and take actions
- most common ML tasks - regression or classification

Simple Analytics



Linear Regression y = f(x) = ax + b
Model uses only **two parameters**:
a = slope,
b = intercept

Standard ML



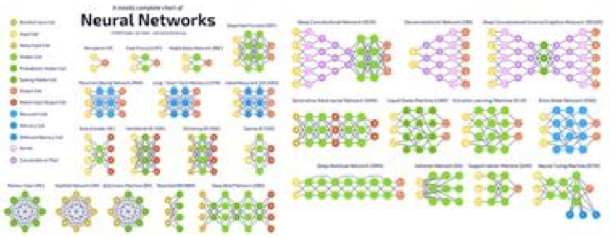
Other standard ML models may have up to **1-2 thousand parameters.**

Regression & Classification

Classification - Logistic Regression

Decision trees (Random Forest, XGBoost)
Clustering (K-Means, ...)
KNN (K-Nearest Neighbours)
SVM (Support Vector Machines),
PCA (Principal Component Analysis),
LDA (Linear Discriminant Analysis),
etc.

Neural Networks (Deep Learning)

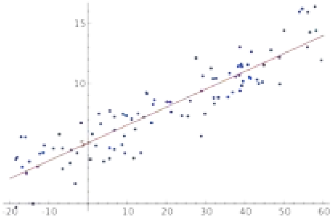


Typical example - classification – dog or cat.
CNN model (Convolutional Neural Network) may have millions of parameters.
Deep learning text (NLP) models may have up to **175 Bln parameters**.

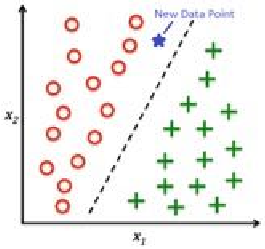
Some applications:
Recognizing images, music, speech
NLP = Natural Language Processing
Machine Translation
supervised, unsupervised, reinforcement learning
Generative Models (GANs)

Five Most Common "Classical" ML Algorithms

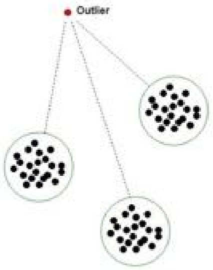
Regression



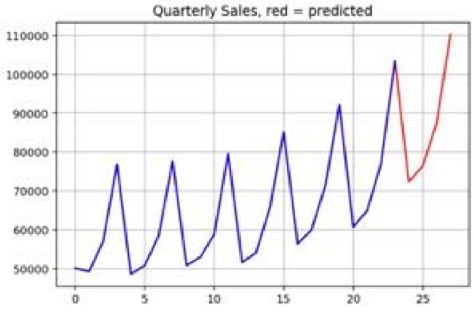
Classification



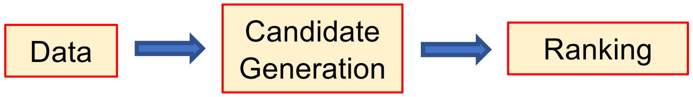
Anomaly Detection



Forecasting



Recommendation, Ranking



Data

Candidate Generation

Ranking

Most Common "Classical" Algorithms

- **Regression & Classification**
  .. Regression – draw line through points
  .. Classification – find to which class the data point belongs

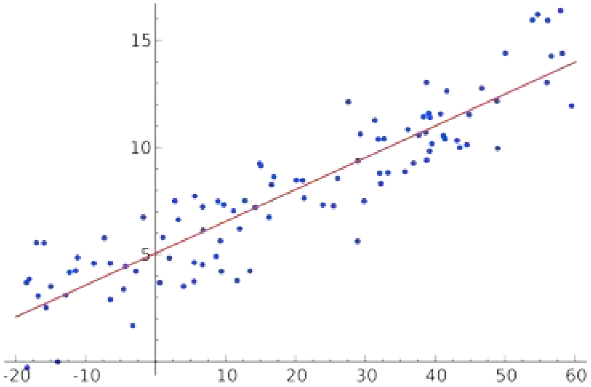
- **Supervised vs unsupervised**
  .. Supervised – data is "labeled" (cat or dog, bought or not, ...)
  .. Unsupervised – we need to find order in data
  (Clustering, Principal Component Analysis, etc.)

- **Predictive Analytics**
  .. Regression
  .. Classification (this type of prospects buy with 70% probability)
  .. Recommendations (Collaborative Filtering – people who bought this also bought that)
  .. Time Series Forecasting with seasonality and trend

- **Anomaly Detection**
  .. how far the event from the means or clusters

- **Recommendation & Ranking**
  .. Amazon and Netflix recommendations
  .. Google search results
  .. Newsfeeds (Facebook, etc.)

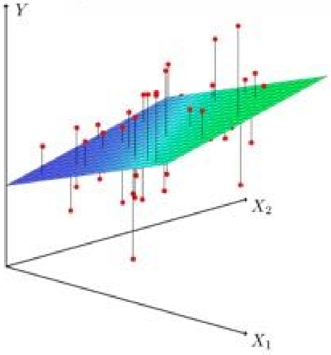
Model – Linear Regression



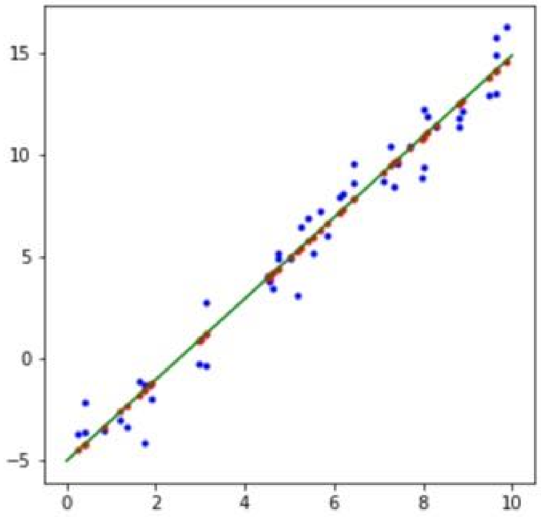
Linear Regression y = f(x) = ax + b
Model uses only two numbers: a = slope, b = intercept

Multivariate Linear Regression

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n$$



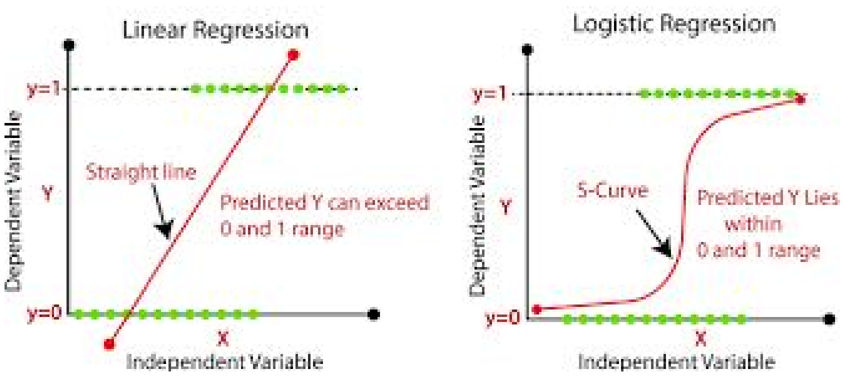
```
rng = np.random.RandomState(seed=None)
x = 10 ∗ rng.rand(50)                  # 50 random numbers in [0,10]
y = 2 ∗ x − 5 + rng.randn(50)          # 50 numbers between [−5, 15]

from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)
model.fit(x[:, np.newaxis], y)   # here x − one or more columns, y − row

# show the model line:
xt = np.linspace(0, 10, 1000)
yt = model.predict(xt[:, np.newaxis]) # this will draw a line
yp = model.predict( x[:, np.newaxis])

fig, ax = plt.subplots()
_ = ax.scatter(x,   y, marker='.', color='blue');  # training data
_ = ax.scatter(x,  yp, marker='.', color='red');   # project onto the line
_ = ax.plot.  (xt, yt,             color='green'); # line
_ = plt.show();
```

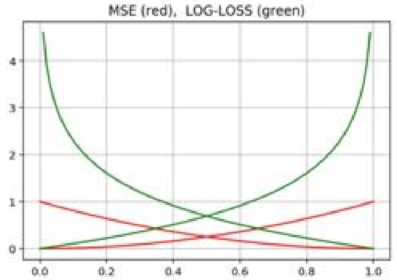


Logistic Regression
(binary classification model)

$$ln\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 x$$

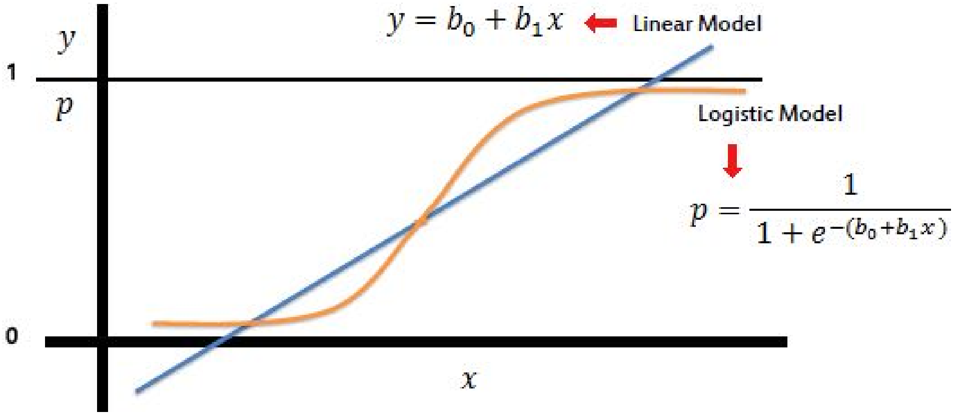
$$=> P = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$





The "beta" coefficients are fitted using iteratively gradient descent.

Common error function consists of two separate functions (for target values 0 and 1). It is called "Log-Loss" error function.



If "q" is model prediction, and "p" is probability of actual label, then similarity between q & p can be expressed as **cross-entropy** H(p,q):

$$H(p,q) \;=\; -\sum_i p_i \log q_i \;=\; -y \log \hat{y} - (1-y)\log(1-\hat{y})$$

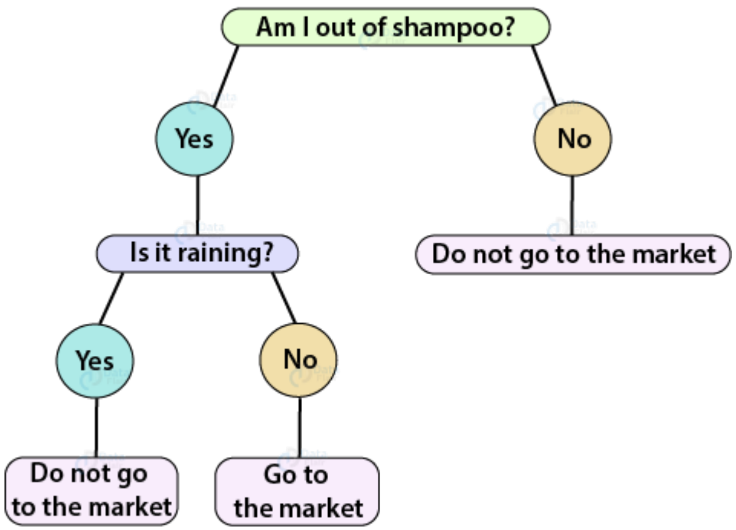
Decision Tree (DT)

DT goes from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves).
DT can do classification or regression.
DTs are intuitive and easily interpretable.

Algorithms for constructing DTs usually work top-down, by choosing and splitting on a variable at each step that most informative for the target label (produces the purest daughter nodes).

Splits can result in more than two branches.

Decision Trees Example



Decision Tree Building Algorithms

- ID3 (Iterative Dichotomiser 3) – uses smallest **Entropy metrics**
- C4.5 – successor of ID3 (since 1993), uses **Entropy metrics**
- CART (Classification And Regression Tree) – (since 1984), uses **Gini metrics**

- CHAID (CHi-squared Automatic Interaction Detector). Performs multi-level splits when computing classification trees.
- MARS: extends decision trees to handle numerical data better.
- Conditional Inference Trees. Statistics-based approach that uses non-parametric tests as splitting criteria, corrected for multiple testing to avoid overfitting. This approach results in unbiased predictor selection and does not require pruning.

Trees:

- Greedy splitting (top->down)
- Greedy pruning (to prevent over-fitting)
- Fast and easy to build
- Surprisingly good
- Explainable

Ensemble Learning

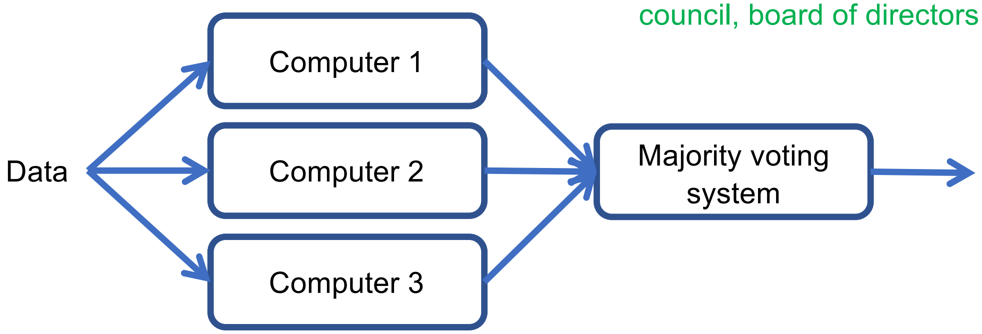
Idea - train several models to do predictions.
Then combine predictions of these models to get a better predictor.

Idea:

make a reliable system from unreliable blocks using single-shot democratic voting

Power of the majority

committee, senate, congress, council, board of directors, ...



Bagging

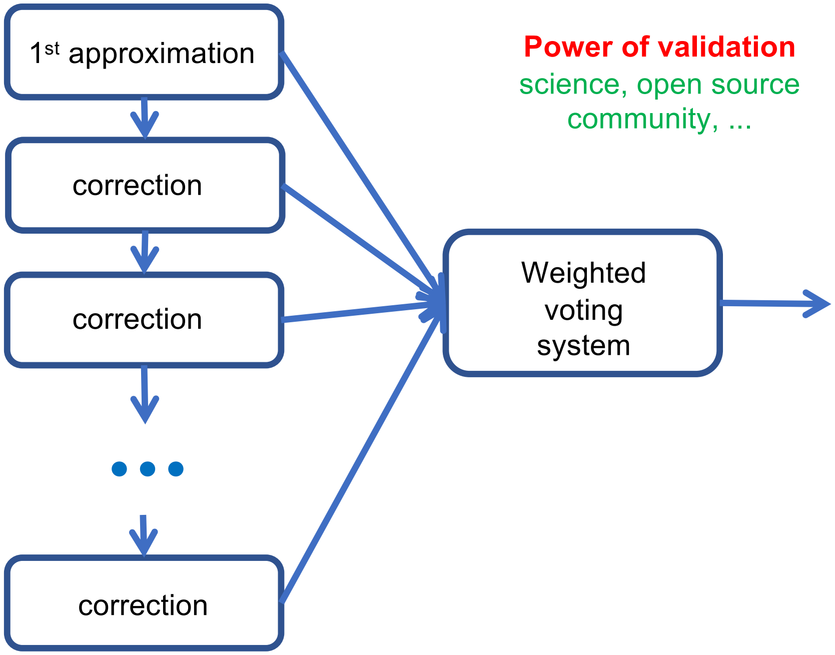
Wrong bias is not removed, it is simply averaged

Idea:

get to correct answer by making multiple corrective steps

Power of validation

science, open source community, ...



Wrong bias is removed using correction and validation

Boosting

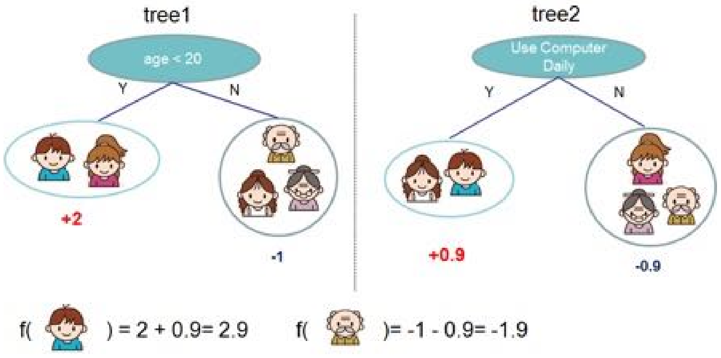
Ensemble Learning

Idea - train several models to do predictions.
Then combine predictions of these models to get a better predictor.

It is possible to combine any types of models into one predictor.
But commonly people combine variations of similar models, like **decision trees** models.

Two most famous approaches are:

- **"Random Forest"**
- **"Gradient Boosted Trees"**



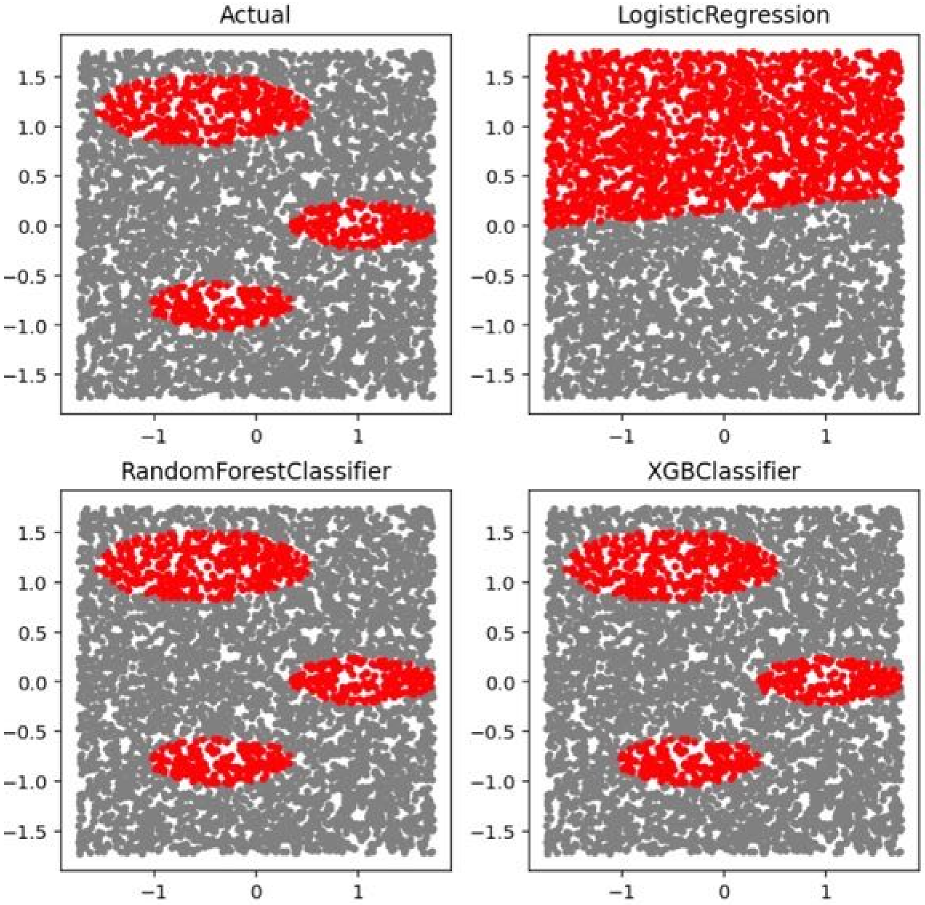
Under the hood these models combine 100 ... 200+ similar trees.

The differences between algorithms:

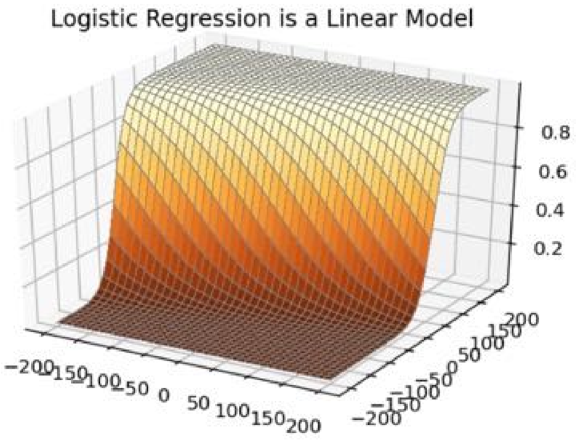
- how you select the data to train those trees
- how you combine the predictions of these trees

Binary Classification Problem

Example – two classes: red and grey.

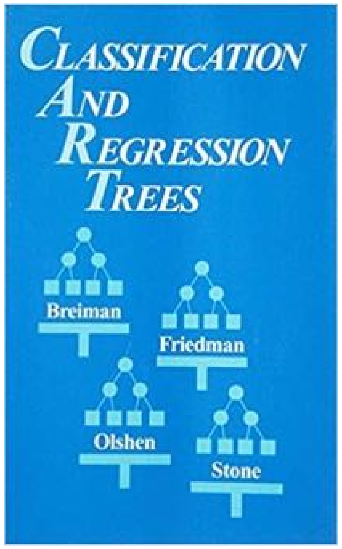


Ensemble Tree Classifiers succeed where Linear Classifiers fail

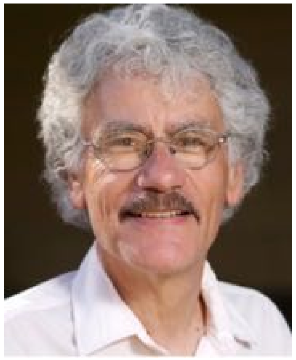


$$\log_b \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

Ensemble tree classifiers
(Random Forest , XGBoost)
can solve complicated problems



Leo Breiman
UC Berkeley
Statistics
1928-2005



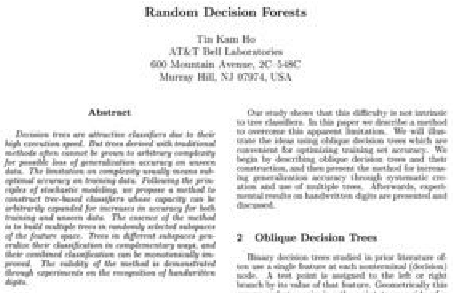
Jerome H. Friedman
Statistics
UC Berkeley,
Stanford
1939-

Richard A. Olshen
Biostatistics
UC Berkeley,
Stanford
1942-

Charles J. Stone
Math, Probability
UCLA,
UC Berkeley
1936-2019

Classification and
Regression Trees
(1984) by BFOS (BFOS = Breiman, Friedman, Olshen, Stone)

Random Decision Forests
(1995) by Tin Kam Ho



Random Decision Forests

Tin Kam Ho
AT&T Bell Laboratories
600 Mountain Avenue, 2C-548C
Murray Hill, NJ 07974, USA

Abstract

Decision trees are attractive classifiers due to their high execution speed. But trees derived with traditional methods often cannot be grown to arbitrary complexity for possible loss of generalization accuracy on unseen data. The limitation on complexity usually means suboptimal accuracy on training data. Following the principles of stochastic modeling, we propose a method to construct tree-based classifiers whose capacity can be arbitrarily expanded for increases in accuracy for both training and unseen data. The essence of the method is to build multiple trees in randomly selected subspaces of the feature space. Trees in different subspaces generalize their classification in complementary ways, and their combined classification can be monotonically improved. The validity of the method is demonstrated through experiments on the recognition of handwritten digits.

Our study shows that this difficulty is not intrinsic to tree classifiers. In this paper we describe a method to overcome this apparent limitation. We will illustrate the ideas using oblique decision trees which are convenient for optimizing training set accuracy. We begin by describing oblique decision trees and their construction, and then present the method for increasing generalization accuracy through systematic creation and use of multiple trees. Afterwards, experimental results on handwritten digits are presented and discussed.

2 Oblique Decision Trees

Binary decision trees studied in prior literature often use a single feature at each nonterminal (decision) node. A test point is assigned to the left or right branch by its value of that feature. Geometrically this

Tin Kam Ho
at Bell Labs in NJ.
then AI at IBM (since 2014)

FORTRAN
Random Forest
2002-2003

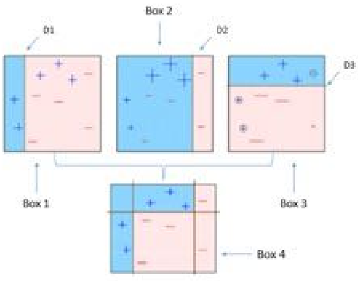
Adele Cutler
UC Berkeley, then Utah State University.
Wrote original Random Forest in Fortran with Leo Breiman in 2002-2003
https://www.stat.berkeley.edu/~breiman/RandomForests/cc_software.htm

Boosting Algorithms

Strength of Weak Learners

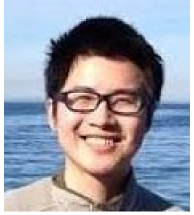
AdaBoost (1995)

Yoav Freund & Robert Schapire (2003 Gödel Prize).



XGBoost (2014)

more flexible and optimized
Tianqi Chen (was a Ph.D. student)



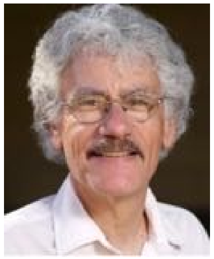
Light GBM (2016, Microsoft)

fast-performing, distributed
(GBM = Gradient Boosting Machine)

CatBoost (2017, Yandex)

big data, categorical data, multiple GPUs.

Leo Breiman
UC Berkeley



Jerome H. Friedman
UC Berkeley,
Stanford

From Wikipedia, etc:

The idea of gradient boosting originated in the observation by **Leo Breiman** that boosting can be interpreted as an optimization algorithm on a suitable cost function (**1997**, Berkeley).

Explicit regression gradient boosting algorithms were subsequently developed by **Jerome H. Friedman** (**1999**, Berkeley).

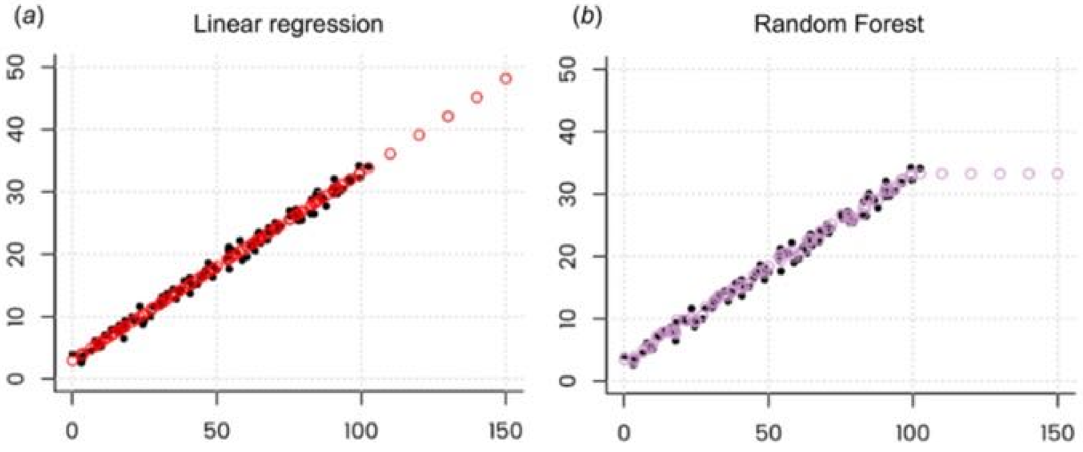
... simultaneously with the more general functional gradient boosting perspective of **Llew Mason, Jonathan Baxter, Peter Bartlett and Marcus Frean** (**1999**).

Stochastic gradient boosting - by Jerome H.Friedman, **2002** – adding random sampling to gradient boosting algorithm improves speed and accuracy.

Ensemble Tree Regressors can NOT do extrapolation

For example, **RandomForestRegressor** or **HistGradientBoostingRegressor**.
It does interpolation for values inside the range of values.
But it can not go outside.
- https://neptune.ai/blog/random-forest-regression-when-does-it-fail-and-why

RandomForest Regressor works by finding closest matches in given range.
We take random subset of data (columns, rows), and build a tree splitting by "x".
Each leave has "x" and "y".
We build hundred trees like this.
For inference, given value "x" we traverse all 100 trees (by "x") and find 100 values of "y".
Then we average them to get the desired prediction.
The result is always within the range of "y" values. It can not extrapolate.
If you look at prediction values they will look like this:

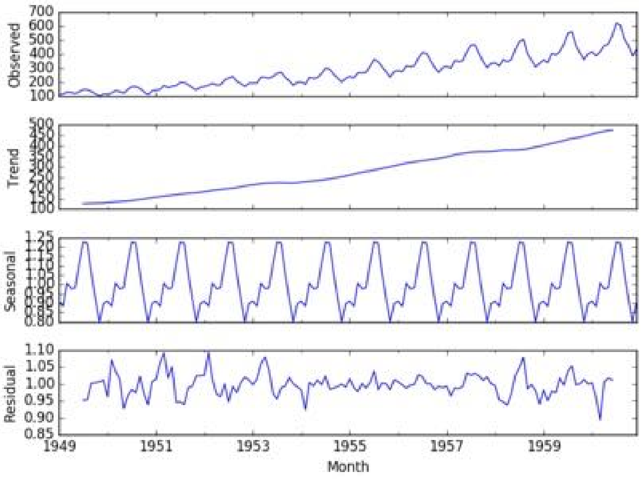


Ensemble Tree Regressors can be successfully used on de-trended time series.
Note – you need to forecast the trend separately.

Other approaches include SVM regression, Linear Regression, Deep Lerning (RNN), combining predictors using stacking, etc.

We can use modified versions of random forest, for example Regression Enhanced Random Forest (RERF, 2017):
- https://arxiv.org/pdf/1904.10416.pdf

Time Series Forecasting



- **Naive Qualitative** (forecast = last value, or an average of last period)
- **Trend, Linear Regression**
- **Multivariate Linear Regression**
- **Exponential Smoothing** (weighted moving average)
- **Holt-Winters** = Triple Exponential Smoothing (smooth, trend, seasonality)

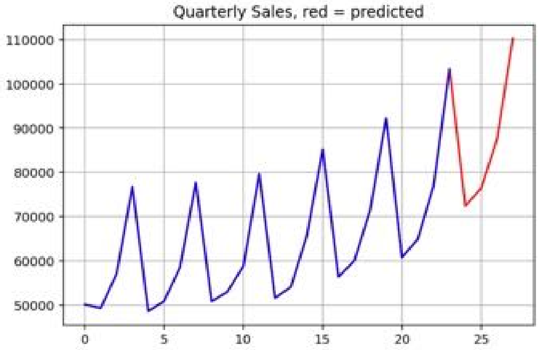
- **Calendar Seasonality Features + "events" features, modeled using Random Forest or Boosted Trees (my favorite)**

- **Fourier decomposition** to model Seasonality
- **ARMA** (Auto-Regressive Moving Average)
- **ARIMA** (Auto-Regressive Integrated Moving Average)
- **SARIMA** (S=Seasonality), SARIMAX (X - eXogenous regressors)
- **Auto-correlations**, bivariate cross-correlation, multivariate time series analysis

- **Causal** (cause-effect) - modeling specific causes/drivers that can influence the target value
  (drivers – number of resources, spending on advertising, promotion,
  legal change, website redesign, sport event, new law, election, etc.).
  Some drivers cause permanent shift, others – temporary response.
- **Monte-Carlo** (Random Walk Simulation)

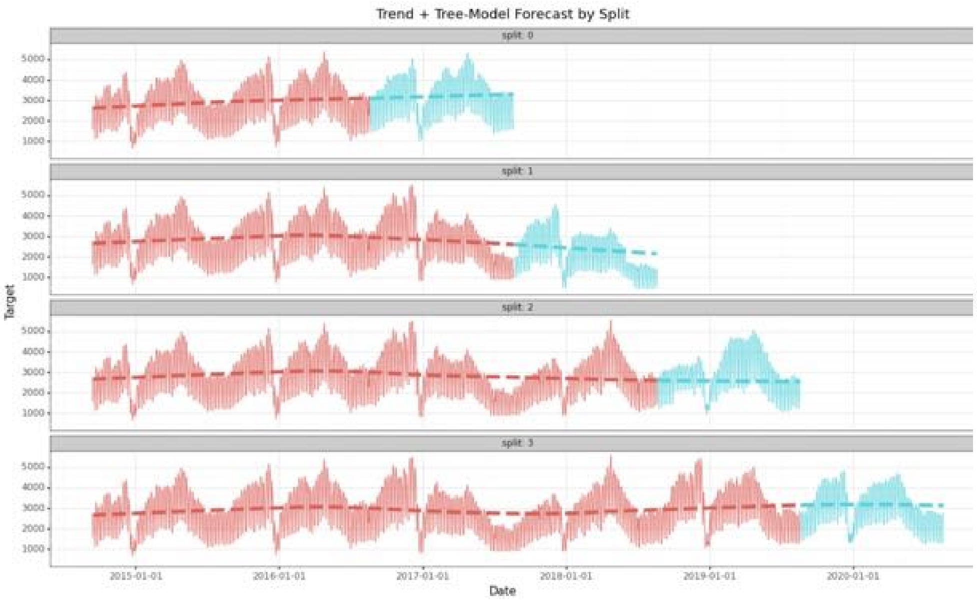
- **Neural Networks** - LSTM (Long Short Term Memory),
  multivariate LSTM, GRU (Gated Recurrent Unit)

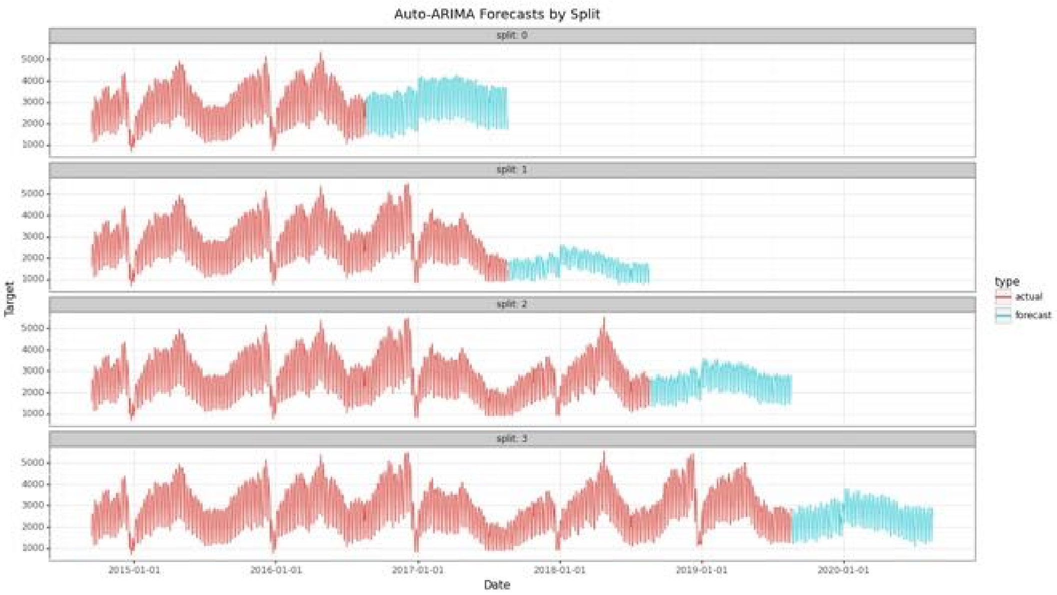
- **Facebook Prohpet** -
- **Azure AutoML** –
- **AWS DeepAR** – supervised, uses RNN (Recurrent Neural Network), 1 or more time series
- etc. etc.



Forecasting Example

Two models – Trend + Ensemble-Tree model vs Auto-ARIMA (SARIMAX)

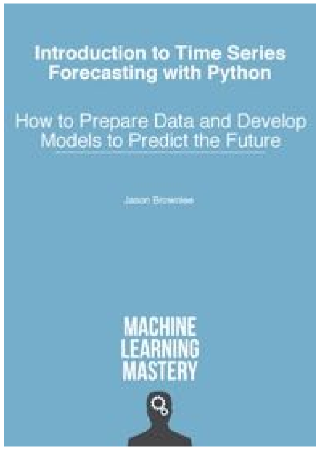




Method	Avg Error (across 4 splits)	Avg Train Time (across 4 splits)
Trend + Tree-Model	14.6%	~2sec
Auto-ARIMA (SARIMAX)	21.6%	~2min

Jason Brownlee

Best two books on time series forecasting:



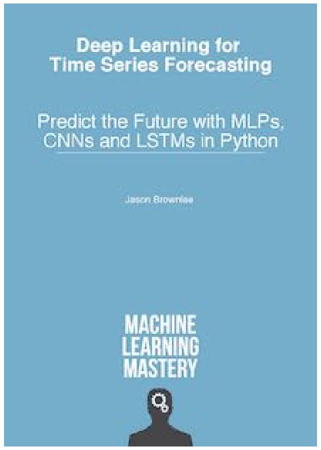
Introduction to Time Series Forecasting with Python

How to Prepare Data and Develop Models to Predict the Future

Jason Brownlee

MACHINE LEARNING MASTERY

Introduction to Time Series Forecasting With Python



Deep Learning for Time Series Forecasting

Predict the Future with MLPs, CNNs and LSTMs in Python

Jason Brownlee

MACHINE LEARNING MASTERY

Deep Learning for Time Series Forecasting

Jason Brownlee

Melbourne, Australia

Jason wrote 21 books on Machine Learning.
I highly recommend to buy all of them.
They come as PDF with separate python code files.

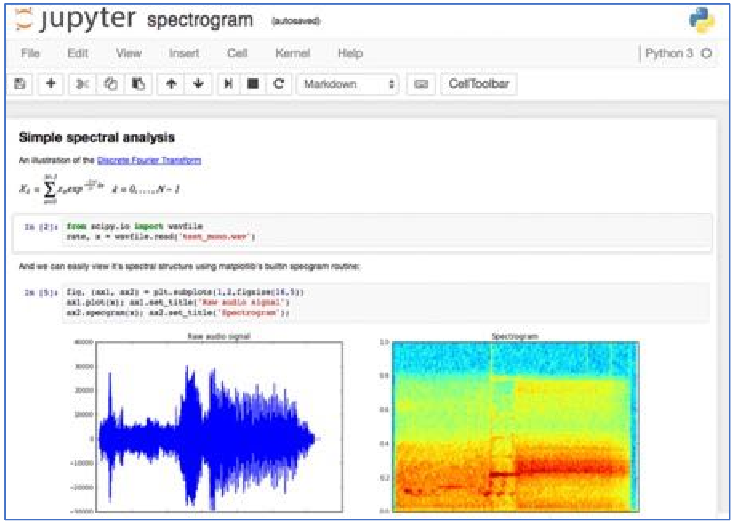
https://www.linkedin.com/in/jasonbrownlee/

https://machinelearningmastery.com/

https://SuperFastPython.com

Python, Jupyter, Scikit-Learn, ...

- Anaconda Python - https://www.anaconda.com
- Jupyter notebooks - https://jupyter.org
- python modules:
  - pandas - https://pandas.pydata.org
  - NumPy - https://numpy.org
  - matplotlib - https://matplotlib.org
  - scikit-learn - https://scikit-learn.org/stable
  - SciPy - https://scipy.org
- For Deep Leaning:
  - Google TensorFlow+Keras - https://www.tensorflow.org
  - Facebook PyTorch - https://pytorch.org



Simple spectral analysis

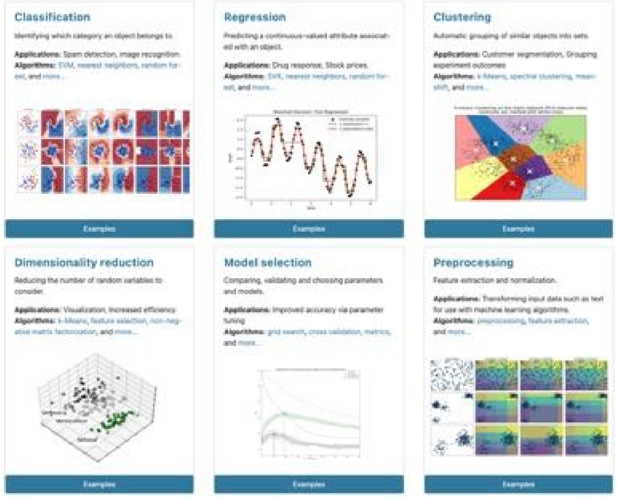
An illustration of the Discrete Fourier Transform

$$X_k = \sum_{n=0}^{N-1} x_n exp^{-\frac{2\pi i}{N} kn} \quad k = 0,\ldots,N-1$$

```
In [2]: from scipy.io import wavfile
        rate, x = wavfile.read('test_mono.wav')
```

And we can easily view it's spectral structure using matplotlib's builtin specgram routine:

```
In [5]: fig, (ax1, ax2) = plt.subplots(1,2,figsize=(16,5))
        ax1.plot(x); ax1.set_title('Raw audio signal')
        ax2.specgram(x); ax2.set_title('Spectrogram');
```



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: k-Means, feature selection, non-negative matrix factorization, and more...

Examples

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning
Algorithms: grid search, cross validation, metrics, and more...

Examples

Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: preprocessing, feature extraction, and more...

Examples

Anomaly Detection: Isolation Forest (IF) & RRCF

(RRCF = AWS Robust Random Cut Forest)

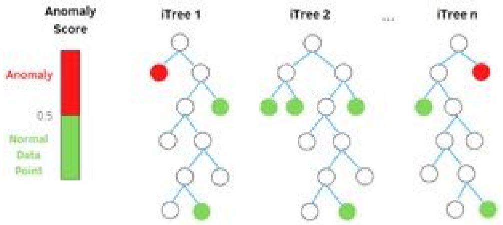
- https://en.wikipedia.org/wiki/Isolation_forest -
- https://stackoverflow.com/questions/63115867/isolation-forest-vs-robust-random-cut-forest-in-outlier-detection –

Isolation Forest:

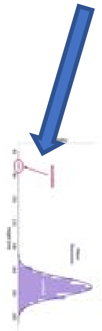
- https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/icdm08b.pdf – paper
- https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e – tutorial

RCF:

- http://proceedings.mlr.press/v48/guha16.pdf – paper
- https://freecontent.manning.com/the-randomcutforest-algorithm/ - tutorial



Anomaly



- **Both algorithms are ensemble methods based on decision trees.**
- **Both aim to isolate every single point.**
- **Ouliers tend to require less steps to get isolated.**

- **Isolation Forest (IF)** is open source (sklearn), whereas **AWS RRCF** is closed source (although look at https://github.com/kLabUM/rrcf)
- **RRCF** can work on **streams** (in streaming analytics service Kinesis Data Analytics) – it has partial_fit() method.
- **RRCF** is more scalable, can be parallelized between **multiple machines**. It also supports Pipe mode (streaming data via unix pipes) which makes it able to learn on much bigger data than what fits on disk
- **RRCF** performs better in **high-dimensional space** because it gives more weight to dimension with higher variance (according to SageMaker doc), while Isolation Forest samples at random
- Anomaly score is calculated differently. **IF**'s score is based on distance from the root node. **RRCF** is based on how much a new point changes the tree structure (i.e., shift in the tree size by including the new point). This makes **RRCF** less sensitive to the sample size

ROC Curve & Precision-Recall Curve

Receiver Operating Characteristic (**ROC**) curves for a binary classifier:

True Positive Rate (TPR) vs **False Positive Rate (FPR)**
at various threshold settings.

ROC curve was first developed by radar engineers during **World War II**
for quantifying the quality of detection of enemy airplanes.

TPR = REC = **Recall or Sensitivity** = TP/P = TP / (TP + FN)
(planes classified as planes / all planes' events)

FPR = **False Positive Rate** = **fall-out** = FP / N = FP / (FP + TN)
(noise classified as planes / all noise events)

PRE = **Precision** = TP/(TP+FP)
(planes classified as planes / all events classified as planes)

TNR = SPC = **Specificity** = TN/N = TN/(FP+TN)
(True Negative Rate)

AUC = Area Under the Curve

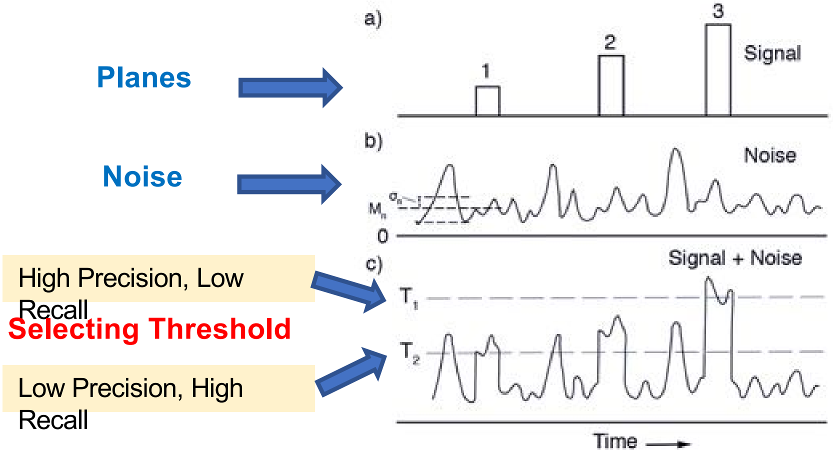
Simply the area under the ROC curve (1 = very good, 0.5 = bad)

F1 Score

- a single numbe ıonic average of the **precision** and **recall** :

$$\frac{1}{\frac{1}{2}\left(\frac{1}{\text{recall}}+\frac{1}{\text{precision}}\right)}$$

Radar detects planes.
It need to separate
"plane" signals from noise.



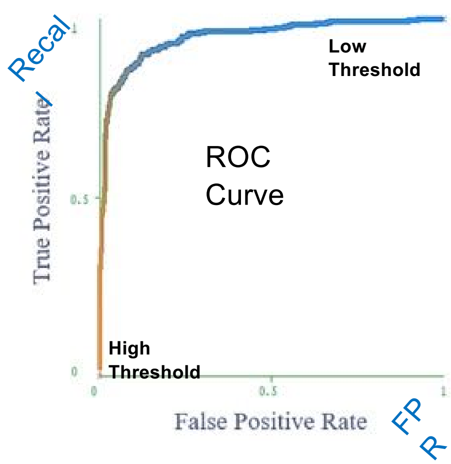
Planes

Noise

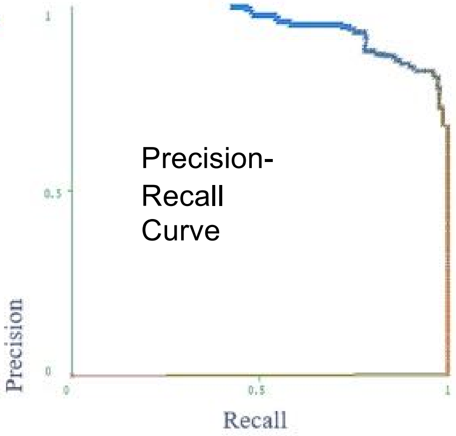
High Precision, Low Recall

Selecting Threshold

Low Precision, High Recall

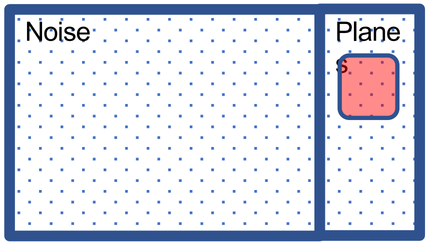


ROC Curve



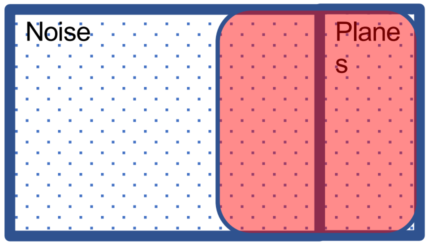
Precision-Recall Curve

Noise **Planes**



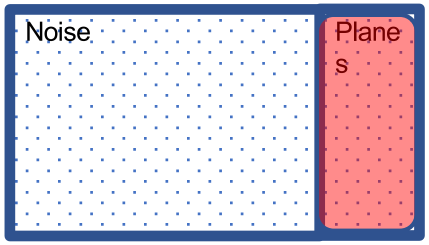
Precision – 100%
Recall – 15%

Good Precision = Sniper



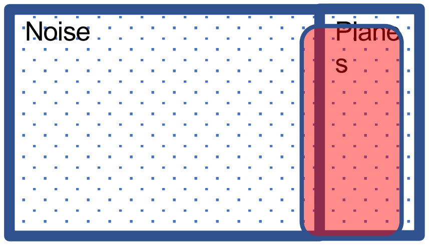
Precision – 50%
Recall – 100%

Good Recall = Cover All

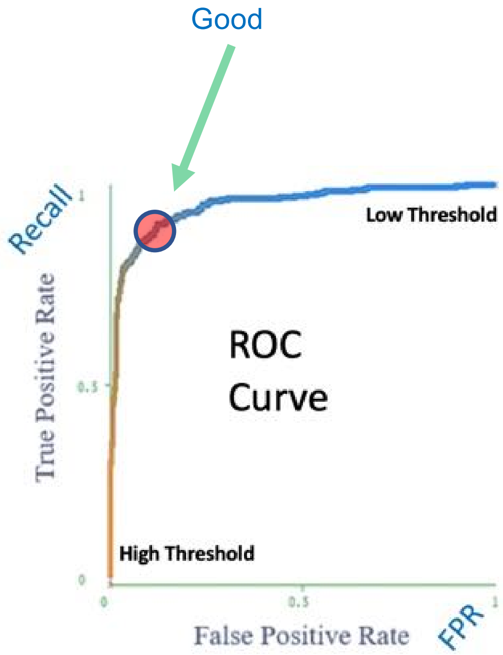


Ideal:
Precision – 100%
Recall – 100%

+



Good:
Precision – 90%
Recall – 90%

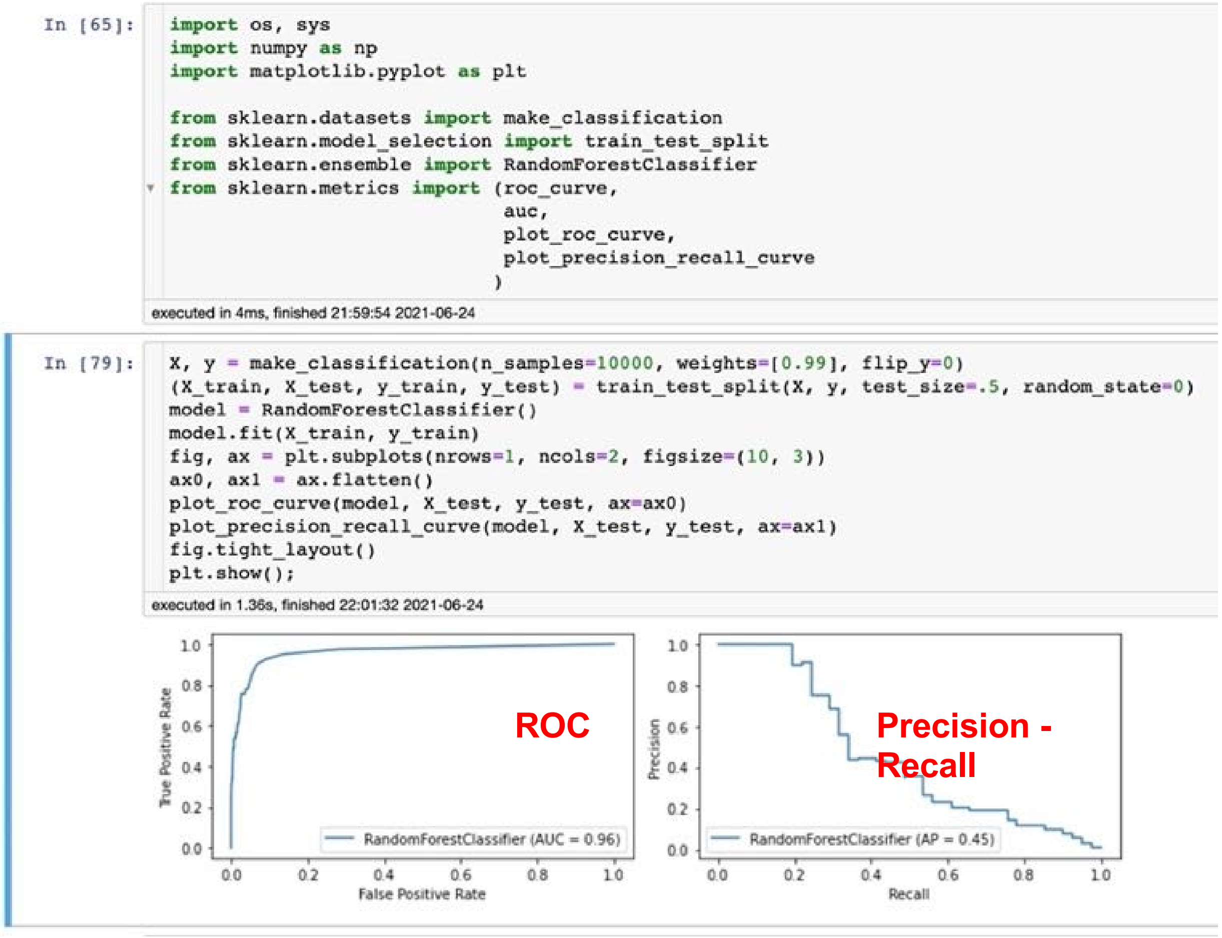


ROC AUC is terrible for imbalanced data. Precision-Recall is much better.

Imagine imbalanced data set:
for 10 airplanes we get 10,000 noise spikes.
Suppose we have a classifier with reasonably good ROC curve, which at the "sweet spot" identifies 90% of planes and only 1% of noise as planes. The AUC will be very close to 1.

But it is actually a terrible classifier.
Because it will cry "plane!" for noise spikes 10 times more than for real planes!

The Precision-Recall curve is a much better indicator. Please see example on the right.



```
In [65]:
import os, sys
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (roc_curve,
                             auc,
                             plot_roc_curve,
                             plot_precision_recall_curve
                            )
```

executed in 4ms, finished 21:59:54 2021-06-24

```
In [79]:
X, y = make_classification(n_samples=10000, weights=[0.99], flip_y=0)
(X_train, X_test, y_train, y_test) = train_test_split(X, y, test_size=.5, random_state=0)
model = RandomForestClassifier()
model.fit(X_train, y_train)
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10, 3))
ax0, ax1 = ax.flatten()
plot_roc_curve(model, X_test, y_test, ax=ax0)
plot_precision_recall_curve(model, X_test, y_test, ax=ax1)
fig.tight_layout()
plt.show();
```

executed in 1.36s, finished 22:01:32 2021-06-24

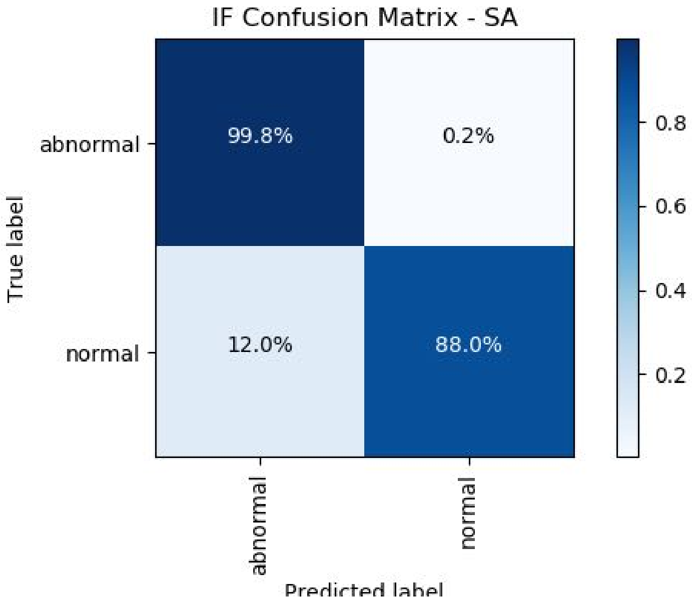
Confusion Matrix

For a simple binary classifier we are predicting one of two possibilities (Yes/No, Positive/Negative, Plane/No-Plane, Normal/Abnormal, etc.).

So we construct a 2x2 matrix to visualize True & False Positives and Negatives.

		Predicted class	
		P	*N*
Actual Class	*P*	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

```
from sklearn import metrics

expected = target
predicted = model.predict(data)

metrics.precision_score(expected, predicted)
metrics.recall_score(expected, predicted)
metrics.roc_curve(expected, predicted)
metrics.roc_auc_score(expected, predicted)

metrics.confusion_matrix(expected, predicted)

[[194156   4864]
 [    23    321]]
```



Accuracy = sum(Diagonal) / sum(ALL) = sum(Correct) / sum(ALL)
Precision = TP / (TP+FP)
Recall = TP / (TP+FN) = TP / All_Actual_Positives

Type I and Type II Errors

Two types of Errors:

- **Type I error** - false rejection of the null hypothesis
  "false-positive" conclusion:
  .. innocent person convicted
  .. positive COVID test for a healthy person

- **Type II error** - false acceptance of the null hypothesis
  "false negative" conclusion:
  .. guilty person not convicted
  .. sick person is released as healthy

	Accept H_0	Reject H_0
H_0 is True	Correct	**type I error (rejected True)**
H_0 is False	**type II error (accepted False)**	Correct

The type of error depends on the null hypothesis

Example: COVID test.

Null Hypothesis 1: person has COVID.
Error Type 1 is when the test says "**Healthy**" on a sick person
Error Type 2 is when the test says "Sick" on a healthy person

Null Hypothesis 2: person is Healthy.
Error Type 1 is when the test says "**Sick**" on a healthy person
Error Type 2 is when the test says "Healthy" on a sick person

Choosing null hypothesis:

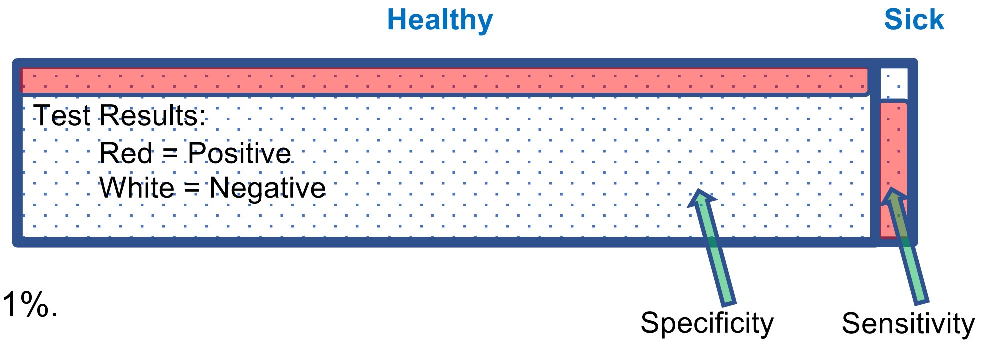
guilty / innocent: the person is presumed innocent until proven guilty.
Null Hypothesis = "Innocent"

sick / healthy: the person is presumed healthy.
Null Hypothesis = "Healthy"

airplaine / noise: the signals are mostly noise.
Null Hypothesis = "Noise"

Medical Test Paradox – Predictive Power of Positive or Negative Test Result

from
https://www.youtube.com/watch?v=IG4VkPoG3ko



We have 1,000 people.
1% of them have a disease:

```
    N_sick    = 10
    N_healthy = 990
```

We use a test with **sensitivity SE** = 90% and **specificity SP** = 91%.

This means that test shows results as following:

```
TP = True Positive   =   N_sick    * SE/100   =     9 out of  10
TN = True Negative   =   N_healthy * SP/100   =   901 out of 990

FN = False Negative  =    1 out of  10
FP = False Positive  =   89 out of 990
```

	Positive	Negative
Disease	9	1
Healthy	89	901

What is the predictive power of this test?

```
Positive result :  TP/(TP+FP)  =    9/(9+89)    =  0.092  (1 in 11 chance)

Negative result :  TN/(TN+FN)  =  901/(901+1)   =  0.999  (very good)
```

```
Sensitivity = Recall = True Positive Rate (TPR) : TPR = (TP / P) = TP / (TP + FN)
Specificity (SPC)    = True Negative Rate (TNR) : SPC = (TN / N) = TN / (FP + TN)
```

How to find "REGULARITY"

Overfitting and Regularization

Goal of training the model is to fit/learn a "**regularity**" in the data, and to avoid fitting noise (avoid overfitting).

How do we know that we are overfitting?

Let's split data into two sets:
- training
- testing

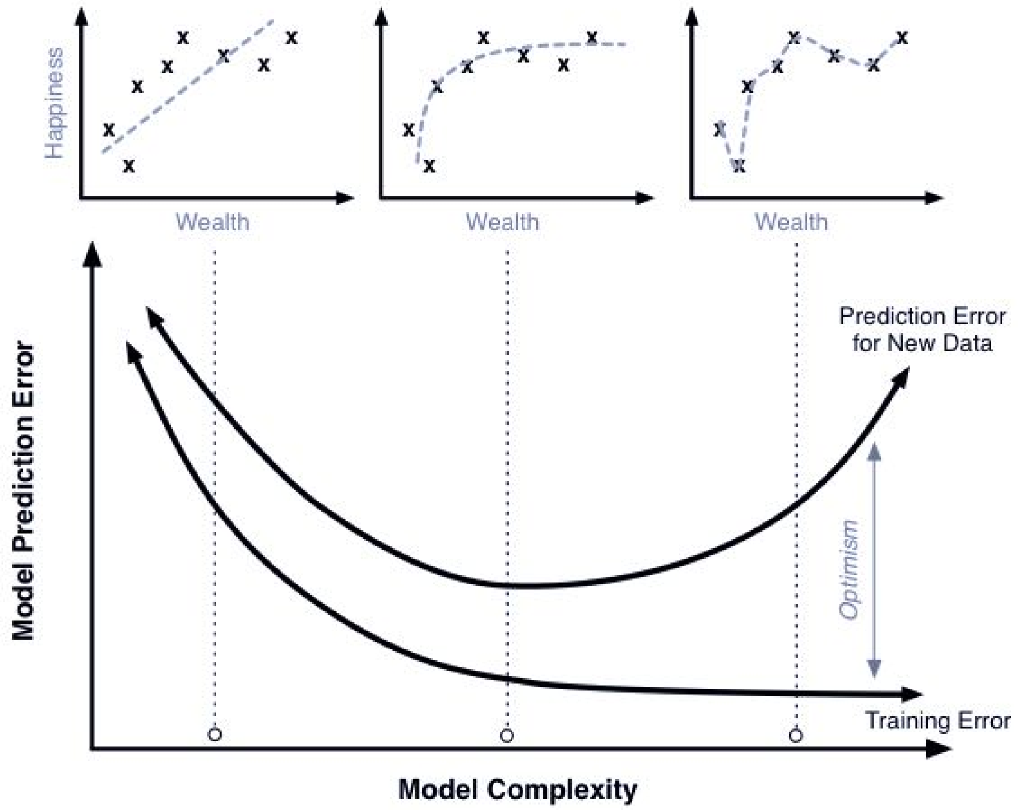
If we **overfit** on the training data (learn regularity and some noise specific to the training data), then accuracy measured on the test data may be bad (model doesn't generalize well).

Solution – try to find a more "**regular**" solution (less noise, less overfitting).

High Bias model (underfitting, model too simple)

Bias/Variance Trade-off, Good Generalization

High Variance model (model too complex, overfitting - fits noise)



Regularization: Ridge Regression

Multivariate (multiple) Linear Regression Regularization.

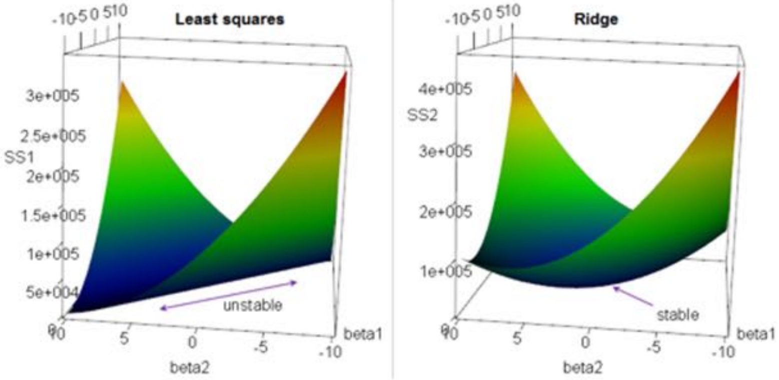
Suppose we have a OLS (Ordinary Linear Squared) regression where two parameters x1 & x2 are not orthogonal, but are equal, proportional to each other, or highly correlated. Then there may be an infinite number of coefficients which will fit the model.

For example, (a*x1 - x2),   (100*a*x1 - 100*x2), etc.

If we are trying to minimize the cost function in the multidimensional
space of coefficients, we will get an infinite diagonal **groove (ridge)**
for a*x1 = x2. Any place at the bottom of this ridge is equally good for us.
So we will get a big variability in possible values of coefficients for x1 & x2.
In other words, the solution will be unstable.

We can reduce this variability by adding additional regularization factor to the loss function as a sum of squares of coeffitients. When we minimizing this expression, bigger "lambda" value will cause smaller "beta" values, thus keeping the coefficients "beta" from becoming too large.
This is called "**Ridge Regression**".
It changes the infinitely long ridge into a local minimum.

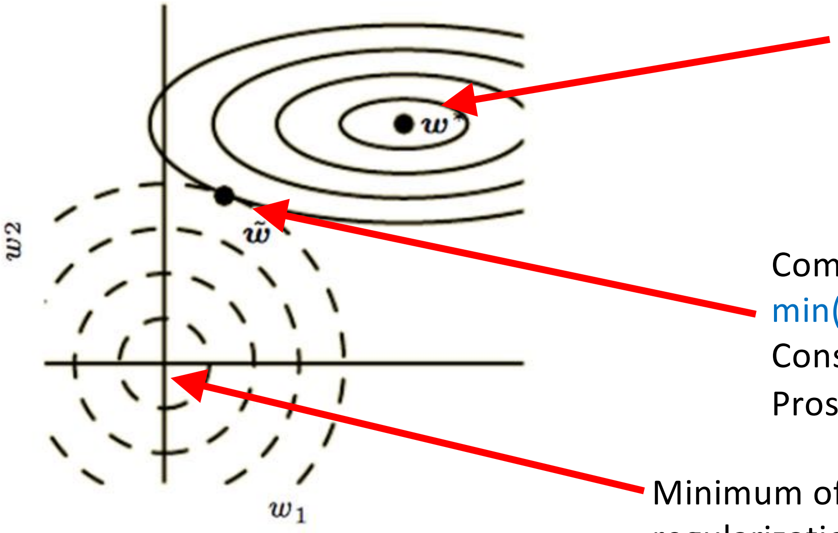
Why Ridge Regression is called Ridge ...



$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2=\mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2$$

One way to do avoid overfitting can be to avoid big "weights" in the model.

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$$



Minimum “cost” without regularization

Compromize
min(cost + penalty)
Cons: bias (shift),
Pros: low variance

Minimum of the “penalty” regularization term

Figure 7.1: An illustration of the effect of L^2 (or weight decay) regularization on the value of the optimal $\boldsymbol{w}$. The solid ellipses represent contours of equal value of the unregularized objective. The dotted circles represent contours of equal value of the L^2 regularizer. At the point $\tilde{\boldsymbol{w}}$, these competing objectives reach an equilibrium. In the first dimension, the eigenvalue of the Hessian of J is small. The objective function does not increase much when moving horizontally away from $\boldsymbol{w}^*$. Because the objective function does not express a strong preference along this direction, the regularizer has a strong effect on this axis. The regularizer pulls w_1 close to zero. In the second dimension, the objective function is very sensitive to movements away from $\boldsymbol{w}^*$. The corresponding eigenvalue is large, indicating high curvature. As a result, weight decay affects the position of w_2 relatively little.

Regularization - LASSO, Elastic Net

In **Ridge Regression** we add a "**regularization**" term as **sum of squares** (**L2 norm**).

In **LASSO Regression** we add a "**regularization**" term as **sum of abs values** (**L1 norm**).

LASSO = Least Absolute Shrinkage and Selection Operator.

LASSO favors solutions on the tips of the bounding surface for coefficients, where some of coefficients are zero. Thus it favors "sparse" solutions.

Elastic Net - A combination of Ridge Regression and LASSO.

Why LASSO favors sparse solutions

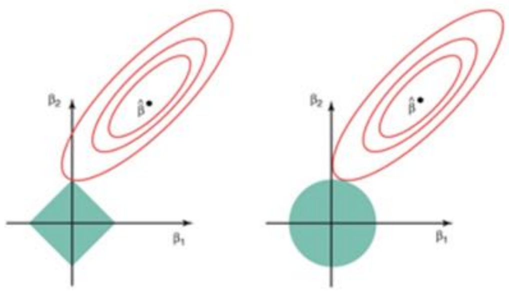


Fig 6.7 from "An Introduction to Statistical Learning" by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani (Springer 2013-2017)

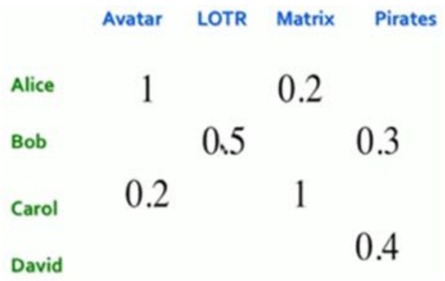
Contours of the error and constraint functions for the lasso (left) and ridge regression (right). The solid blue areas are the constraint regions, $| \beta_1 | + | \beta_2 | \leq s$ and $\beta_1^2 + \beta_2^2 \leq s$, while the red ellipses are the contours of the RSS (Residual Sum of Squares).

Recommender Systems

1. Content based (CB) – based on user history
2. Collaborative filtering (CF) – based on other users
3. Latent factor based (users and products presented as latent vectors)

- Lists of favorites, “essential” items
- List of “Top 10”, Most popular, New items
- Tailored to individual users

Utility Matrix:



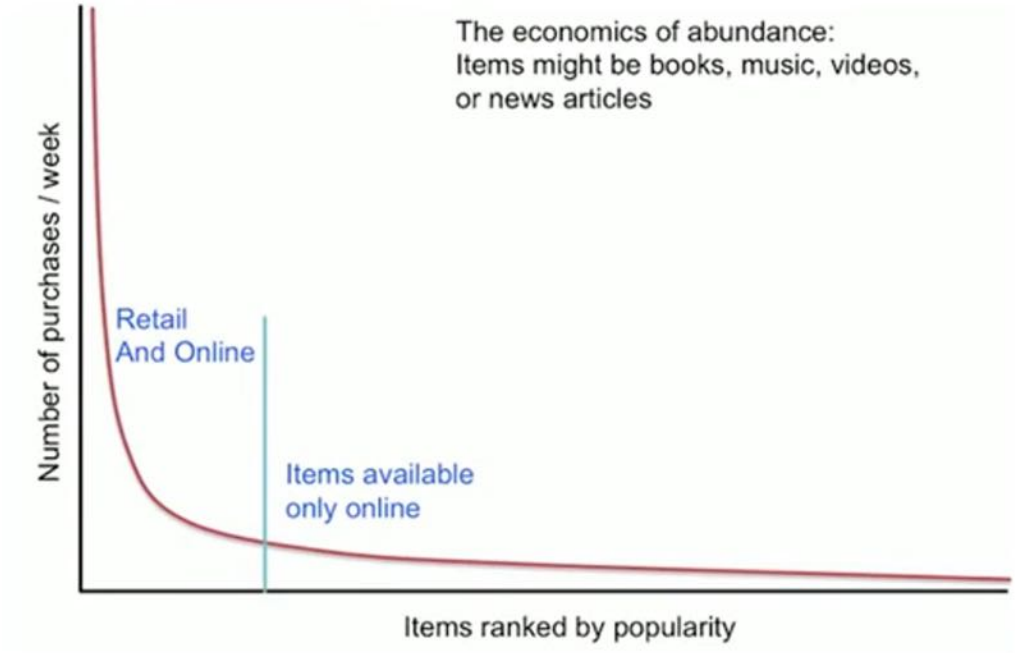
	Avatar	LOTR	Matrix	Pirates
Alice	1		0.2	
Bob		0.5		0.3
Carol	0.2		1	
David				0.4

Ratings:
- Explicit reviews and ratings
- Implicit from user buying behavior

Problem: Utility Matrix is very sparse, most people haven’t rated most items.
Problem: New items have no ratings, new users have no ratings

Long Tail

Total sales in long-tail may be big,
but sales of each individual item are small
and each item is almost invisible.
Recommendations help to sell items in "long tail".

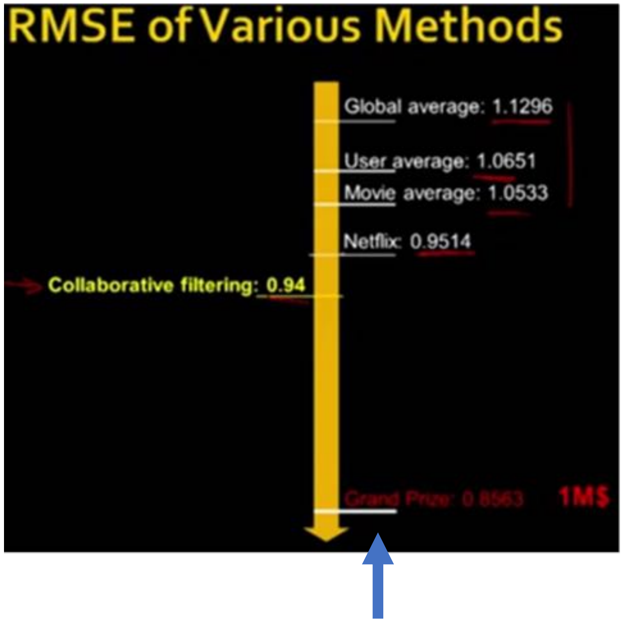


Latent Factor Recommender Systems

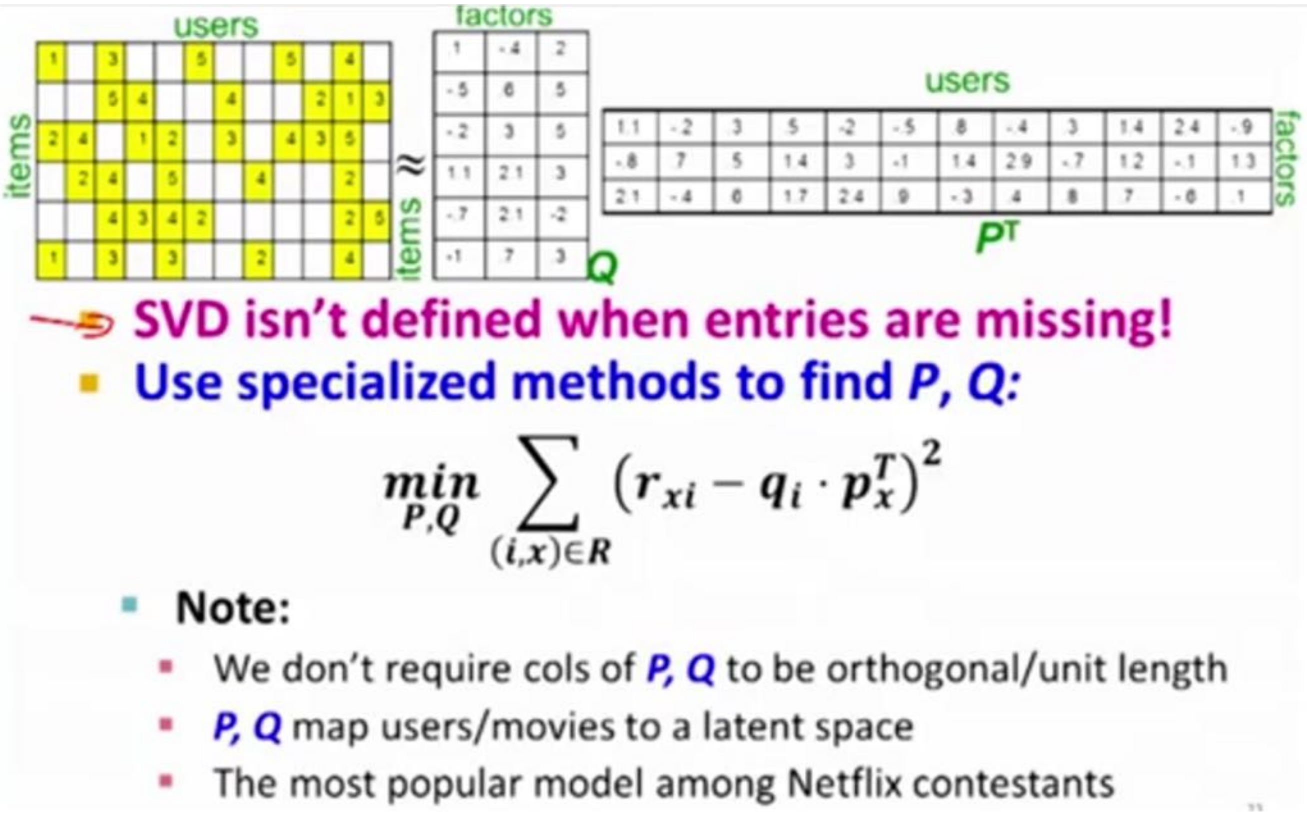
Each user is represented as a K-dimensional vector of latent variables.
Each movie is also represented as a K-dimensional vector (in picture below K=3)
We decompose original utility matrix into a product of 2 matrixes of latent vectors.
This is “matrix factorization”.
To find latent vectors, we solve minimization problem.

Dimensionality Reduction

If some rows/columns of a matrix can be presented as linear combinations of others, this decreases the “rank” of the matrix. We can then reduce the matrix to a smaller matrix (for example, convert from 3D to 2D plane).



Netflix prize (2009)



$$\begin{array}{c|cccccccccccc} & & & & & & \text{users} & & & & & & \\ \hline \text{items} & 1 & & 3 & & & 5 & & & 5 & & 4 & \\ & & & 5 & 4 & & & 4 & & & 2 & 1 & 3 \\ & 2 & 4 & & 1 & 2 & & 3 & & 4 & 3 & 5 & \\ & & 2 & 4 & & 5 & & & 4 & & & 2 & \\ & & & 4 & 3 & 4 & 2 & & & & & 2 & 5 \\ & 1 & & 3 & & 3 & & & 2 & & & 4 & \end{array} \approx \begin{array}{c|ccc} & & \text{factors} & \\ \hline \text{items} & .1 & -.4 & .2 \\ & -.5 & .6 & .5 \\ & -.2 & .3 & .5 \\ & 1.1 & 2.1 & .3 \\ & -.7 & 2.1 & -2 \\ & -1 & .7 & .3 \end{array} Q$$

$$\begin{array}{c|cccccccccccc} & & & & & & \text{users} & & & & & & \\ \hline \text{factors} & 1.1 & -.2 & .3 & .5 & -2 & -.5 & .8 & -.4 & .3 & 1.4 & 2.4 & -.9 \\ & -.8 & .7 & .5 & 1.4 & .3 & -1 & 1.4 & 2.9 & -.7 & 1.2 & -.1 & 1.3 \\ & 2.1 & -.4 & .6 & 1.7 & 2.4 & .9 & -.3 & .4 & .8 & .7 & -.6 & .1 \end{array} P^T$$

- SVD isn’t defined when entries are missing!
- Use specialized methods to find ***P, Q:***

$$\min_{P,Q} \sum_{(i,x)\in R} \left(r_{xi} - q_i \cdot p_x^T\right)^2$$

- **Note:**
  - We don’t require cols of ***P, Q*** to be orthogonal/unit length
  - ***P, Q*** map users/movies to a latent space
  - The most popular model among Netflix contestants

Ranking

- https://en.wikipedia.org/wiki/Ranking
- Ranking is a week ordering of items by assigning a rank (a positive integer number) to each item.
- Rank is inversely related to the value.
  We say that the ranking is "higher" if the value of the assigned rank integer number is "lower".
  So, for example, rank 2 is higher than rank 3. The highest possible rank is 1.
- Difference between Rating and Ranking:
  .. Rating usually means a category or a range of values. "a classification or ranking of someone or something based on a comparative assessment of their quality, standard, or performance."
  .. Ranking = ordering number between values
- Ranking used in statistics (wikipedia)
- Predictive vs. Earned (historic) Ranking
- Real-time ranking
- Top-N
- Rank using multiple features
- Numeric, extract sentiment, etc.
- Loss functions: Ranking Loss, Contrastive Loss, Margin Loss, Triplet Loss, Hinge Loss, etc.
- Ranking models using Neural Networks
- Learning to rank (multiple methods): https://en.wikipedia.org/wiki/Learning_to_rank
- xgb.XGBRanker: https://medium.com/predictly-on-tech/learning-to-rank-using-xgboost-83de0166229d
- sklearn: https://towardsdatascience.com/learning-to-rank-with-python-scikit-learn-327a5cfd81f
- Microsoft: https://lucidworks.com/post/abcs-learning-to-rank/ - RankNet, LambdaRank, LambdaMART
- scikit-network: https://scikit-network.readthedocs.io/en/latest/reference/ranking.html
- sklearn-ranking: https://pypi.org/project/sklearn-ranking/
- LTR: https://everdark.github.io/k9/notebooks/ml/learning_to_rank/learning_to_rank.html

LightGBM As Ranker

- https://medium.com/@raghavbhutani41/gradient-boosting-ranking-algorithm-lightgbm-667050dddaaf
- https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMRanker.html

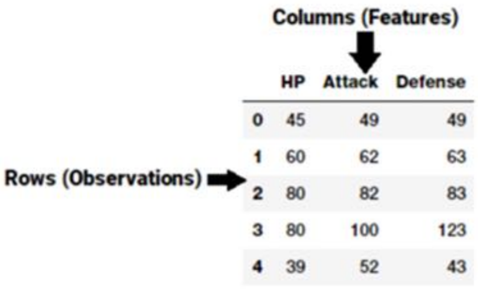
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,       y,       test_size=0.2, random_state=1)
X_train, X_val,  y_train, y_val  = train_test_split(X_train, y_train, test_size=0.2, random_state=1)

query_train = [X_train.shape[0]]
query_val   = [X_val.shape[0]]
query_test  = [X_test.shape[0]]

import lightgbm as lgb
gbm = lgb.LGBMRanker()
gbm.fit( X_train, y_train,
         group=query_train,
         eval_set=[(X_val, y_val)],
         eval_group=[query_val],
         eval_at=[5, 10],
         early_stopping_rounds=50 )
```

Feature Engineering

"Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data." - Dr. Jason Brownlee

Cleaning Data and Feature Engineering typically take 80% of time of a Data Science Project!

A feature is typically
a column in a data set



Columns (Features)

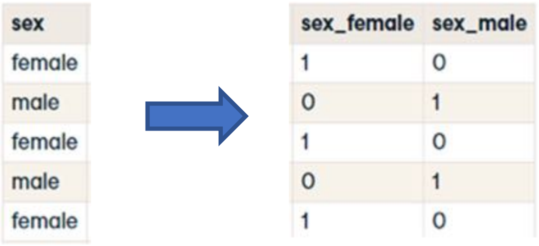
Rows (Observations)

	HP	Attack	Defense
0	45	49	49
1	60	62	63
2	80	82	83
3	80	100	123
4	39	52	43

Typical Operations:

- **Removing obviously non-relevant data** (empty or not-changing columns/rows, row_id, etc.)
- **Synthetic Features** (Example: Income / Price)
- **Select relevant columns** (Kolmogorov-Smirnov test, etc.)
- **Sparse features** – consider converting/combining (Example: grouping penny-stocks together)
- **Impute missing data** (with mean, median, most frequent, KNN, extrapolation or interpolation, MICE (Multivariate Imputation by Chained Equation), etc.
- **Outliers** (use box-plots and scatterplots to find), filter-out or change
- **Collinearity** – reduce (combine features, use PCA (Principal Component Analysis)
- **Factor Analysis** (finding underlying common factor for several features) – reduce number of features
- **Aggregating, Binning / Bucketing** - df[col] = pd.qcut(df[col], 10)
- **Grouping** of some features together (numeric - sum() or mean(), categorical – by frequency)
- **Transformations & Interactions** (see H2O.ai Driverless AI as a good example)
- **One-hot Encoding** (use pd.get_dummies() function in pandas)
- **Split** features (for example, split text)
- **Scaling** (Normalization or Standardization)
- **Dates** – convert to features (Day of week, Month, Year, Day of year, Quarter, diff between dates)
- **Feature Importance** – training with different subsets of features (ex. Random Forest) allows to figure out feature importance – and helps to remove non-important features
- **Text** - extract synthetic features (for example, sentiment (Positive, Neutral, Negative)

One-Hot Encoding



sex
female
male
female
male
female

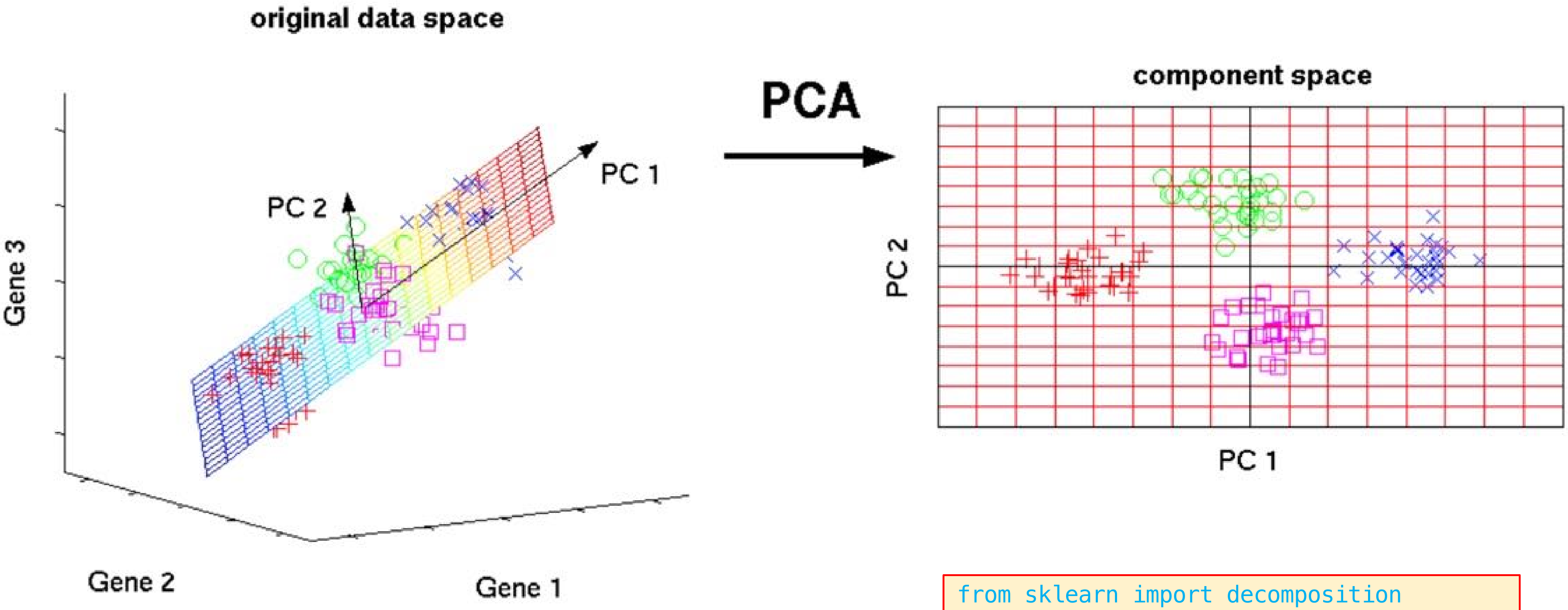
sex_female	sex_male
1	0
0	1
1	0
0	1
1	0

Feature Engineering Tools – sklearn

- **Pandas, NumPy, Matplotlib, Seaborn**

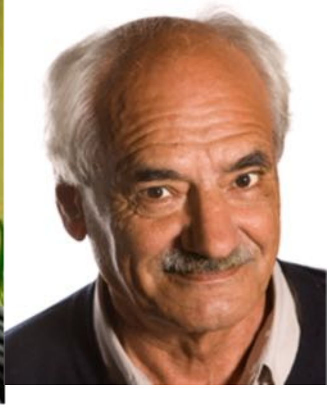
- **Scikit-learn Dataset transformations:** https://scikit-learn.org/stable/data_transforms.html
- **6.1. Pipelines and composite estimators**
- 6.1.1. Pipeline: chaining estimators
- 6.1.2. Transforming target in regression
- 6.1.3. FeatureUnion: composite feature spaces
- 6.1.4. ColumnTransformer for heterogeneous data
- 6.1.5. Visualizing Composite Estimators
- **6.2. Feature extraction**
- 6.2.1. Loading features from dicts
- 6.2.2. Feature hashing
- 6.2.3. Text feature extraction
- 6.2.4. Image feature extraction
- **6.3. Preprocessing data**
- 6.3.1. Standardization, or mean removal and variance scaling
- 6.3.2. Non-linear transformation
- 6.3.3. Normalization
- 6.3.4. Encoding categorical features
- 6.3.5. Discretization
- 6.3.6. Imputation of missing values
- 6.3.7. Generating polynomial features
- 6.3.8. Custom transformers
- **6.4. Imputation of missing values**
- 6.4.1. Univariate vs. Multivariate Imputation
- 6.4.2. Univariate feature imputation
- 6.4.3. Multivariate feature imputation
- 6.4.4. References
- 6.4.5. Nearest neighbors imputation
- 6.4.6. Marking imputed values
- **6.5. Unsupervised dimensionality reduction**
- 6.5.1. PCA: principal component analysis
- 6.5.2. Random projections
- 6.5.3. Feature agglomeration
- **6.6. Random Projection**
- 6.6.1. The Johnson-Lindenstrauss lemma
- 6.6.2. Gaussian random projection
- 6.6.3. Sparse random projection
- **6.7. Kernel Approximatio**n
- 6.7.1. Nystroem Method for Kernel Approximation
- 6.7.2. Radial Basis Function Kernel
- 6.7.3. Additive Chi Squared Kernel
- 6.7.4. Skewed Chi Squared Kernel
- 6.7.5. Polynomial Kernel Approximation via Tensor Sketch
- 6.7.6. Mathematical Details
- **6.8. Pairwise metrics, Affinities and Kernels**
- 6.8.1. Cosine similarity
- 6.8.2. Linear kernel
- 6.8.3. Polynomial kernel
- 6.8.4. Sigmoid kernel
- 6.8.5. RBF kernel
- 6.8.6. Laplacian kernel
- 6.8.7. Chi-squared kernel
- **6.9. Transforming the prediction target (y)**
- 6.9.1. Label binarization
- 6.9.2. Label encoding

PCA = Principal Component Analysis



```
from sklearn import decomposition
# ...
pca = decomposition.PCA(n_components=3)
pca.fit(X)
X = pca.transform(X)
```

Vladimir N. Vapnik



Alexey Chervonenkis

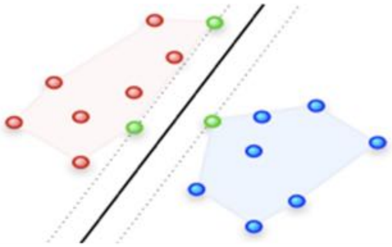
Bernhard E. Boser

Isabelle M. Guyon

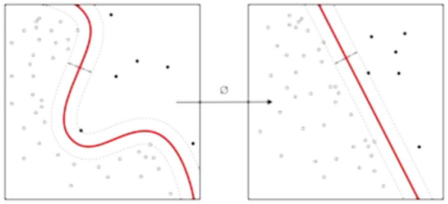
Support Vector Machine (SVM)

The original SVM algorithm was invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963. In 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes.

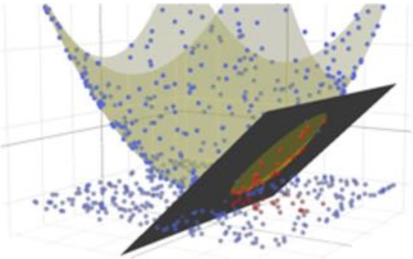
A vanilla **SVM** is a type of **linear separator**. We draw a straight line through our data down the middle to separate it into two classes.



If we can't draw a line through data, we can transform the data into a feature representation where the separation becomes possible.



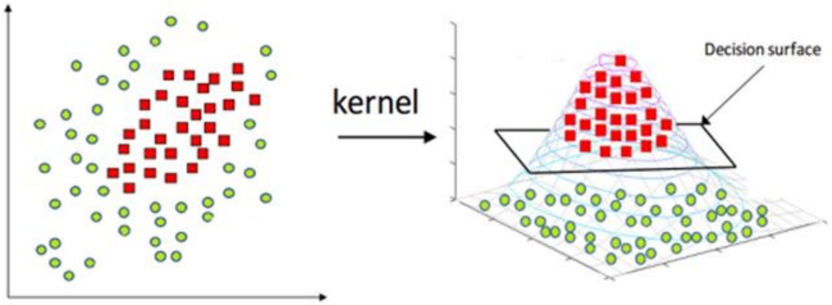
“kernel trick” – use “kernel functions”, which enable them to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space. This operation is often computationally cheaper than the explicit computation of the coordinates. This approach is called the "kernel trick".



SVM Continued ...

A nonlinear classification problem can be converted to a linear classification problem by mapping the input vectors from the input space to a **higher dimensional feature space**.

A **kernel** is a similarity function. It is a function that you, as the domain expert, provide to a machine learning algorithm. It takes two inputs and spits out how similar they are. You use kernels instead of dot-products of your vectors.



Many machine learning algorithms can be expressed entirely in terms of dot products.
Mercer's theorem: Under some conditions, every kernel function can be expressed as a dot product in a (possibly infinite dimensional) feature space.
In many cases, computing the kernel is easy, but computing the feature vector corresponding to the kernel is really really hard. The feature vector for even simple kernels can blow up in size.
For kernels like the RBF (Radial Basis Function) kernel ($k(x,y) = \exp(-(x-y)^2)$,
the corresponding feature vector is infinite dimensional. Yet, computing the kernel is almost trivial.

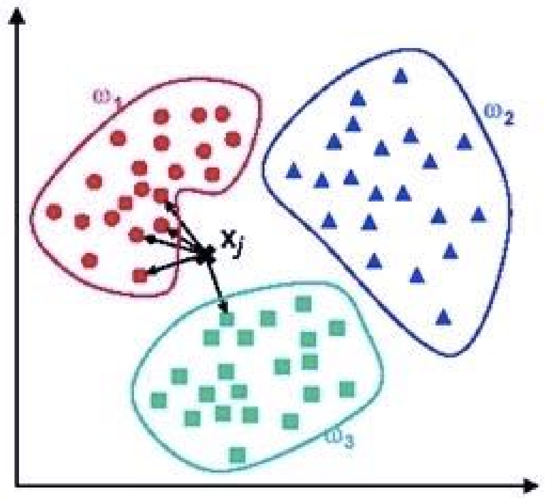
Many machine learning algorithms can be written to *only* use dot products,
and then we can replace the dot products with kernels.
By doing so, we don't have to use the feature vector at all. This means that we can work with highly complex, efficient-to-compute, and yet high performing kernels without ever having to write down the huge and potentially infinite dimensional feature vector.

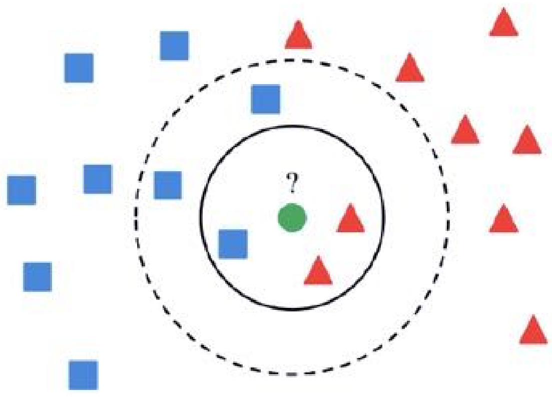
Polynomial kernels can be thought of projecting a vector into a higher dimensional space.
Gaussian kernel can be thought as a projection into infinite-dimensional space because its Taylor expansion has infinite number of elements.

K-Nearest Neighbors (KNN)

Identify new data based on proximity to old data.

"Tell me who your closest five friends are,
and I will tell you who you are"



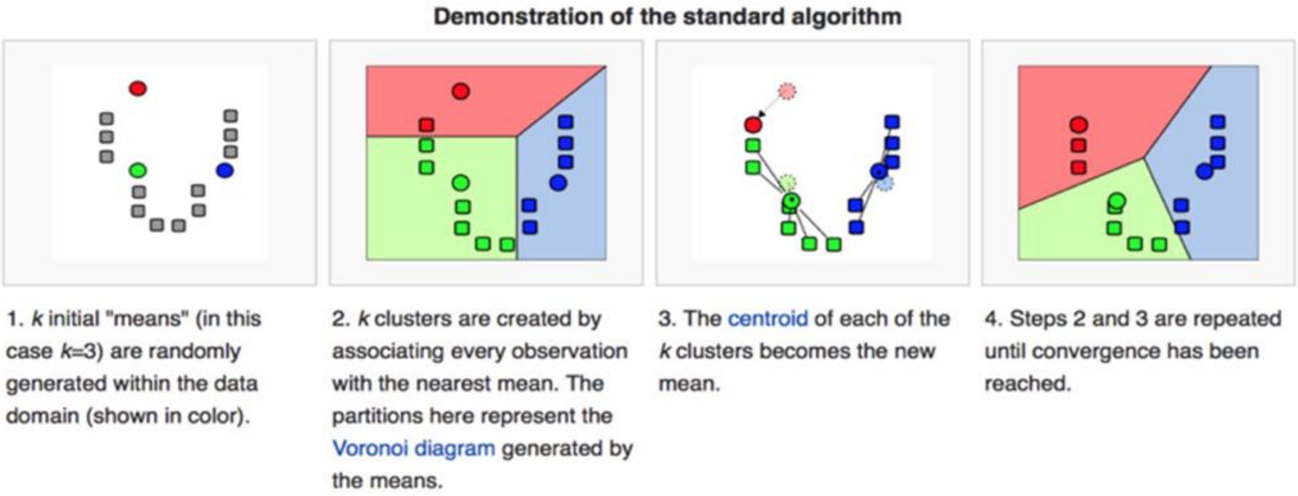


K-means clustering

Partition “n” observations into “k” clusters
in which each observation belongs to the cluster
with the nearest mean

K-means algorithm finds clusters iterativly repeating two steps:

- **Assignment step**:
  Assign each observation to the cluster whose mean has the least squared Euclidean distance
  (this is intuitively the "nearest" mean)
- **Update step**:
  Calculate the new means as centroids of the observations in the new clusters.



Demonstration of the standard algorithm

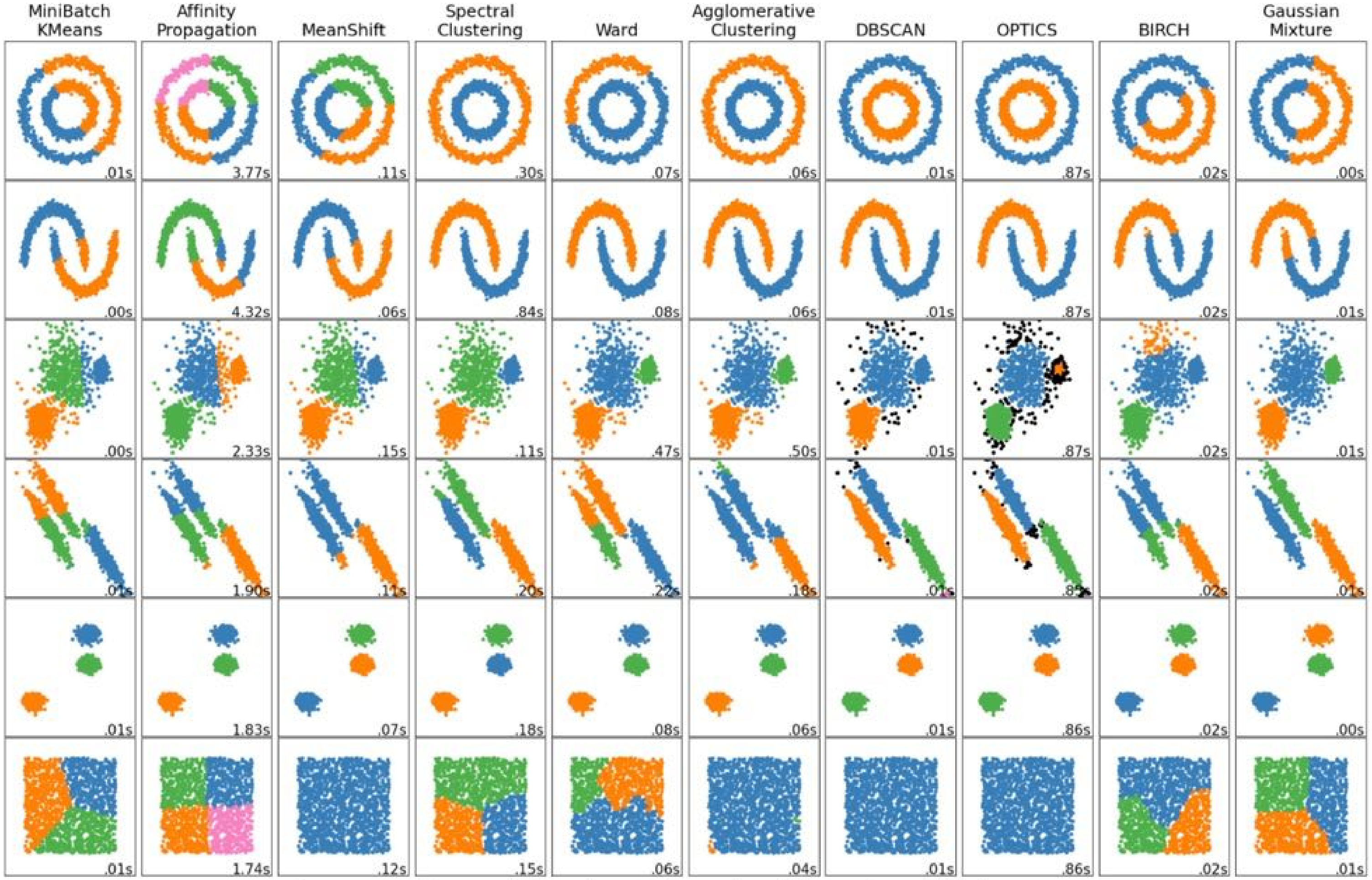
1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

3. The centroid of each of the *k* clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

Clustering Algorithms



A comparison of the clustering algorithms in scikit-learn