

Machine Learning in Finance

Applied Machine Learning Days 2019

Lausanne

Miquel Noguer i Alonso PhD



Miquel Noguer-Alonso PhD – Bio

Miquel Noguer Alonso. He is a financial markets practitioner with more than 20 years of experience in asset management. He is the Co-Founder of Artificial Intelligence Finance Institute. Head of Development at Global AI (Big Data Artificial Intelligence in Finance company) and the Deputy Editor of The Journal of Machine Learning in Finance

He worked for UBS AG (Switzerland) as Executive Director. He was member of European Investment Committee for the last 10 years. He worked as a Chief Investment Officer and CIO for Andbank from 2000 to 2006. He started his career at KPMG.

He is Adjunct Professor at Columbia University teaching Asset Allocation, Big Data in Finance and Fintech. He is also Professor at ESADE teaching Hedge Fund, Big Data in Finance and Fintech. He taught the first Fintech and Big Data course at the London Business School in 2017.

He received an MBA and a Degree in business administration and economics in ESADE in 1993. In 2010 he earned a PhD in quantitative finance with a Summa Cum Laude distinction (UNED – Madrid Spain). He completed a Postdoc in Columbia Business School in 2012.

His research interests range from Artificial Intelligence and Big Data in Finance, machine learning, algorithmic trading and Fintech.

Artificial Intelligence Finance Institute

MISSION

The Artificial Intelligence Finance Institute's (AIFI) mission is to be the world's leading educator in the application of artificial intelligence to investment management, capital markets and risk. We offer one of the industry's most comprehensive and in-depth educational programs, geared towards investment professionals seeking to understand and implement cutting edge AI techniques.

Taught by a diverse staff of world leading academics and practitioners, the AIFI courses teach both the theory and practical implementation of artificial intelligence and machine learning tools in investment management. As part of the program, students will learn the mathematical and statistical theories behind modern quantitative artificial intelligence modeling. Our goal is to train investment professionals in how to use the new wave of computer driven tools and techniques that are rapidly transforming investment management, risk management and capital markets.

Artificial Intelligence Finance Institute Certificate

The course globally offered online expounds on the theory and implementation of artificial intelligence in finance. Students are expected to learn the mathematical and statistical theories behind modern quantitative artificial intelligence modeling. The course will provide education on the theory and practical application of artificial intelligence in finance through exposure to world leading practitioners and academics.

PROGRAMME OF THE COURSE

Course Logistics

Artificial Intelligence in Investment Management Certificate

25% Super Early Bird Discount until Friday February 8th 2019

Lectures: New York City and globally online. March 5 – June 11 2019.

Tuesday and Thursday. 06.00 pm – 09.00 pm

75 Hours: Lectures + Practice + Speakers

Evaluation: Projects + Final Exam

Course Fee: \$9,500

Machine Learning in Finance

1.	Quantitative Finance	Slide 9
2.	Definitions	Slide 32
3.	Modeling	Slide 52
1.	General Framework.....	Slide 53
2.	Pre-processing and Feature Selection.....	Slide 61
3.	Model evaluation : Performance Metrics	Slide 67
4.	Cross-Validation	Slide 84
5.	Model Selection	Slide 94
4.	Supervised Learning.....	Slide 104
1.	Supervised Learning – Classification	Slide 110
1.	Classification Logistic Regression	Slide 120
2.	Classification - K – Nearest	Slide 125
3.	Classification - Decision Trees	Slide 135
4.	Classification – Support Vector Machines.....	Slide 155
5.	Classification – Ensembles.....	Slide 177

Machine Learning in Finance

4.	Supervised Learning	
2.	Supervised Learning – Regressions	Slide 228
1.	Regressions Models	Slide 229
2.	Regressions – Modern Regression.....	Slide 237
3.	Regressions – Non Linear.....	Slide 248
4.	Regressions – Neural Networks	Slide 251
5.	Unsupervised Learning	Slide 292
1.	Clustering.....	Slide 297
2.	Principal Component Analysis.....	Slide 305
3.	Auto-Encoders.....	Slide 270

Machine Learning in Finance

6. Deep Learning.....	Slide 321
1. Mathematics of Deep Learning	Slide 325
2. Deep learning Example	Slide 333
3. Deep Learning Rationale	Slide 358
4. Deep Learning Training	Slide 363
1. Activation Functions.....	Slide 367
2. Adaptive Learning Rate	Slide 376
3. Drop – Out	Slide 382
5. Neural Networks with Memory.....	Slide 389
6. Deep Learning Finance Applications.....	Slide 411
1. Auto – Encoders Smart Indexing	Slide 417
2. LSTM's for Stock Prediction.....	Slide 428
7. Reinforcement Learning.....	Slide 436
8. Natural Language Processing.....	Slide 443

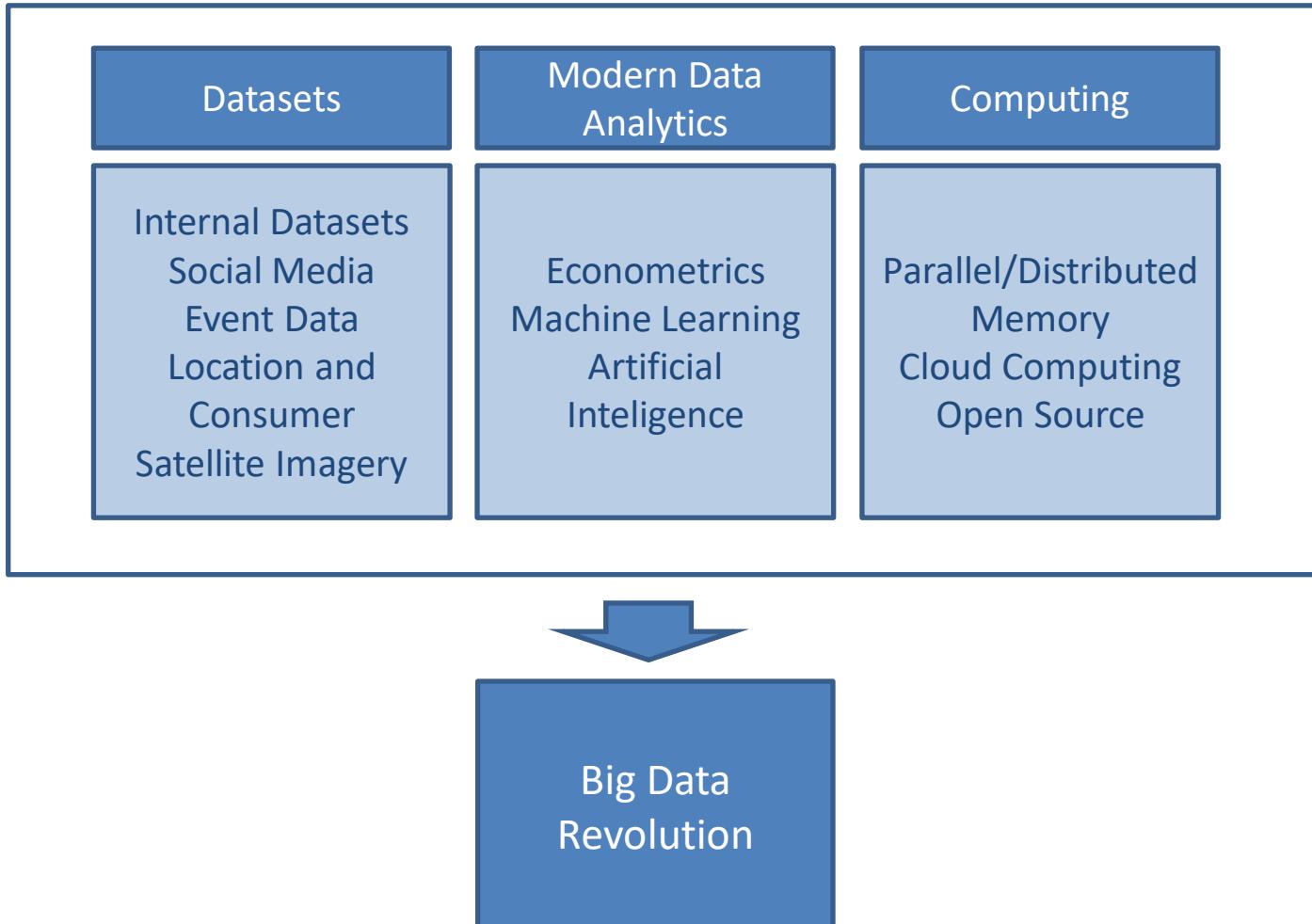
Machine Learning in Finance

- 8. Code.....Slide 484
- 9. Concluding Remarks.....Slide 496

Machine Learning in Finance

1 - Quantitative Finance

Big Data Revolution in Finance



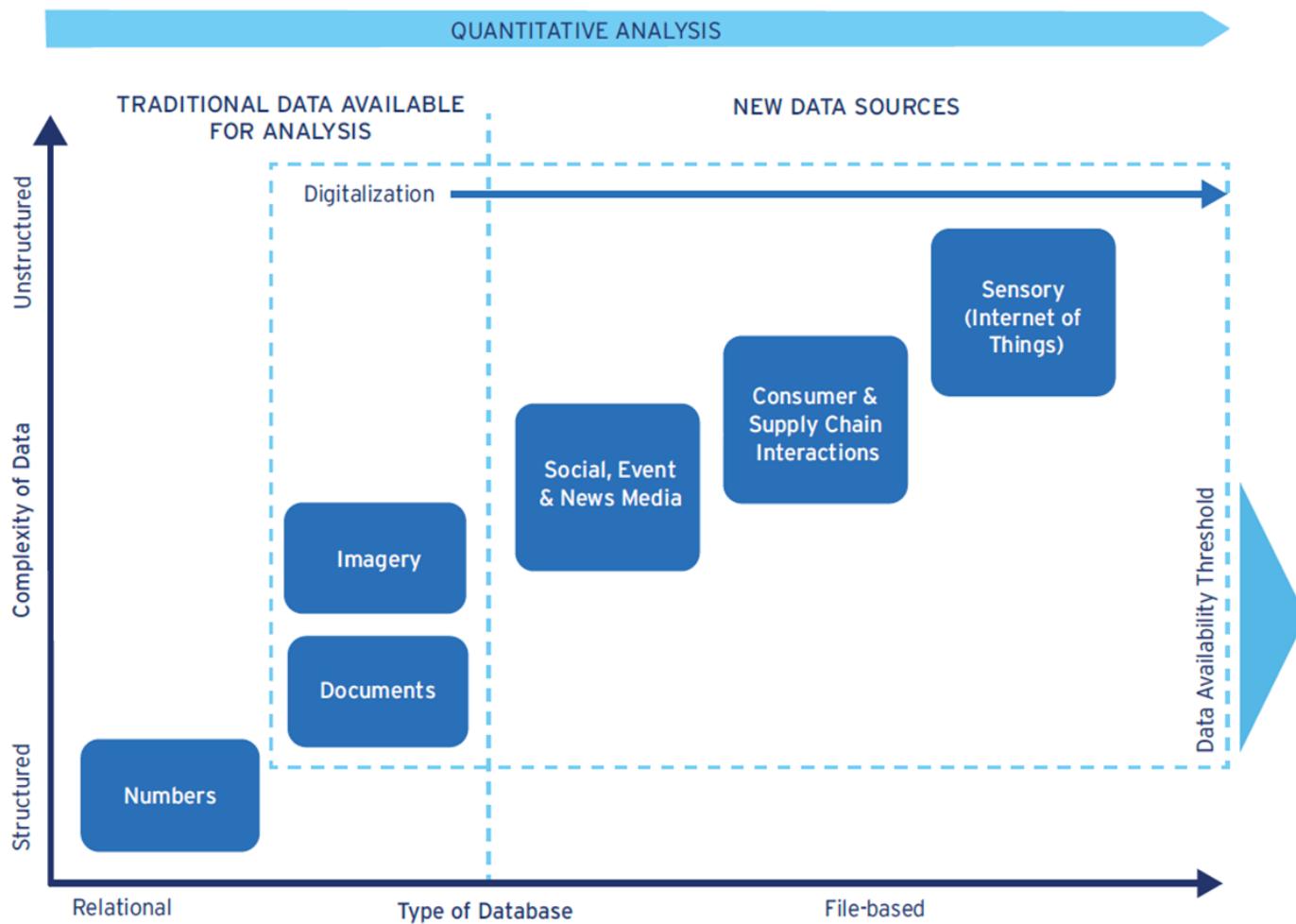
Highlighted Data Providers Organized by Data Set

Financial Markets Bloomberg Reuters Market Data Feeds CRSP, Compustat, I/B/E/S	Consumer Data Yoddle Mint	Social Media Yelp Pinterest Foursquare Facebook Linkedin Twitter	Event Data iSentium Dataminr EagleAlpha Minetta Brook RavenPack	Location & Consumer Google Maps Nokia Here Foursquare Pinpoint Placed Airsage
Government Havers Analytics Socrata	Satellite Imagery Terra Bella Orbital insights			

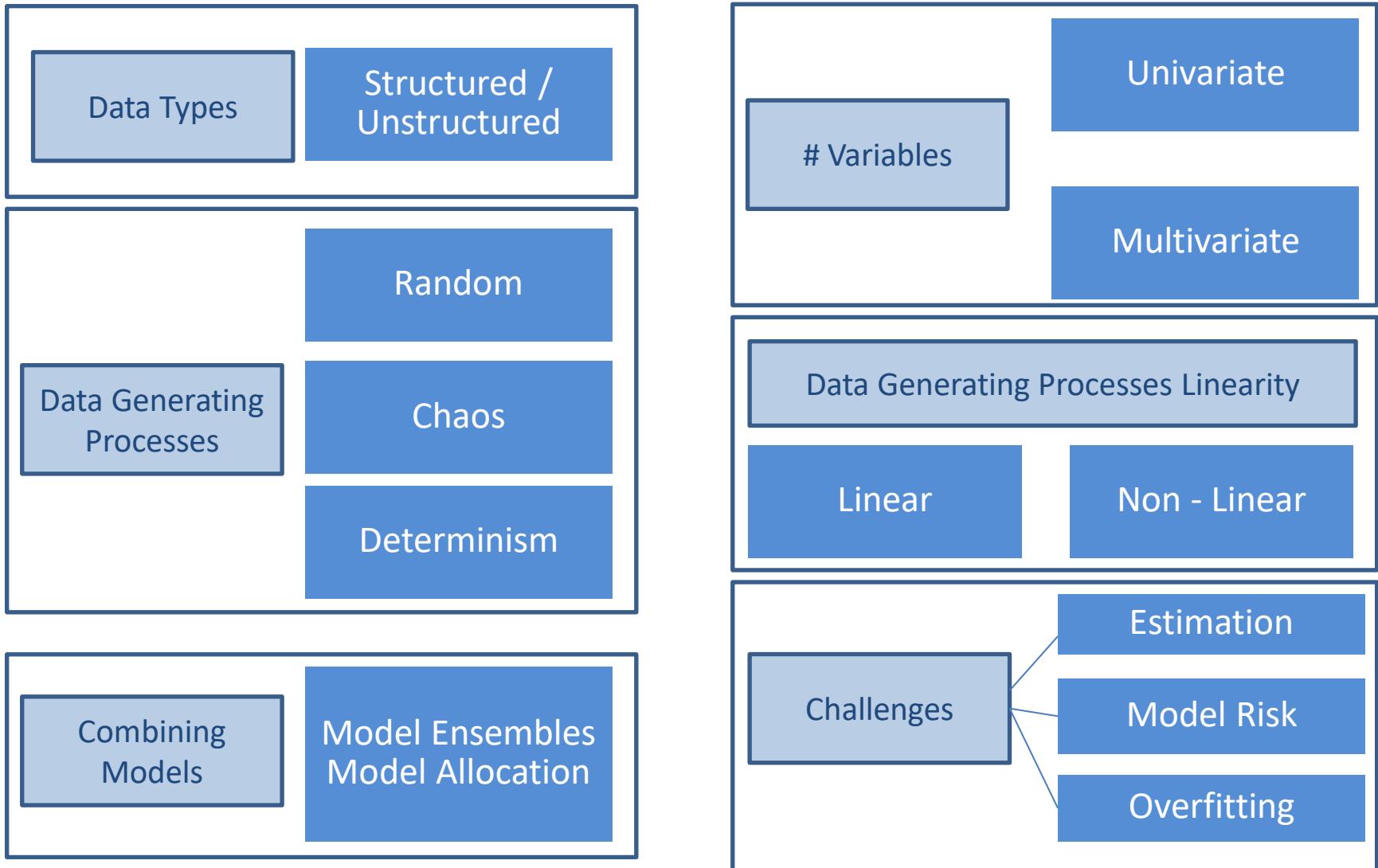
Big Data Infrastructure and Technology

HOSTINGS AND SERVERS	LANGUAGE / TOOL SETS / APPLICATIONS / PLATFORMS	DATABASES	ANALYSIS AND VISUALIZATION TOOLS
<p>Internal Network / Co-Location / Private Cloud</p> <p>Public Cloud (AWS, Azure)</p>	<p>High-level (Java, C, C++)</p> <p>Scripted (Python, Perl, PHP)</p> <p>SQL Interface (Hive)</p> <p>Complex Data Transformation (Pig)</p> <p>Functional Languages (SCALA, F#, Q)</p> <p>Complex Event Processing (Storm, Spark)</p> <p>Messaging (Kafka)</p> <p>Unstructured Data Discovery (Elasticsearch, Solr)</p>	<p>Unstructured / File Based / Open Source / Batch (Hadoop)</p> <p>Unstructured / Document Based / Open Source / Operational (MongoDB, HBase)</p> <p>In-Memory / Structured / High-Speed (KX/kdb+, Cassandra, HBase)</p>	<p>Visualization tools (Tableau, Qlikview, Spotfire, BO, SAS, Micro Strategy, Platfora)</p>

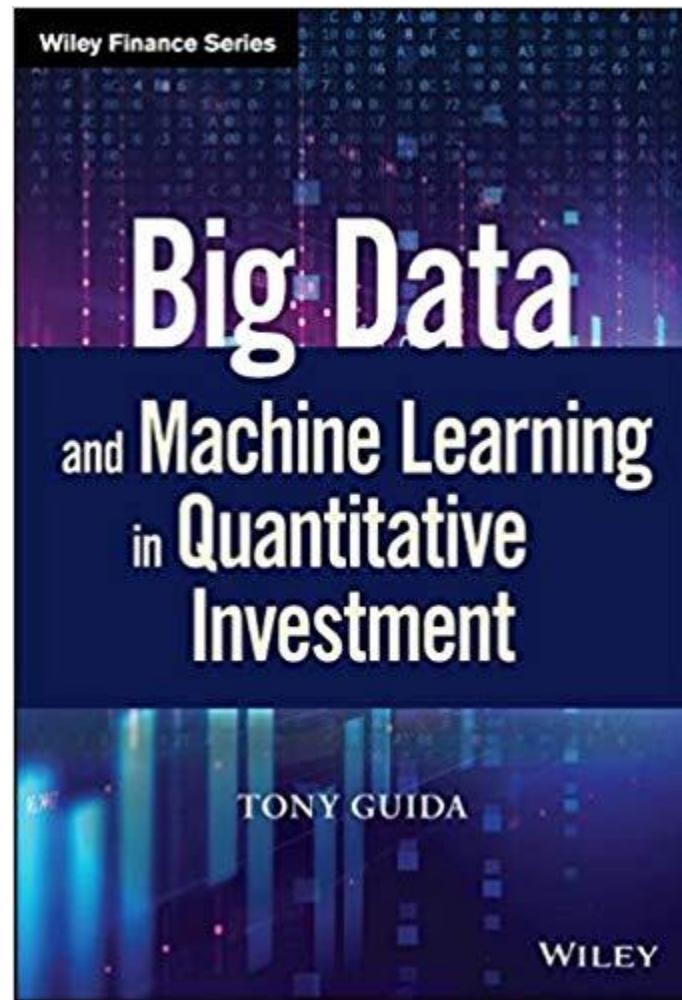
Modern Data Analysis



Modern Data Analysis



Big Data and Machine Learning in Quantitative Investment



Modern Financial Engineering

Complex systems and financial markets

COMPLEX SYSTEMS NORMAL FEATURES

- Feedback
- Non-stationarity
- Many interacting agents
 - Adaptation
 - Evolution
- Single realization
- Open system

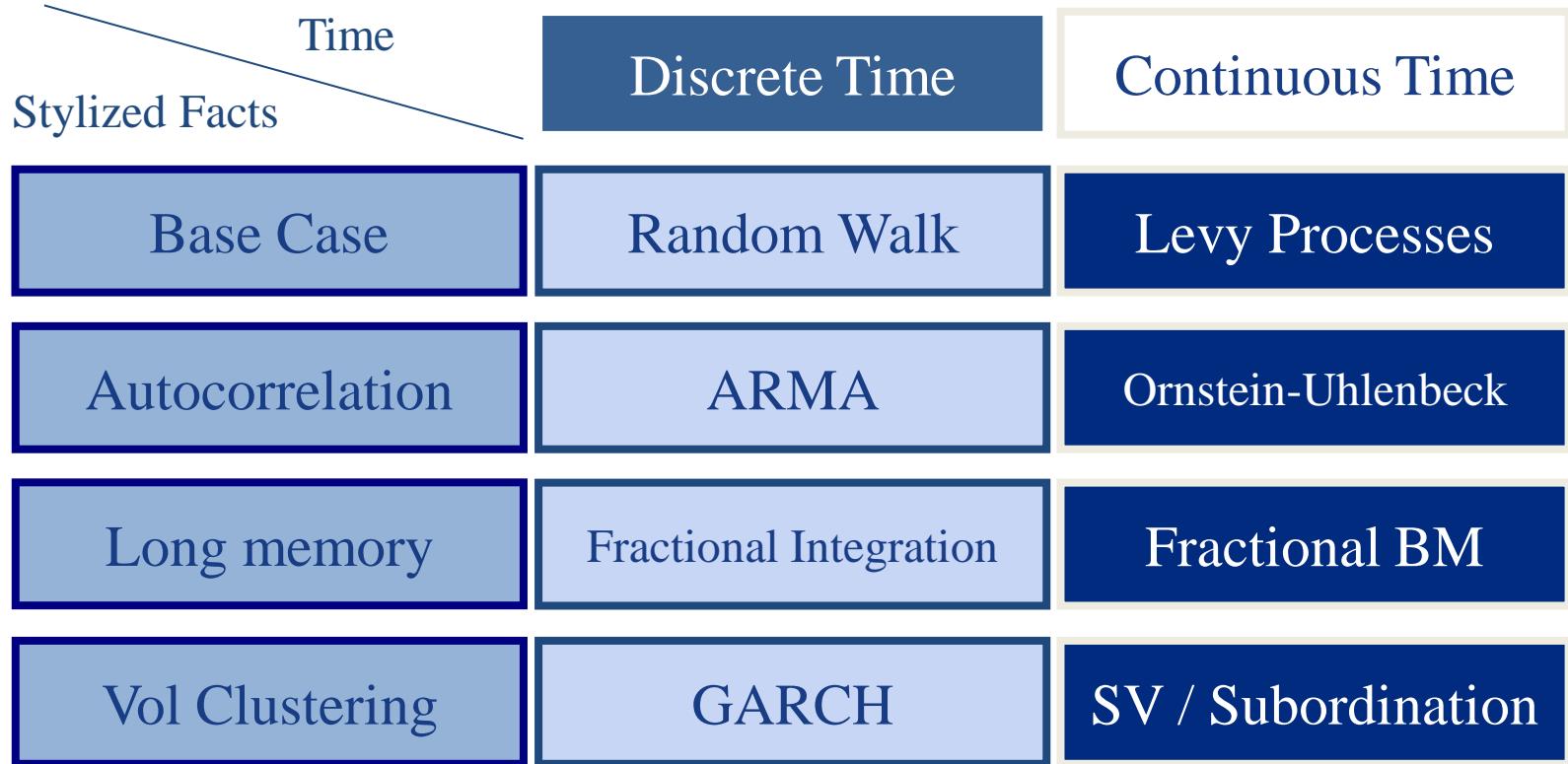
STYLIZED FACTS

- Non-Gaussian Distribution of Returns
- Short-term autocorrelation of return
- Long-term autocorrelation of volatility
 - Bubbles and bursts
 - Stranger than strange attractors in general
 - Hurst exponents

Time series and Modeling

Data Processing			
1	Risk Drivers identification	$\{x_t\}_{t \leq \bar{t}}$	risk drivers data
	Quest for Invariance	$\{\epsilon_t\}_{t=1}^{\bar{t}}$	invariants realizations
	Estimation	f_{ϵ}	invariants distribution
	Projection	$f_{X_{t_{now} \sim t_{hor}}}$	risk drivers path
	Pricing	$f_{\Pi_{t_{now} \rightarrow t_{hor}}}$	instruments ex-ante P&L
Risk Mgmt.			
6	Portfolio aggregation	$v_{h, t_{now}}$	portf value - CVA/liquidity adj.
		$f_{Y_{h, t_{now} \rightarrow t_{hor}}}$	portf. ex-ante performance
		$f_{\Pi_{h, t_{now} \rightarrow t_{hor}}}$	mkt/credit/op portf ex-ante P&L
7	Evaluation	$satis(h)$	satisfaction/risk
		$Y_{h, t_{now} \rightarrow t_{hor}} s$	stress-test
8	Attribution	$\beta_h, f_{U_h, z}$	exposures, factors distribution
		$\{satis_k(h)\}_{k=0}^{\bar{k}}$	satisfaction/risk decomposition
Portfolio Mgmt.			
9	Construction	h^*	optimal portfolio
		$h(i_t)$	optimal policy
10	Execution	$\{\Delta h_{n,t}, \hat{p}_{n,t}\}$	transaction prices/times

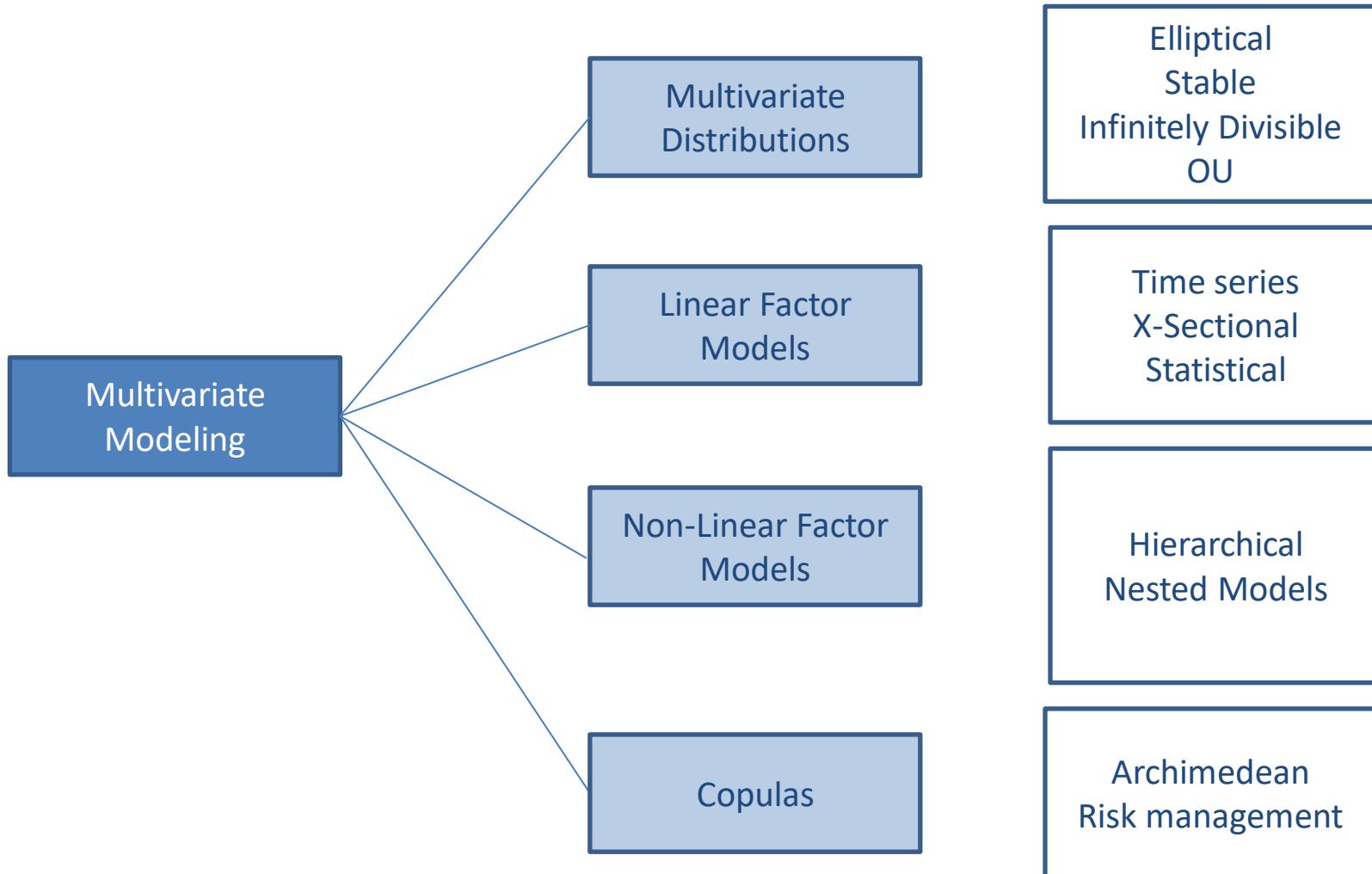
Discrete and Continuous Processes in Finance



Source : Meucci

Multivariate Finance

Finance Multivariate Modeling



Linear Factor Models

$$\mathbf{X} = \mathbf{a} + \mathbf{B}\mathbf{F} + \mathbf{U}$$
$$\left\{ \begin{array}{ll} \mathbf{X} & N \times 1 \\ \mathbf{a} & N \times 1 \\ \mathbf{B} & N \times K \\ \mathbf{F} & K \times 1 \\ \mathbf{U} & N \times 1 \end{array} \right.$$

$$\begin{pmatrix} X_1 \\ \vdots \\ X_n \\ \vdots \\ X_N \end{pmatrix} = \begin{pmatrix} a_1 \\ \vdots \\ a_n \\ \vdots \\ a_N \end{pmatrix} + \begin{pmatrix} b_{1,1} & \cdots & b_{1,k} & \cdots & b_{n,K} \\ \vdots & & \vdots & & \vdots \\ b_{n,1} & \cdots & b_{n,k} & \cdots & b_{n,K} \\ \vdots & & \vdots & & \vdots \\ b_{N,1} & \cdots & b_{N,k} & \cdots & b_{N,K} \end{pmatrix} \begin{pmatrix} F_1 \\ \vdots \\ F_k \\ \vdots \\ F_K \end{pmatrix} + \begin{pmatrix} U_1 \\ \vdots \\ U_n \\ \vdots \\ U_N \end{pmatrix}$$

Linear Factor Models

Linear Factor Models

$$\mathbf{X} = \mathbf{a} + \mathbf{b}\mathbf{Z} + \mathbf{U}$$

$N \times 1$ $N \times 1$ $N \times K$ $N \times 1$
 $K \times 1$

Dominant-Residual

$$(\mathbf{a}, \mathbf{b}, \mathbf{Z}) \equiv \underset{(\alpha, \beta, \bar{\mathbf{Z}}) \in \mathcal{C}}{\operatorname{argmax}} T\{\mathbf{X}, \alpha + \beta \bar{\mathbf{Z}}\}$$

Time-series \mathbf{Z} fully constrained

OLS
 T is R-square
 \mathbf{b} fully unconstrained

Cross-section \mathbf{b} fully constrained

GLS
 T is R-square
 \mathbf{Z} fully unconstrained

Statistical

PCA
 T is R-square
 \mathbf{Z} \mathbf{b} fully unconstrained

Systematic-Idiosyncratic

$$\operatorname{Cv}\{\mathbf{Z}_k, \mathbf{U}_n\} = 0$$

$$\operatorname{Cv}\{\mathbf{U}_m, \mathbf{U}_n\} = 0$$

Pure Exogenous

\mathbf{a} \mathbf{b} \mathbf{Z} fully specified



Factor Analysis $\operatorname{Cv}\{\mathbf{X}\} = \text{low-rank-diagonal}$

Linear Factor Models

Application	Linear Factor Model	Purpose of Linear Factor Model
Multivariate estimation	<p>Invariants</p> $\epsilon_{t \rightarrow t+1} = a + bZ + U$ <p style="text-align: center;">↑ ↑ dominant-residual LFM > truncation > systematic-idiosyncratic LFM</p>	Distribution of invariants f_ϵ is statistically efficient
Asset pricing	<p>Projected P&L of infinite securities</p> $\Pi_{T \rightarrow T+\tau} = a + bZ + U$ <p style="text-align: center;">↑ ↑ systematic-idiosyncratic LFM</p>	<p>Expected excess P&L is ...</p> $E\{\Pi_{T \rightarrow T+\tau}\} - rP_T = b_1\lambda_1 + \dots + b_K\lambda_K$ <p style="text-align: center;">↑ ↑ "beta" with factor... factor risk premium</p>
Alpha-search	<p>securities projected P&L</p> $\Pi_{T \rightarrow T+\tau} = a + bZ + U$ <p style="text-align: center;">↑ ↑ systematic-idiosyncratic LFM</p>	<p>Expected outperformance is...</p> $\alpha = b_1\lambda_1 + \dots + b_K\lambda_K$ <p style="text-align: center;">↑ ↑ ...characteristic signals ... mixing weights</p>
Portfolio optimization	<p>securities projected P&L</p> $\Pi_{T \rightarrow T+\tau} = a + bZ + U$ <p style="text-align: center;">↑ ↑ systematic-idiosyncratic LFM</p>	<p>Covariance is low-rank-diagonal</p> $\underbrace{Cv\{\Pi_{T \rightarrow T+\tau}\}}_{N \times N} = \underbrace{b Cv\{Z\} b'}_{K \times K} + \underbrace{\text{diag}(V\{U\})}_{N \times 1}$
Risk attribution	<p>portfolio projected P&L</p> $\Pi_h = a_h + b_h Z + U_h$ <p style="text-align: center;">↑ ↑ Bottom-up LFM</p>	<p>Portfolio-specific factor exposures</p> $b_{1,h}, \dots, b_{K,h}$

Multivariate Time Series

The basic multivariate time series models based on linear autoregressive, moving average models are:

Vector Autoregression VAR(p)

$$y_t = c + \sum_{i=1}^p \Phi_i y_{t-i} + \varepsilon_t$$

Vector Moving Average VMA(q)

$$y_t = c + \sum_{j=1}^q \Theta_j \varepsilon_{t-j} + \varepsilon_t$$

Vector Autoregression Moving Average VARMA (p,q)

$$y_t = c + \sum_{i=1}^p \Phi_i y_{t-i} + \sum_{j=1}^q \Theta_j \varepsilon_{t-j} + \varepsilon_t$$

Multivariate Time Series

Vector Autoregression Moving Average with a linear time trend VARMAL(p, q)

$$y_t = c + \delta_t \sum_{i=1}^p \Phi_i y_{t-i} + \sum_{j=1}^q \Theta_j \varepsilon_{t-j} + \varepsilon_t$$

Vector Autoregression Moving Average with eXogenous inputs VARMAX(p, q)

$$y_t = c + \beta x_t \sum_{i=1}^p \Phi_i y_{t-i} + \sum_{j=1}^q \Theta_j \varepsilon_{t-j} + \varepsilon_t$$

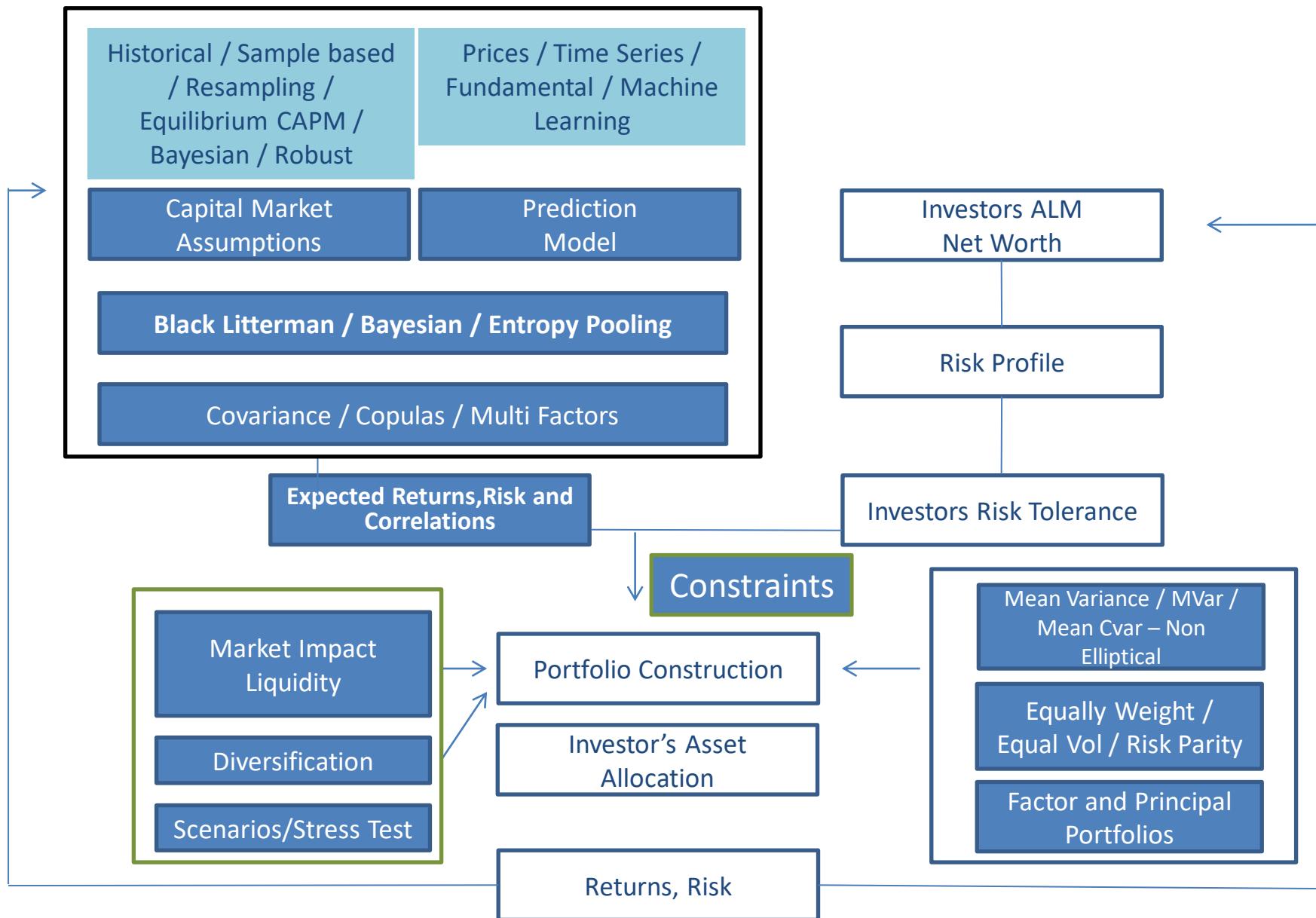
Structural Vector Autoregression Moving Average SVARMA(p, q)

$$\Phi_0 y_t = c + \beta x_t \sum_{i=1}^p \Phi_i y_{t-i} + \sum_{j=1}^q \Theta_j \varepsilon_{t-j} + \Theta_0 \varepsilon_t$$

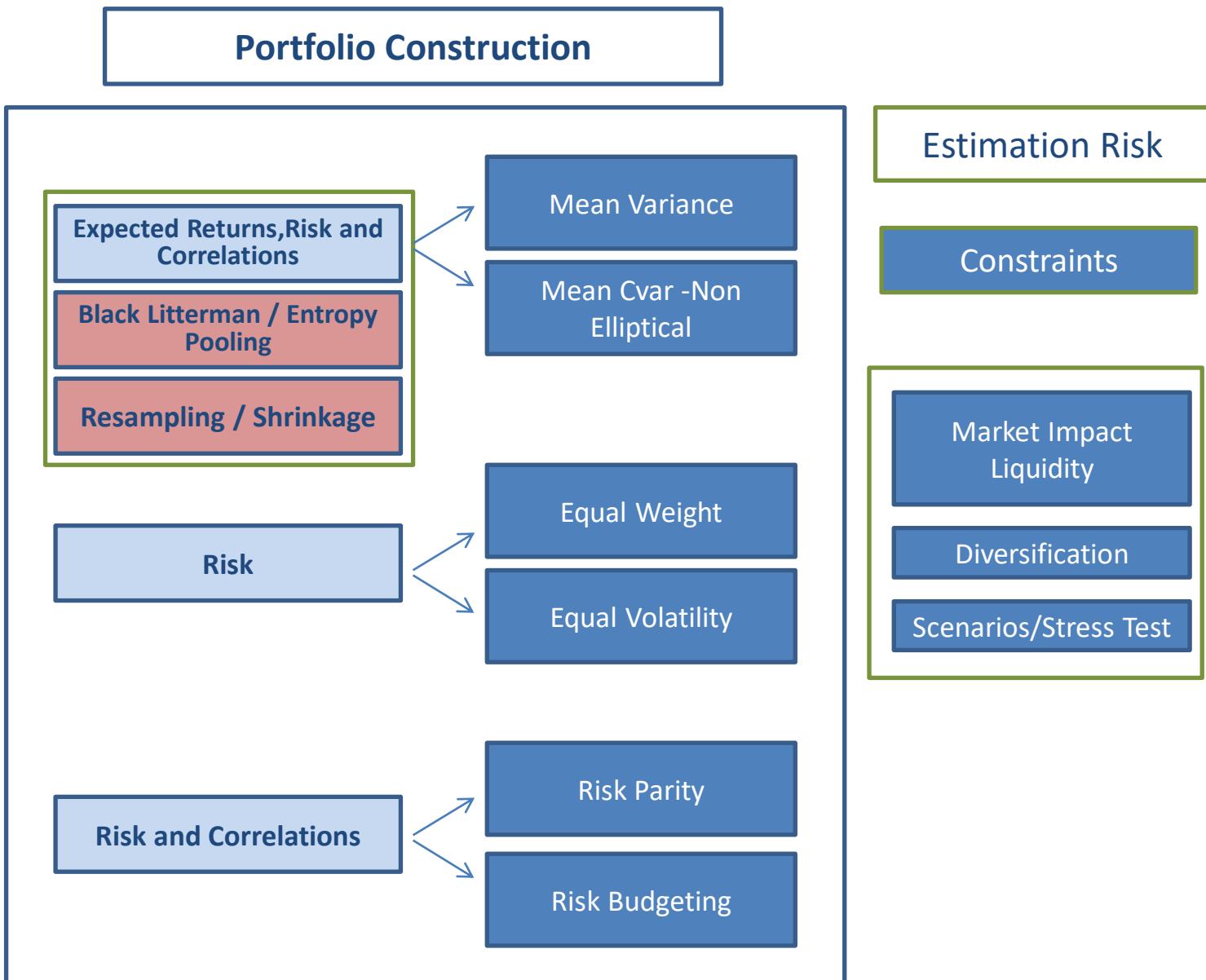
Multivariate Time Series

- y_t is the vector of response time series variables at time t. y_t has n elements.
- c is a constant vector of offsets, with n elements.
- Φ_i are n-by-n matrices for each i. The Φ_i are autoregressive matrices. There are p autoregressive matrices, and some can be entirely composed of zeros.
- ε_t is a vector of serially uncorrelated innovations, vectors of length n. The ε_t are multivariate normal random vectors with a covariance matrix Σ .
- Θ_j are n-by-n matrices for each j. The Θ_j are moving average matrices. There are q moving average matrices, and some can be entirely composed of zeros.
- δ is a constant vector of linear time trend coefficients, with n elements.
- x_t is an r-by-1 vector representing exogenous terms at each time t. r is the number of exogenous series. Exogenous terms are data (or other unmodeled inputs) in addition to the response time series y_t . Each exogenous series appears in all response equations.
- β is an n-by-r constant matrix of regression coefficients of size r. So the product βx_t is a vector of size n.

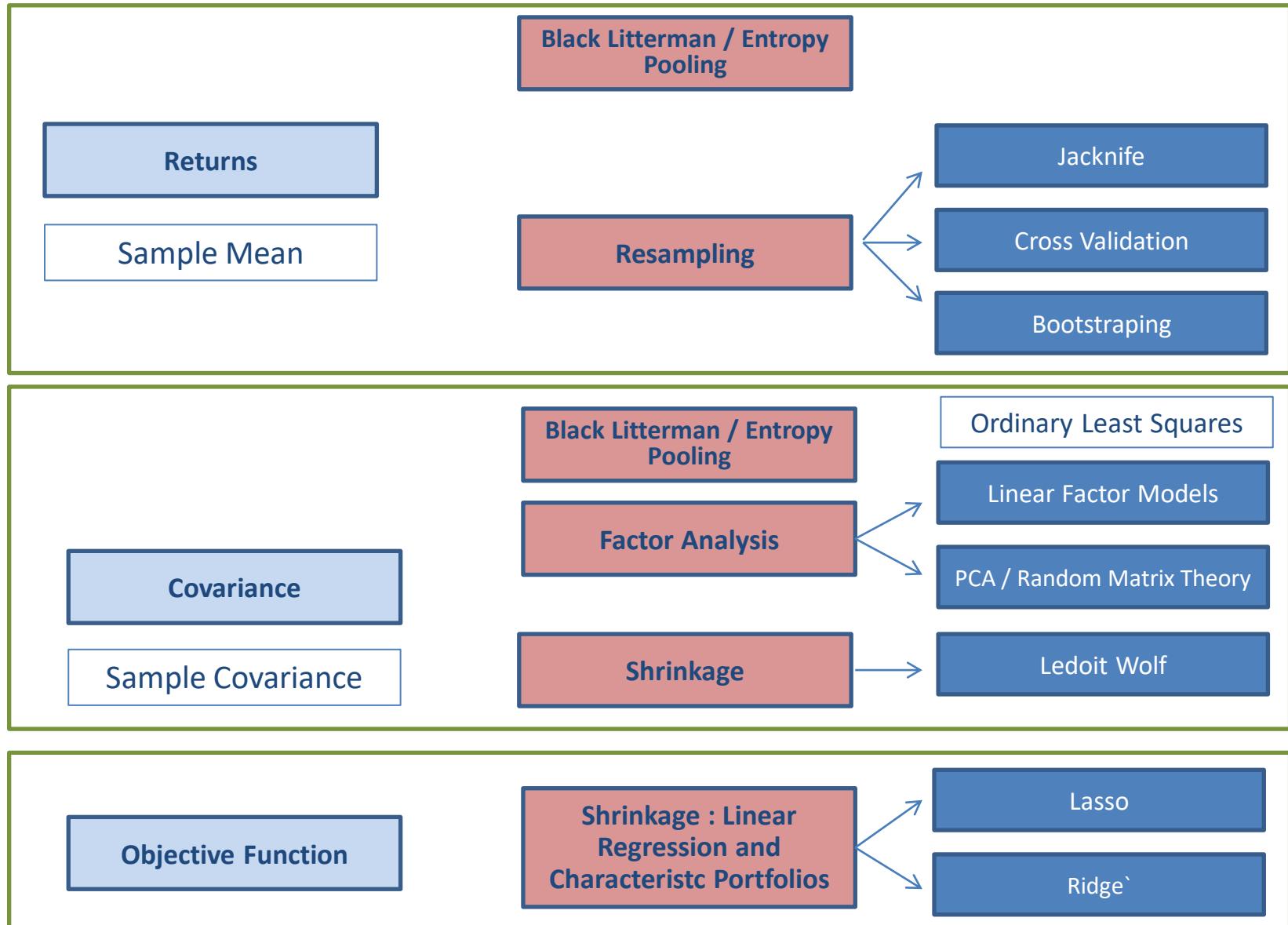
Modern Financial Engineering



Modern Portfolio Construction



Modern Portfolio Construction - Estimation



Portfolio Construction - General Form Equations

Mean-Variance Optimization	$\Lambda \Omega \Lambda w = R \Leftrightarrow \Omega \Lambda w = S = \Lambda^{-1} R$
Minimum Variance	$\Lambda \Omega \Lambda w = 1$
Maximum Diversification	$\Omega \Lambda w = 1$
Equal Volatility	$\Lambda w = 1$
Equally Weighted	$w = 1$
Risk Parity	$(\Lambda w) \circ (\Omega \Lambda w) = 1$
Risk Budgeting	$(\Lambda w) \circ (\Omega \Lambda w) = 1 \circ b$

R asset expected return vector

S asset Sharpe Ratio vector

Λ Asset diagonal volatility matrix

Ω asset correlation matrix

w asset notional weights vector

b asset risk budget vector

Machine Learning in Finance

2. Definitions

Supervised Learning

Predictive or Descriptive

Regression

Classification

Learn Regression Function

$$f: \mathbb{R}^n \rightarrow \mathbb{R}$$

Given:
Inputs and outputs

$$(X_i, Y_i)$$

Unsupervised Learning

Descriptive

Clustering

Representation Learning

Learn Class Function

$$f: \mathbb{R}^n \rightarrow \{1, \dots, k\}$$

Given:
Inputs
(X_i)

Learn Representer function

$$f: \mathbb{R}^n \rightarrow \mathbb{R}^k$$

Given :
Inputs
(X_i)

Reinforcement Learning

Prescriptive

Learn Policy

Inverse Reinforcement Learning

Learn Policy Function

$$f: \mathbb{R}^n \rightarrow \mathbb{R}^k$$

Given :
Tuples

$$(X_i, a_i, X_{i+1}, r_i)$$

Learn Reward Function

$$f: \mathbb{R}^n \rightarrow \mathbb{R}$$

Given :
Tuples

$$(X_i, a_i, X_{i+1})$$

Supervised Learning

Unsupervised Learning

Reinforcement Learning

Regression

Classification

Clustering

Representation Learning

Learn Policy

Inverse Reinforcement Learning

Earnings Prediction

Returns Prediction

Algorithmic Trading

Ratings

Stock Picking

Fraud

AML

Customer segmentation

Stock classification

Factor Modeling Estimation

Regime changes

Trading Strategies

Option replication

Reverse engineering of consumer, trading

Sales / Marketing Activity	Description	Big Data Techniques
Client DNA	<ul style="list-style-type: none"> • Financial DNA • Personal DNA 	Rule Based UnSupervised
Client Servicing	<ul style="list-style-type: none"> • Machine Agent <ul style="list-style-type: none"> • Pricing 	Reinforcement Rule Based Supervised
Channel Management	<ul style="list-style-type: none"> • Offers • Profitability • Sales Channel 	Reinforcement Rule Based Supervised
Marketing	<ul style="list-style-type: none"> • Segmentation • Campaigns 	Reinforcement Rule Based Supervised
Risk Management	<ul style="list-style-type: none"> • Risk Management <ul style="list-style-type: none"> • Fraud • Anti Money Laundering 	Reinforcement Rule Based Supervised / Unsupervised

What is Machine Learning?

- Machine Learning:
 - Designing algorithms that can learn patterns from data (and exploit them)
 - Approach: human supplies training examples, the machine learns
 - Example: Show the machine a bunch of spam and legitimate emails and let it learn to predict if a new email is spam or not
- Machine Learning primarily uses the statistically motivated approach
 - No hand-crafted rules - subtle pattern nuances are often difficult to specify
 - Instead, let the machine figure out the rules on its own by looking at data
 - .. by building statistical models of the data
- The statistical model helps uncover the process which generated the data
- Desirable Property: Generalization
 - The model shouldn't overfit on the training data
 - It should generalize well on unseen (future) test data

Machine Learning Definitions

- Nomenclature: x denotes an input/example/instance, y denotes a response/output/label/prediction
- Supervised Learning: learning with a teacher
 - Given: N labeled training examples $\{(x_1y_1), \dots, (x_Ny_N)\}$
 - Goal: learn mapping f that predicts label y for a test example x
 - Example: Spam classification, webpage categorization
- Unsupervised Learning: learning without a teacher
 - Given: a set of N unlabeled inputs $\{x_1, \dots, x_N\}$
 - Goal: learn some intrinsic structure in the inputs (e.g., groups/clusters)
 - Example: Automatically grouping news stories (Google News)
- Reinforcement Learning: learning by interacting
 - Given: an agent acting in an environment (having a set of states)
 - Goal: learn a policy (state to action mapping) that maximizes agent's reward
 - Example: Automatic vehicle navigation, (computer) learning to play Chess

Supervised Learning

- Given: N labeled training examples $\{(x_1 y_1), \dots, (x_N y_N)\}$
- Goal: learn a model that predicts the label y for a test example x
- Assumption: The training and the test examples are drawn from the same data distribution
- Things to keep in mind:
 - No single learning algorithm is universally good (“no free lunch”)
 - Different learning algorithms work with different assumptions
 - Generalization is particularly important for supervised learning

Supervised Learning

- $f : x \rightarrow y$
- Classification: when y is a discrete variable
 - Discrete variable: takes a value from a discrete set $y \in \{1, \dots, K\}$
 - Example: Category of a webpage (sports, politics, business, science, etc.)
- Regression: when y is a real-valued variable Example: Price of a stock

WORLD U.S. N.Y. / REGION BUSINESS TECHNOLOGY SCIENCE HEALTH

Search Global DealBook Markets

Europe's Debt Crisis Weakens Quarterly Growth

By JACK EWING
Published: August 16, 2011

FRANKFURT — Europe's [sovereign debt crisis](#) threatened to spill over into the broader economy after official figures released Tuesday showed that growth in the euro zone fell to its lowest rate in two years. Germany — the Continent's powerhouse — slowed almost to a standstill.



Enlarge This Image

Most of Europe's main stock indexes lost ground after the data suggested that the debt and economic problems in countries like Greece and Italy were infecting the rest of the 17-country euro zone. The crisis has led a number of governments to sharply cut spending while weathering market turmoil that has damaged business and consumer confidence.



Supervised Learning : Classification

- Problem Types:
- Binary Classification: y is binary (two classes: 0/1 or -1/+1)
 - Example: Spam Filtering (tell whether this email is spam or legitimate)
- Multi-class Classification: y is discrete with one of $K > 2$ possible values
 - Example: Predicting your IEOR grade (e.g., A, A-, B+, B, B-, other)
- Multi-label Classification: When y is a vector of discrete variables
 - Each input x has multiple labels
 - Each element of y is one label (individual labels can be binary/multi-class)
 - Example: Image annotation (each image can have multiple labels)
- Structured Prediction: When y is a vector with a structure
 - Elements of y are not independent but related to each-other
 - Example: Predicting parts-of-speech (POS) tags for a sentence

Input	The	man	ate	the	really	tasty	sandwich
Output	DET	NOUN	VERB	DET	ADV	ADJ	Noun

Supervised Learning : Regression

- Problem Types:
- Univariate Regression: y is a single real-valued number
 - Example: Predicting the future price of a stock
- Multivariate Regression: y is a real-valued vector
 - Each element of y tells the value of one response variable
 - Example: Torque values in multiple joints of a robotic arm
 - Akin to multi-label classification

UnSupervised Learning

- Unsupervised Learning: learning without a teacher
 - Given: a set of unlabeled inputs $\{x_1, \dots, x_N\}$
 - Goal: learn some intrinsic structure in the data
 - Some Examples: Data Clustering, Dimensionality Reduction
- Data Clustering
 - Grouping a given set of inputs based on their similarities
 - Example: clustering new stories based on their topics (e.g., Google News)
 - Clustering sometimes is also referred to as (probability) density estimation
- Dimensionality Reduction
- Often, real-world data is high dimensional
 - Reducing dimensionality helps in several ways
 - Computational benefits: speeding up learning algorithms
 - Better input representations for supervised learning tasks
 - Used for data visualization by reducing data to smaller dimensions

Reinforcement Learning

- Unlike supervised/unsupervised learning, RL does not use examples
 - Rather, it learns (gathers experience) by interacting with the world
 - Defined by an agent and an environment the agent acts in
 - Agent has a set A of actions, environment has a set S of states
-
- Goal: Find a sequence of actions by the agent that maximizes its reward
 - Output: A policy which maps states to actions
 - RL problems always include time as a variable
 - Example problems: Chess, Robot control, autonomous driving
 - In RL, the key trade-off is exploration versus exploitation

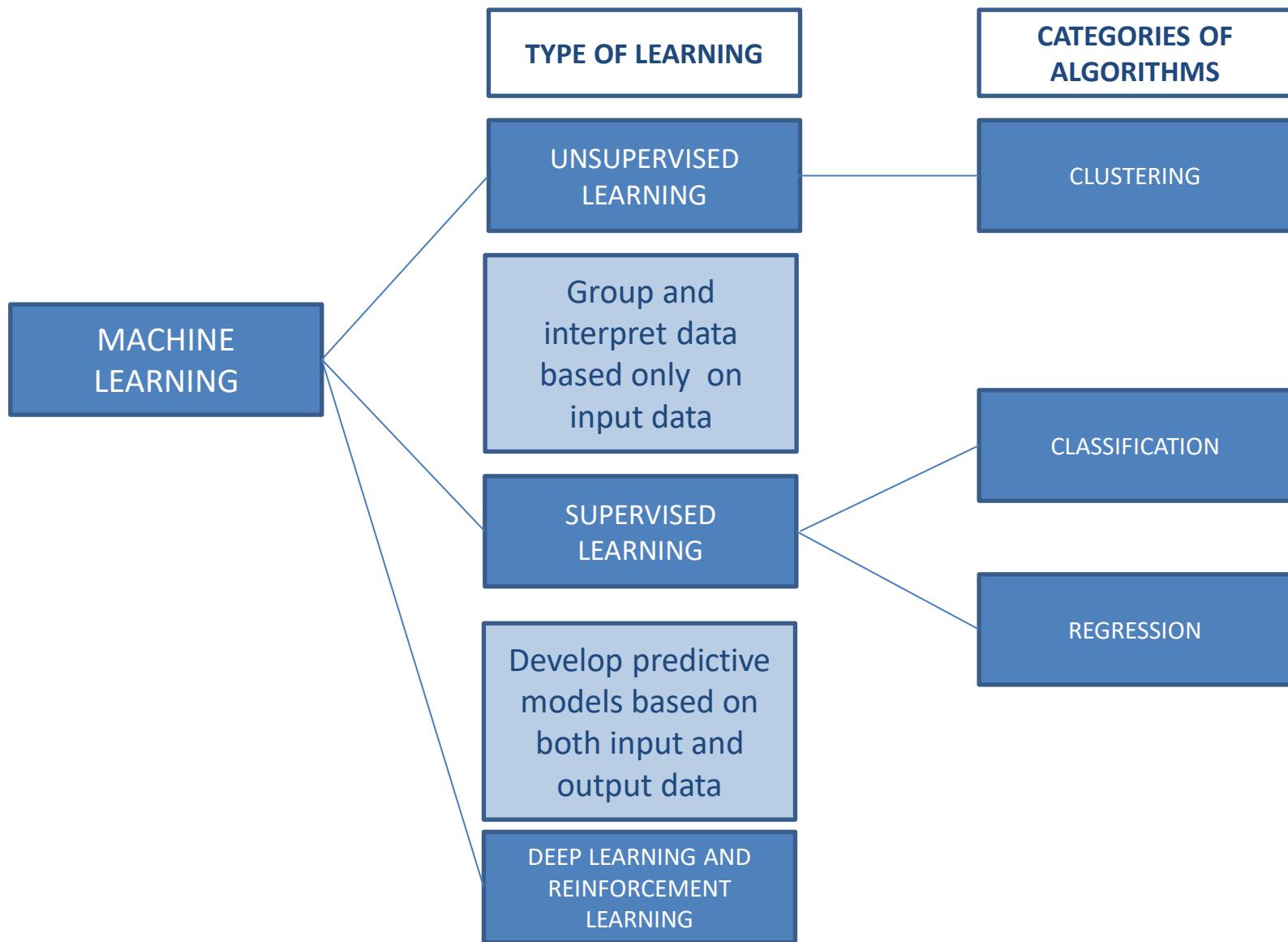
Bayesian Learning

- Not really a different learning paradigm
 - Rather, a way of doing machine learning (can be used for any learning paradigm - supervised, unsupervised, etc.)
- Most ML algorithms: Provide them data, get a model out of it
 - No way to know how confident your model parameters are
 - No way to know how confident your predictions are
- But in some problem domains, confidence estimates are important
- Bayesian Learning gives a way to quantify confidence/uncertainty
 - By maintaining a probability distribution over the parameters/predictions
 - So we also have mean and variance estimates of the parameters/predictions
- Another advantage: Incorporating prior knowledge about the problem, Bayesian methods can automatically control overfitting (and can learn well with small amounts of data)

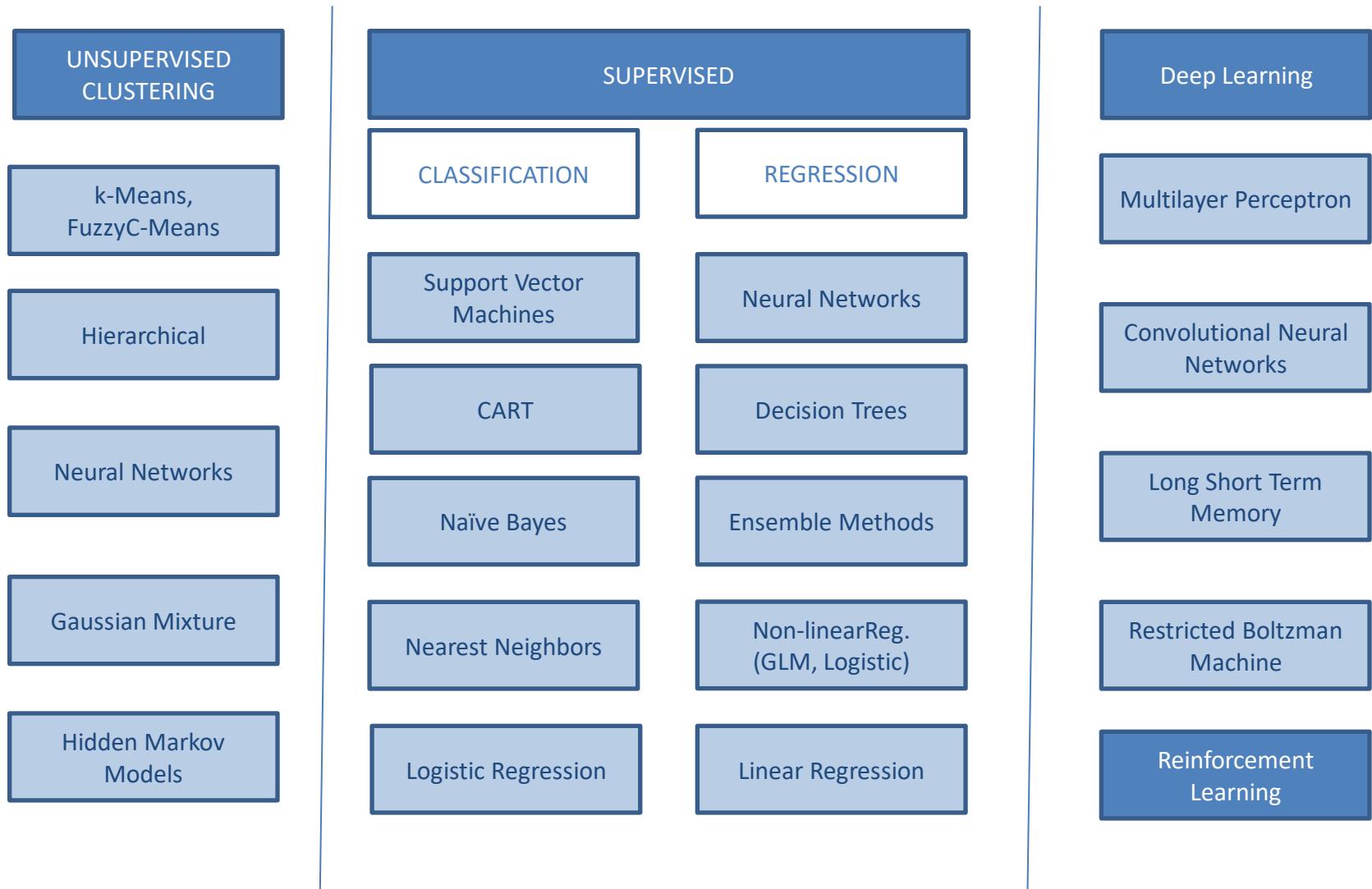
Data Representation

- Data has form: $\{(x_1 y_1), \dots, (x_N y_N)\}$ (labeled), or $\{x_1, \dots, x_N\}$ (unlabeled)
 - What the label y looks like is task-specific (as we saw)
 - What about x which denotes a real-world object (e.g., image or text document)?
 - Each example x is a set of (numeric) features/attributes/dimensions
 - Features encode properties of the object which x represents
 - x is commonly represented as a $D \times 1$ vector
 - Representing a 28×28 image: x can be a 784×1 vector of pixel values
 - Representing a text document: x can be a vector of word-counts of words appearing in that document
 - For some problems, non-vectorial representations may be more appropriate

Financial Modeling : Machine Learning



MACHINE LEARNING OVERVIEW



SUPERVISED LEARNING CLASSIFICATION

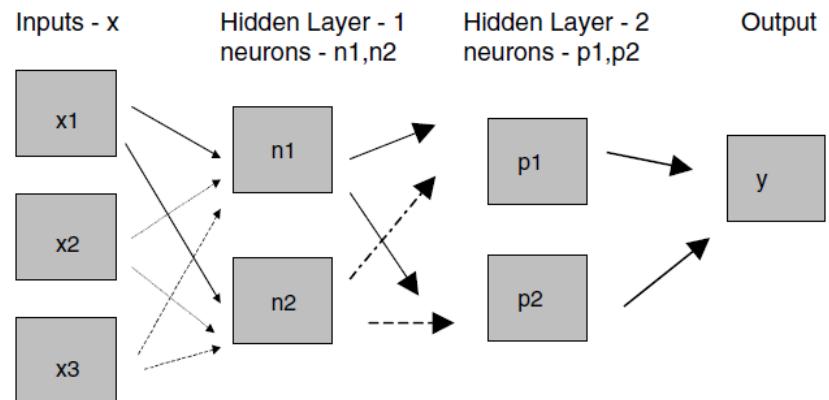
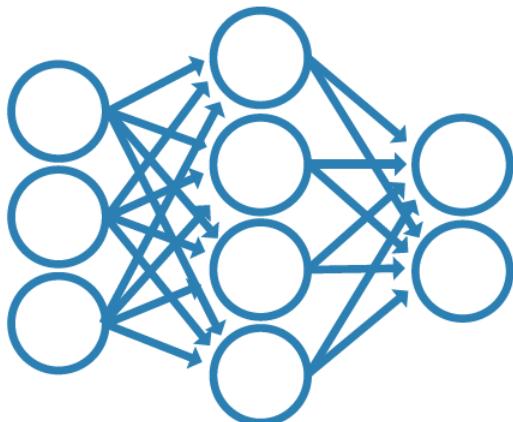
Neural Networks

How it Works

Inspired by the human brain, a neural network consists of highly connected networks of neurons that relate the inputs to the desired outputs. The network is trained by iteratively modifying the strengths of the connections so that given inputs map to the correct response.

Best Used...

- For modeling highly nonlinear systems
- When data is available incrementally and you wish to constantly update the model
- When there could be unexpected changes in your input data
- When model interpretability is not a key concern



$$n_{k,t} = \omega_{k,0} + \sum_{i=1}^{i^*} \omega_{k,i} x_{i,t}$$

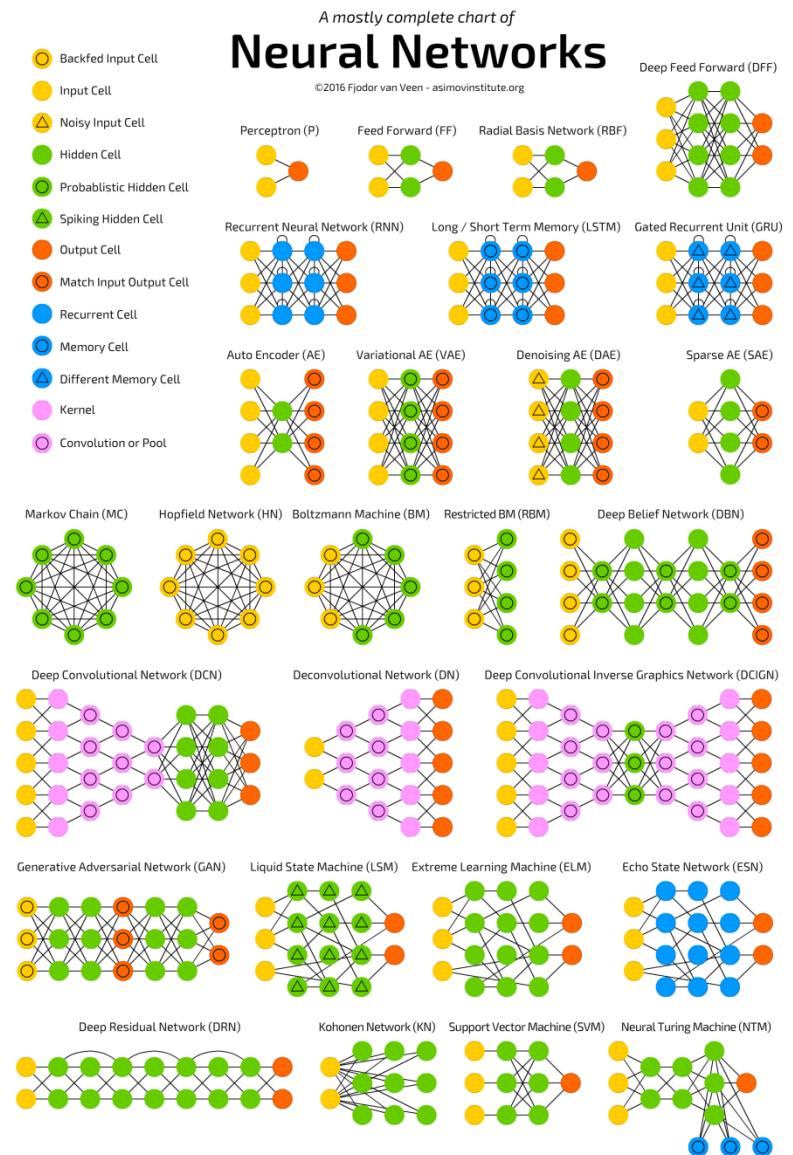
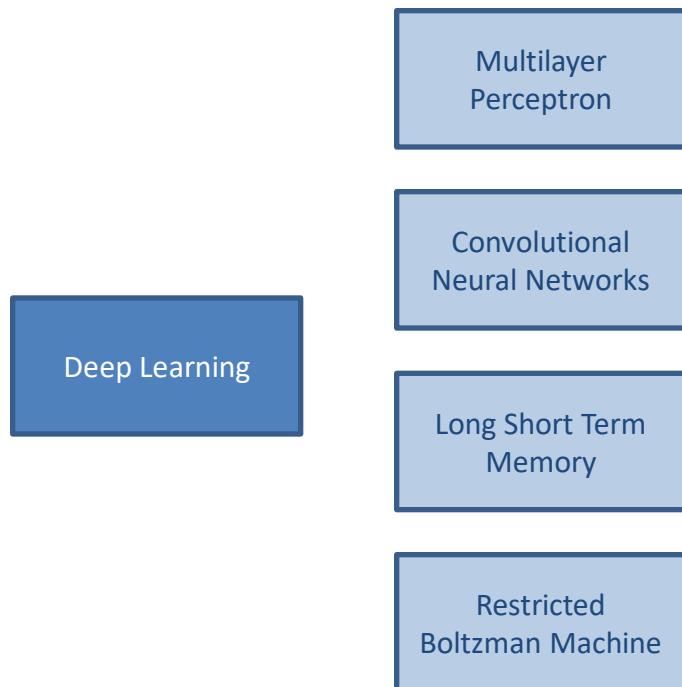
$$N_{k,t} = \frac{1}{1 + e^{-n_{k,t}}}$$

$$p_{l,t} = \rho_{l,0} + \sum_{k=1}^{k^*} \rho_{l,k} N_{k,t}$$

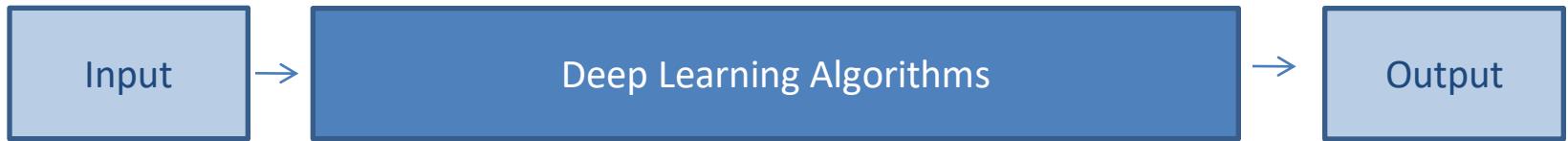
$$P_{l,t} = \frac{1}{1 + e^{-p_{l,t}}}$$

$$y_t = \gamma_0 + \sum_{l=1}^{l^*} \gamma_l P_{l,t}$$

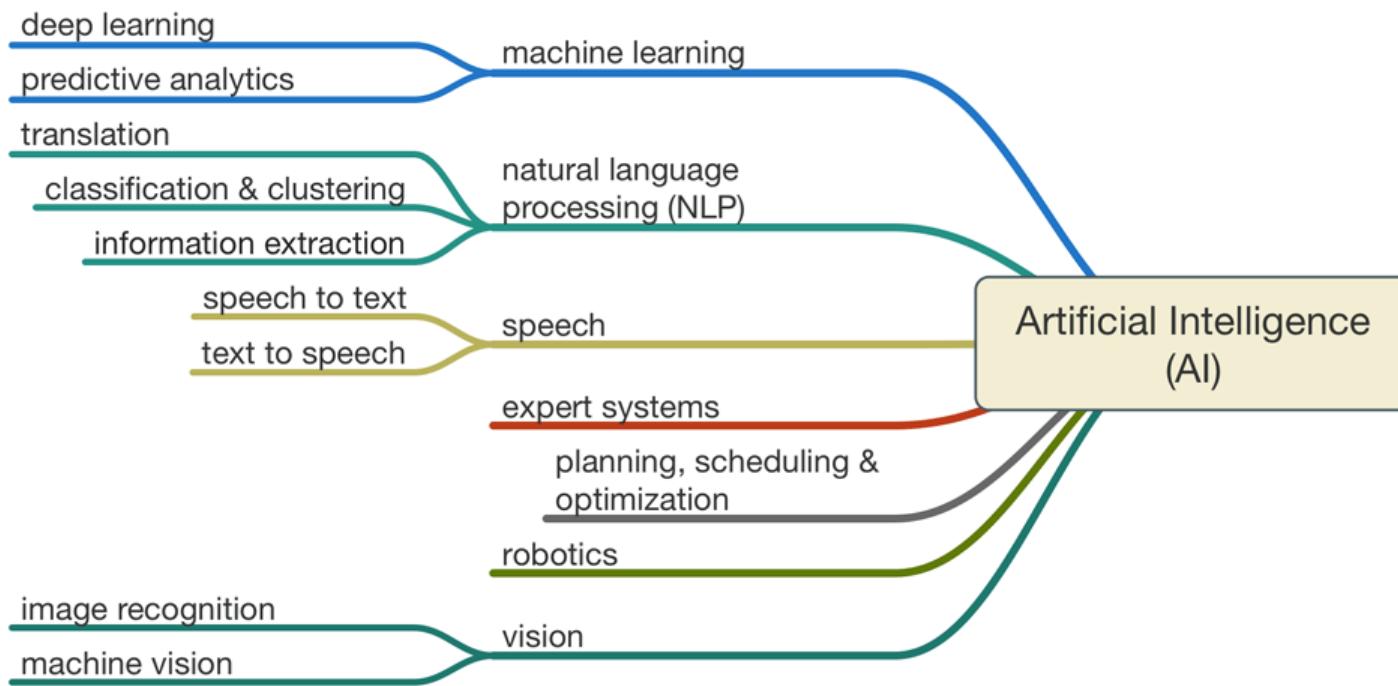
Deep Learning



Deep Learning Flow



AI vs ML



Machine Learning in Finance

3. Modeling

Machine Learning in Finance

Modeling 3.1 General Framework

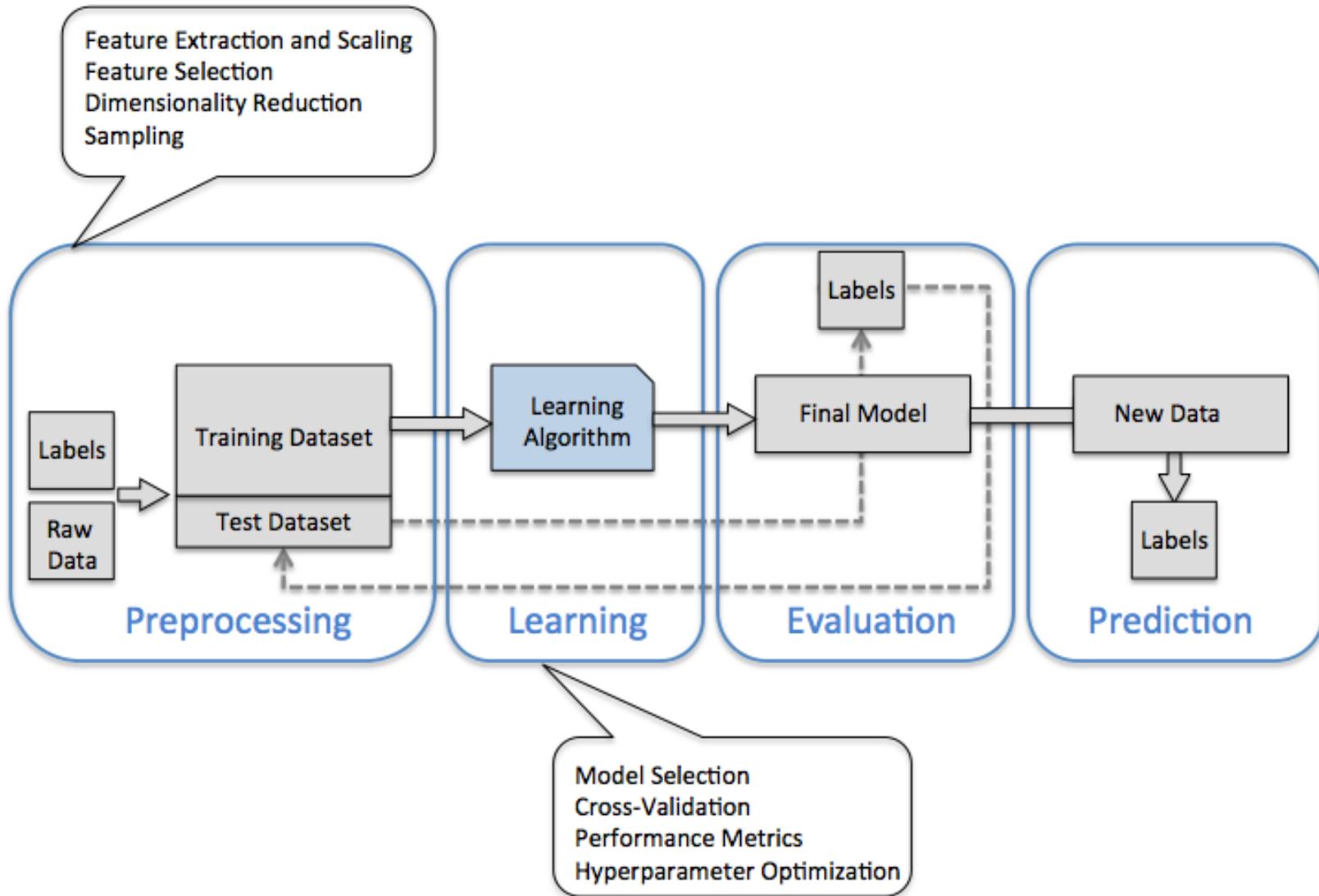
Machine Learning Definition

In general, a learning problem considers a set of n samples of data and then tries to predict properties of unknown data. If each sample is more than a single number and, for instance, a multi-dimensional entry (aka multivariate data), it is said to have several attributes or features.

Learning problems fall into a few categories:

- Supervised learning, in which the data comes with additional attributes that we want to predict. This problem can be either:
 - classification:
 - Regression: if the desired output consists of one or more continuous variables, then the task is called regression.
- Unsupervised learning, in which the training data consists of a set of input vectors x without any corresponding target values. The goal in such problems may be to discover groups of similar examples within the data, where it is called clustering, or to determine the distribution of data within the input space, known as density estimation, or to project the data from a high-dimensional space down to two or three dimensions for the purpose of visualization.

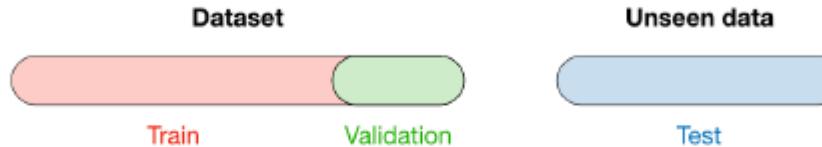
Machine Learning Modeling



Model Selection

Training set	Validation set	Testing set
- Model is trained - Usually 80% of the dataset	- Model is assessed - Usually 20% of the dataset - Also called hold-out or development set	- Model gives predictions - Unseen data

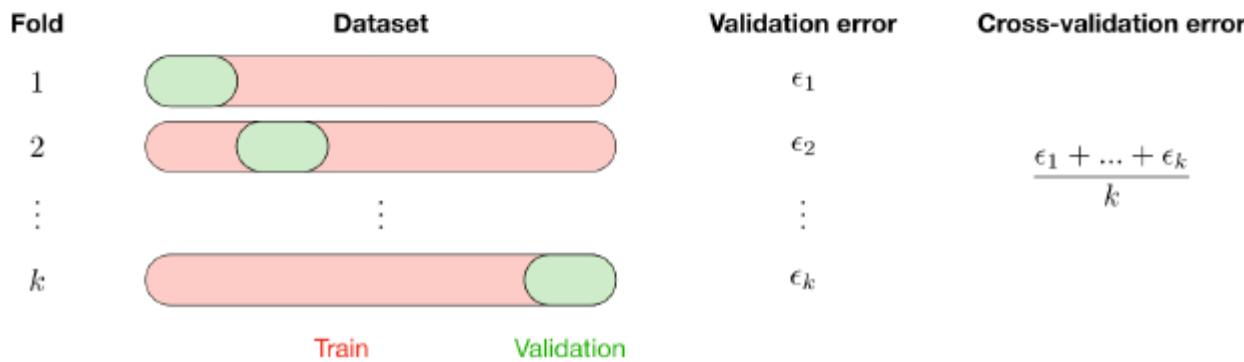
Once the model has been chosen, it is trained on the entire dataset and tested on the unseen test set. These are represented in the figure below:



□ **Cross-validation** – Cross-validation, also noted CV, is a method that is used to select a model that does not rely too much on the initial training set. The different types are summed up in the table below:

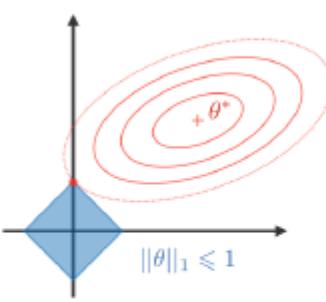
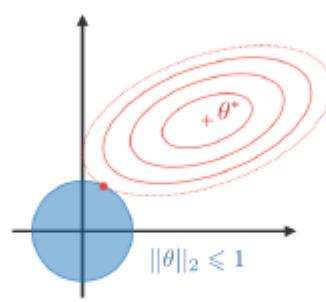
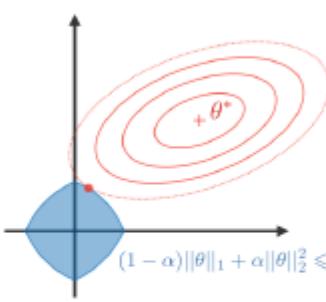
<i>k</i> -fold	Leave- <i>p</i> -out
- Training on $k - 1$ folds and assessment on the remaining one - Generally $k = 5$ or 10	- Training on $n - p$ observations and assessment on the p remaining ones - Case $p = 1$ is called leave-one-out

The most commonly used method is called *k*-fold cross-validation and splits the training data into *k* folds to validate the model on one fold while training the model on the $k - 1$ other folds, all of this *k* times. The error is then averaged over the *k* folds and is named cross-validation error.



Model Selection

□ Regularization – The regularization procedure aims at avoiding the model to overfit the data and thus deals with high variance issues. The following table sums up the different types of commonly used regularization techniques:

LASSO	Ridge	Elastic Net
<ul style="list-style-type: none"> - Shrinks coefficients to 0 - Good for variable selection 	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
		
$\dots + \lambda \ \theta\ _1$ $\lambda \in \mathbb{R}$	$\dots + \lambda \ \theta\ _2^2$ $\lambda \in \mathbb{R}$	$\dots + \lambda \left[(1 - \alpha) \ \theta\ _1 + \alpha \ \theta\ _2^2 \right]$ $\lambda \in \mathbb{R}, \quad \alpha \in [0,1]$

□ Model selection – Train model on training set, then evaluate on the development set, then pick best performance model on the development set, and retrain all of that model on the whole training set.

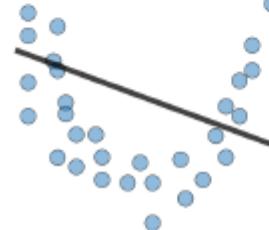
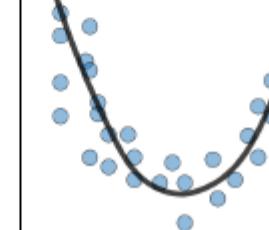
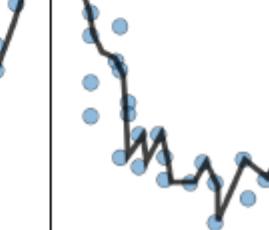
Model Selection

Diagnostics

□ **Bias** – The bias of a model is the difference between the expected prediction and the correct model that we try to predict for given data points.

□ **Variance** – The variance of a model is the variability of the model prediction for given data points.

□ **Bias/variance tradeoff** – The simpler the model, the higher the bias, and the more complex the model, the higher the variance.

	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none">- High training error- Training error close to test error- High bias	<ul style="list-style-type: none">- Training error slightly lower than test error	<ul style="list-style-type: none">- Low training error- Training error much lower than test error- High variance
Regression			

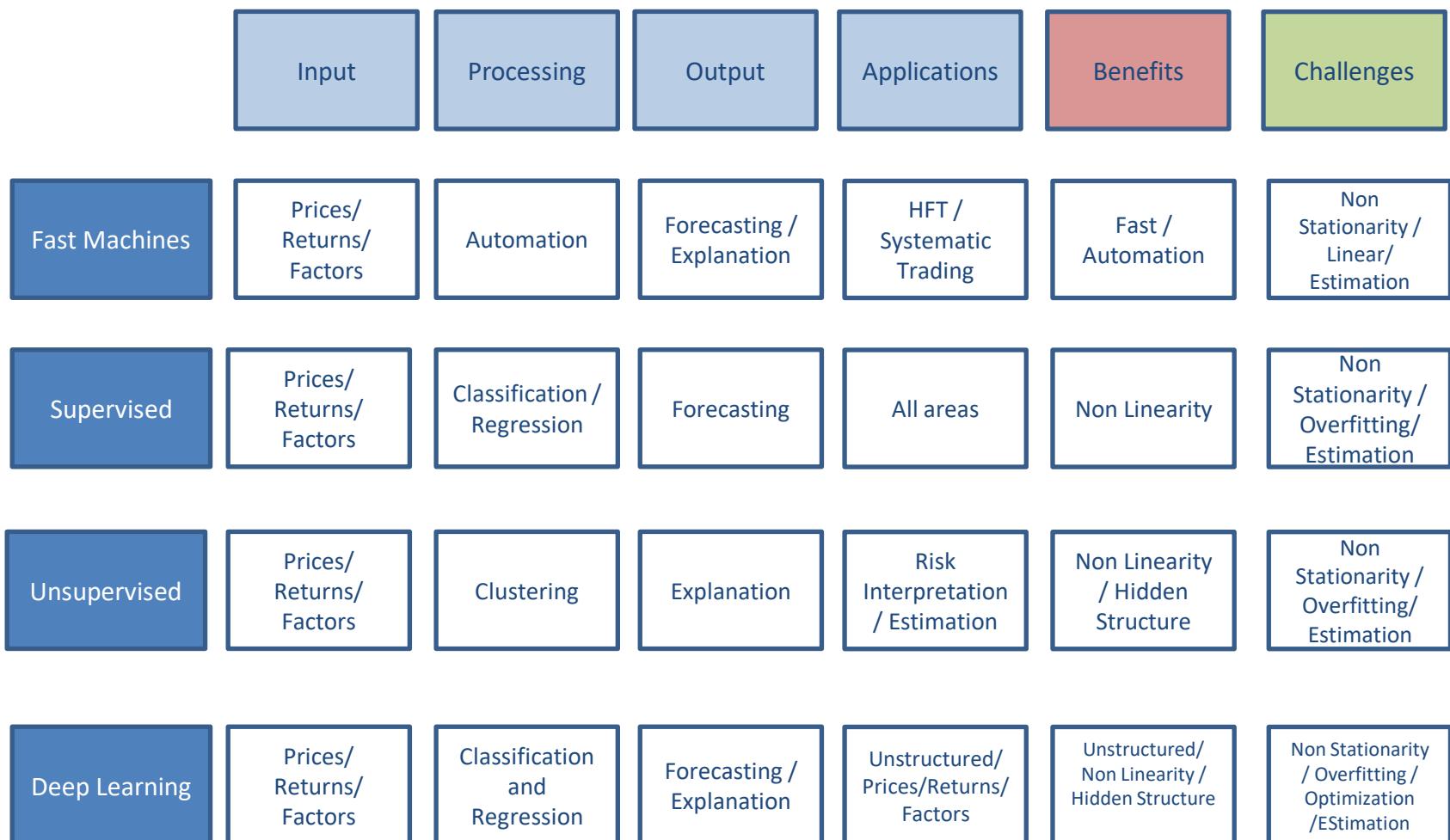
Model Selection

Classification			
Deep learning			
Remedies	<ul style="list-style-type: none">- Complexify model- Add more features- Train longer		

❑ **Error analysis** – Error analysis is analyzing the root cause of the difference in performance between the current and the perfect models.

❑ **Ablative analysis** – Ablative analysis is analyzing the root cause of the difference in performance between the current and the baseline models.

Machine Learning in Finance - Instructions of Use



Machine Learning in Finance

Modeling

3.2 Pre-processing and Feature
Selection

Pre-Processing Data

In general, learning algorithms benefit from standardization of the data set.

If some outliers are present in the set, robust scalers or transformers are more appropriate. The behaviors of the different scalers, transformers, and normalizers on a dataset containing marginal outliers is highlighted in [Compare the effect of different scalers on data with outliers](#).

The `sklearn.preprocessing` package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.

Pre-Processing Data

1. **Standardization, or mean removal and variance scaling**
 1. **Standardization of datasets** is a common requirement for many machine learning estimators implemented in scikit-learn; they might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with zero mean and unit variance.
In practice we often ignore the shape of the distribution and just transform the data to center it by removing the mean value of each feature, then scale it by dividing non-constant features by their standard deviation.
2. **Scaling Features to a range**

An alternative standardization is scaling features to lie between a given minimum and maximum value, often between zero and one, or so that the maximum absolute value of each feature is scaled to unit size. This can be achieved using MinMaxScaler or MaxAbsScaler, respectively.

The motivation to use this scaling include robustness to very small standard deviations of features and preserving zero entries in sparse data.

Pre-Processing Data

3. Standardization - Scaling sparse data¶

Centering sparse data would destroy the sparseness structure in the data, and thus rarely is a sensible thing to do. However, it can make sense to scale sparse inputs, especially if features are on different scales.

4. Scaling data with outliers¶

If your data contains many outliers, scaling using the mean and variance of the data is likely to not work very well. In these cases, you can use `robust_scale` and `RobustScaler` as drop-in replacements instead. They use more robust estimates for the center and range of your data.

2. Non-Linear Transformations

1. Mapping to a Uniform Distribution – Grading - Like scalers, `QuantileTransformer` puts all features into the same, known range or distribution. However, by performing a rank transformation, it smooths out unusual distributions and is less influenced by outliers than scaling methods. It does, however, distort correlations and distances within and across features. Providing a non-parametric transformation based on the quantile function to map the data to a uniform distribution with values between 0 and 1.

Pre-Processing Data

2. Non-Linear Transformations

1. Mapping to a Uniform Distribution.
 2. Mapping to a Gaussian distribution¶. In many modeling scenarios, normality of the features in a dataset is desirable. Power transforms are a family of parametric, monotonic transformations that aim to map data from any distribution to as close to a Gaussian distribution as possible in order to stabilize variance and minimize skewness.
- 3. Normalization** - Normalization is the process of scaling individual samples to have unit norm. This process can be useful if you plan to use a quadratic form such as the dot-product or any other kernel to quantify the similarity of any pair of samples. This assumption is the base of the Vector Space Model often used in text classification and clustering contexts. The function `normalize` provides a quick and easy way to perform this operation on a single array-like dataset, either using the L1 or L2 norms.

Pre-Processing Data

4. **Encoding Categorical Features** - Often features are not given as continuous values but categorical. For example a person could have features ["male", "female"], ["from Europe", "from US", "from Asia"], ["uses Firefox", "uses Chrome", "uses Safari", "uses Internet Explorer"]. Such features can be efficiently coded as integers, for instance ["male", "from US", "uses Internet Explorer"] could be expressed as [0, 1, 3] while ["female", "from Asia", "uses Chrome"] would be [1, 2, 1]. To convert categorical features to such integer codes, we can use the OrdinalEncoder. This estimator transforms each categorical feature to one new feature of integers (0 to n_categories - 1).
5. **Discretization** (otherwise known as quantization or binning) provides a way to partition continuous features into discrete values. Certain datasets with continuous features may benefit from discretization, because discretization can transform the dataset of continuous attributes to one with only nominal attributes.
6. **Missing Values**

Machine Learning in Finance

Modeling

3.3 Performance Metrics

Classification Metrics

In a context of a binary classification, here are the main metrics that are important to track to assess the performance of the model.

- **Confusion matrix** – The confusion matrix is used to have a more complete picture when assessing the performance of a model. It is defined as follows:

		Predicted class	
		+	-
Actual class	+	TP True Positives	FN False Negatives Type II error
	-	FP False Positives Type I error	TN True Negatives

- **Main metrics** – The following metrics are commonly used to assess the performance of classification models:

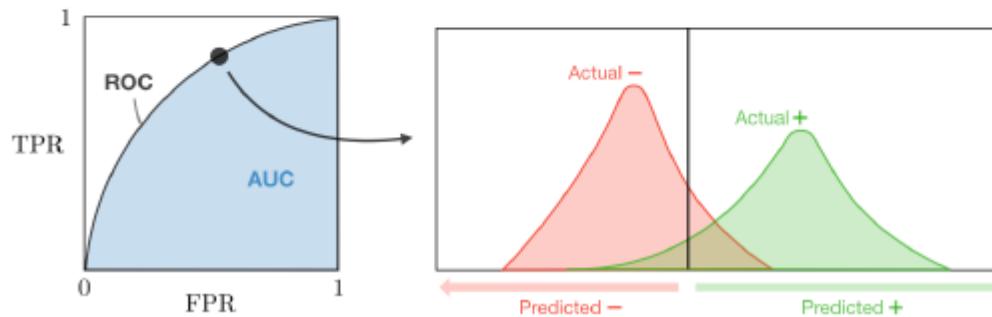
Metric	Formula	Interpretation
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	Overall performance of model
Precision	$\frac{TP}{TP + FP}$	How accurate the positive predictions are
Recall Sensitivity	$\frac{TP}{TP + FN}$	Coverage of actual positive sample
Specificity	$\frac{TN}{TN + FP}$	Coverage of actual negative sample
F1 score	$\frac{2TP}{2TP + FP + FN}$	Hybrid metric useful for unbalanced classes

Classification Metrics

□ **ROC** – The receiver operating curve, also noted ROC, is the plot of TPR versus FPR by varying the threshold. These metrics are summed up in the table below:

Metric	Formula	Equivalent
True Positive Rate TPR	$\frac{TP}{TP + FN}$	Recall, sensitivity
False Positive Rate FPR	$\frac{FP}{TN + FP}$	1-specificity

□ **AUC** – The area under the receiving operating curve, also noted AUC or AUROC, is the area below the ROC as shown in the following figure:



Regression Metrics

Regression

□ **Basic metrics** – Given a regression model f , the following metrics are commonly used to assess the performance of the model:

Total sum of squares	Explained sum of squares	Residual sum of squares
$SS_{\text{tot}} = \sum_{i=1}^m (y_i - \bar{y})^2$	$SS_{\text{reg}} = \sum_{i=1}^m (f(x_i) - \bar{y})^2$	$SS_{\text{res}} = \sum_{i=1}^m (y_i - f(x_i))^2$

□ **Coefficient of determination** – The coefficient of determination, often noted R^2 or r^2 , provides a measure of how well the observed outcomes are replicated by the model and is defined as follows:

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

□ **Main metrics** – The following metrics are commonly used to assess the performance of regression models, by taking into account the number of variables n that they take into consideration:

Mallow's Cp	AIC	BIC	Adjusted R^2
$\frac{SS_{\text{res}} + 2(n+1)\hat{\sigma}^2}{m}$	$2[(n+2) - \log(L)]$	$\log(m)(n+2) - 2\log(L)$	$1 - \frac{(1-R^2)(m-1)}{m-n-1}$

Classification Metrics

1. Classification Accuracy.
2. Logarithmic
3. Area Under ROC Curve.
4. Confusion Matrix.
5. Classification Report.

1. Classification Accuracy

Classification accuracy is the number of correct predictions made as a ratio of all predictions made. This is the most common evaluation metric for classification problems, it is also the most misused. It is really only suitable when there are an equal number of observations in each class (which is rarely the case) and that all predictions and prediction errors are equally important, which is often not the case.

```
# Cross Validation Classification Accuracy
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indian
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LogisticRegression()
scoring = 'accuracy'
results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))
```

1 Accuracy: 0.770 (0.048)

2. Logarithmic Loss

Logarithmic loss (or logloss) is a performance metric for evaluating the predictions of probabilities of membership to a given class.

The scalar probability between 0 and 1 can be seen as a measure of confidence for a prediction by an algorithm. Predictions that are correct or incorrect are rewarded or punished proportionally to the confidence of the prediction. Smaller logloss is better with 0 representing a perfect logloss. function.

```
1 # Cross Validation Classification LogLoss
2 import pandas
3 from sklearn import model_selection
4 from sklearn.linear_model import LogisticRegression
5 url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"
6 names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
7 dataframe = pandas.read_csv(url, names=names)
8 array = dataframe.values
9 X = array[:,0:8]
10 Y = array[:,8]
11 seed = 7
12 kfold = model_selection.KFold(n_splits=10, random_state=seed)
13 model = LogisticRegression()
14 scoring = 'neg_log_loss'
15 results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
16 print("Logloss: %.3f (%.3f)" % (results.mean(), results.std()))
```

3. Area under Roc Curve

Area under ROC Curve (or AUC for short) is a performance metric for binary classification problems.

The AUC represents a model's ability to discriminate between positive and negative classes. An area of 1.0 represents a model that made all predictions perfectly. An area of 0.5 represents a model as good as random. Learn more about ROC [here](#).

ROC can be broken down into sensitivity and specificity. A binary classification problem is really a trade-off between sensitivity and specificity.

Sensitivity is the true positive rate also called the recall. It is the number instances from the positive (first) class that actually predicted correctly.

Specificity is also called the true negative rate. Is the number of instances from the negative class (second) class that were actually predicted correctly.

3. Area under Roc Curve

```
# Cross Validation Classification ROC AUC
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LogisticRegression()
scoring = 'roc_auc'
results = model_selection.cross_val_score(model, X, Y, cv=kfold,
scoring=scoring)
print("AUC: %.3f (%.3f)" % (results.mean(), results.std()))
```

4. Confusion Matrix

The confusion matrix is a handy presentation of the accuracy of a model with two or more classes.

The table presents predictions on the x-axis and accuracy outcomes on the y-axis. The cells of the table are the number of predictions made by a machine learning algorithm.

For example, a machine learning algorithm can predict 0 or 1 and each prediction may actually have been a 0 or 1. Predictions for 0 that were actually 0 appear in the cell for prediction=0 and actual=0, whereas predictions for 0 that were actually 1 appear in the cell for prediction = 0 and actual=1. And so on.

4. Confusion Matrix

```
# Cross Validation Classification Confusion Matrix
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/pima-indians-diabetes/pima-indians-
diabetes.data"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test =
model_selection.train_test_split(X, Y, test_size=test_size,
random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
matrix = confusion_matrix(Y_test, predicted)
print(matrix)
```

5. Classification Report

Scikit-learn does provide a convenience report when working on classification problems to give you a quick idea of the accuracy of a model using a number of measures.

The `classification_report()` function displays the precision, recall, f1-score and support for each class.

The example demonstrates the report on the binary classification problem

5. Classification Report

Scik# Cross Validation Classification Report

```
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_size=test_size, random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
report = classification_report(Y_test, predicted)
print(report)
```

	precision	recall	f1-score	support	
1					
2					
3	0.0	0.77	0.87	0.82	162
4	1.0	0.71	0.55	0.62	92
5					
6 avg / total	0.75	0.76	0.75	254	

Regression Metrics

1. Mean Absolute Error.
2. Mean Squared Error.
3. R².

1. Mean Absolute Error

The Mean Absolute Error (or MAE) is the sum of the absolute differences between predictions and actual values. It gives an idea of how wrong the predictions were.

The measure gives an idea of the magnitude of the error, but no idea of the direction (e.g. over or under predicting).

A value of 0 indicates no error or perfect predictions. Like logloss, this metric is inverted by the `cross_val_score()` function.

```
# Cross Validation Regression MAE
import pandas
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/housing/housing.data"
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = pandas.read_csv(url, delim_whitespace=True,
names=names)
array = dataframe.values
X = array[:,0:13]
Y = array[:,13]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LinearRegression()
scoring = 'neg_mean_absolute_error'
results = model_selection.cross_val_score(model, X, Y, cv=kfold,
scoring=scoring)
print("MAE: %.3f (%.3f)" % (results.mean(), results.std()))
```

```
1 MAE: -4.005 (2.084)
```

2. Mean Squared Error

The Mean Squared Error (or MSE) is much like the mean absolute error in that it provides a gross idea of the magnitude of error.

Taking the square root of the mean squared error converts the units back to the original units of the output variable and can be meaningful for description and presentation. This is called the Root Mean Squared Error (or RMSE).

```
# Cross Validation Regression MSE
import pandas
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data"
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = pandas.read_csv(url, delim_whitespace=True, names=names)
array = dataframe.values
X = array[:,0:13]
Y = array[:,13]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LinearRegression()
scoring = 'neg_mean_squared_error'
results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("MSE: %.3f (%.3f)" % (results.mean(), results.std()))
```

1 MSE: -34.705 (45.574)

3. R Square

The R^2 (or R Squared) metric provides an indication of the goodness of fit of a set of predictions to the actual values. In statistical literature, this measure is called the coefficient of determination.

This is a value between 0 and 1 for no-fit and perfect fit respectively.

You can see that the predictions have a poor fit to the actual values with a value close to zero and less than 0.5.

```
# Cross Validation Regression R^2
import pandas
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/housing/housing.data"
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = pandas.read_csv(url, delim_whitespace=True,
names=names)
array = dataframe.values
X = array[:,0:13]
Y = array[:,13]
seed = 7
kfold = model_selection.KFold(n_splits=10,
random_state=seed)
model = LinearRegression()
scoring = 'r2'
results = model_selection.cross_val_score(model, X, Y,
cv=kfold, scoring=scoring)
print("R^2: %.3f (%.3f)" % (results.mean(), results.std()))
```

1 R^2: 0.203 (0.595)

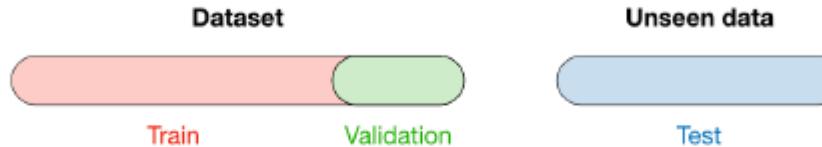
Machine Learning in Finance

Modeling
3.4 Cross-Validation

Model Selection

Training set	Validation set	Testing set
- Model is trained - Usually 80% of the dataset	- Model is assessed - Usually 20% of the dataset - Also called hold-out or development set	- Model gives predictions - Unseen data

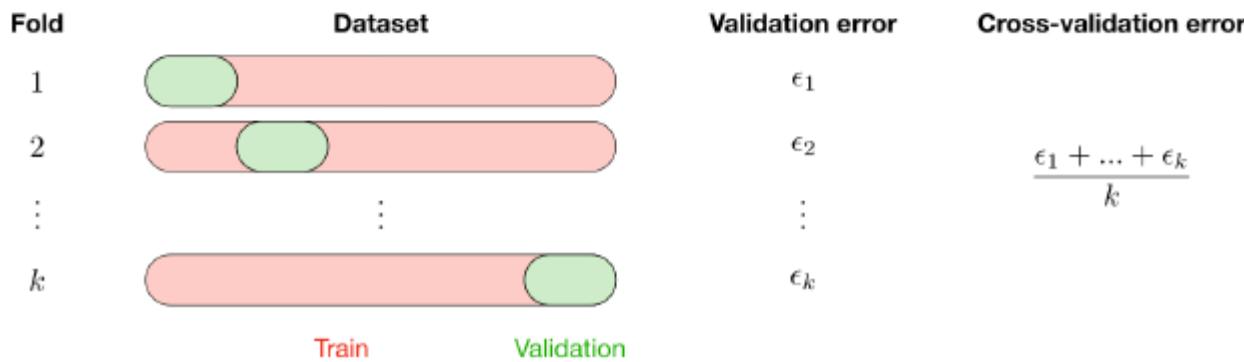
Once the model has been chosen, it is trained on the entire dataset and tested on the unseen test set. These are represented in the figure below:



□ **Cross-validation** – Cross-validation, also noted CV, is a method that is used to select a model that does not rely too much on the initial training set. The different types are summed up in the table below:

<i>k</i> -fold	Leave- <i>p</i> -out
- Training on $k - 1$ folds and assessment on the remaining one - Generally $k = 5$ or 10	- Training on $n - p$ observations and assessment on the p remaining ones - Case $p = 1$ is called leave-one-out

The most commonly used method is called *k*-fold cross-validation and splits the training data into *k* folds to validate the model on one fold while training the model on the $k - 1$ other folds, all of this *k* times. The error is then averaged over the *k* folds and is named cross-validation error.



Cross-Validation

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called overfitting.

To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a test set X_{test} , y_{test} . Note that the word “experiment” is not intended to denote academic use only, because even in commercial settings machine learning usually starts out experimentally.

In scikit-learn a random split into training and test sets can be quickly computed with the `train_test_split` helper function. Let’s load the iris data set to fit a linear support vector machine on it:

Cross-Validation

A solution to this problem is a procedure called cross-validation (CV for short). A test set should still be held out for final evaluation, but the validation set is no longer needed when doing CV. In the basic approach, called k-fold CV, the training set is split into k smaller sets (other approaches are described below, but generally follow the same principles). The following procedure is followed for each of the k “folds”:

A model is trained using $k-1$ of the folds as training data, the resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).

The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop. This approach can be computationally expensive, but does not waste too much data (as is the case when fixing an arbitrary validation set), which is a major advantage in problems such as inverse inference where the number of samples is very small.

Cross-Validation

When evaluating different settings (“hyperparameters”) for estimators, such as the C setting that must be manually set for an SVM, there is still a risk of overfitting on the test set because the parameters can be tweaked until the estimator performs optimally.

This way, knowledge about the test set can “leak” into the model and evaluation metrics no longer report on generalization performance. To solve this problem, yet another part of the dataset can be held out as a so-called “validation set”: training proceeds on the training set, after which evaluation is done on the validation set, and when the experiment seems to be successful, final evaluation can be done on the test set.

However, by partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets.

Cross-Validation – IID's

Assuming that some data is Independent and Identically Distributed (i.i.d.) is making the assumption that all samples stem from the same generative process and that the generative process is assumed to have no memory of past generated samples. The following cross-validators can be used in such cases.

NOTE

While i.i.d. data is a common assumption in machine learning theory, it rarely holds in practice. If one knows that

the samples have been generated using a time-dependent process, it's safer to use a time-series aware cross-validation scheme. Similarly if we know that the generative process has a group structure (samples from different subjects, experiments, measurement devices) it's safer to use group-wise cross-validation.

Cross-Validation – IID's

KFold divides all the samples in k groups of samples, called folds (if $k = n$, this is equivalent to the Leave One Out strategy), of equal sizes (if possible). The prediction function is learned using $k - 1$ folds, and the fold left out is used for test.

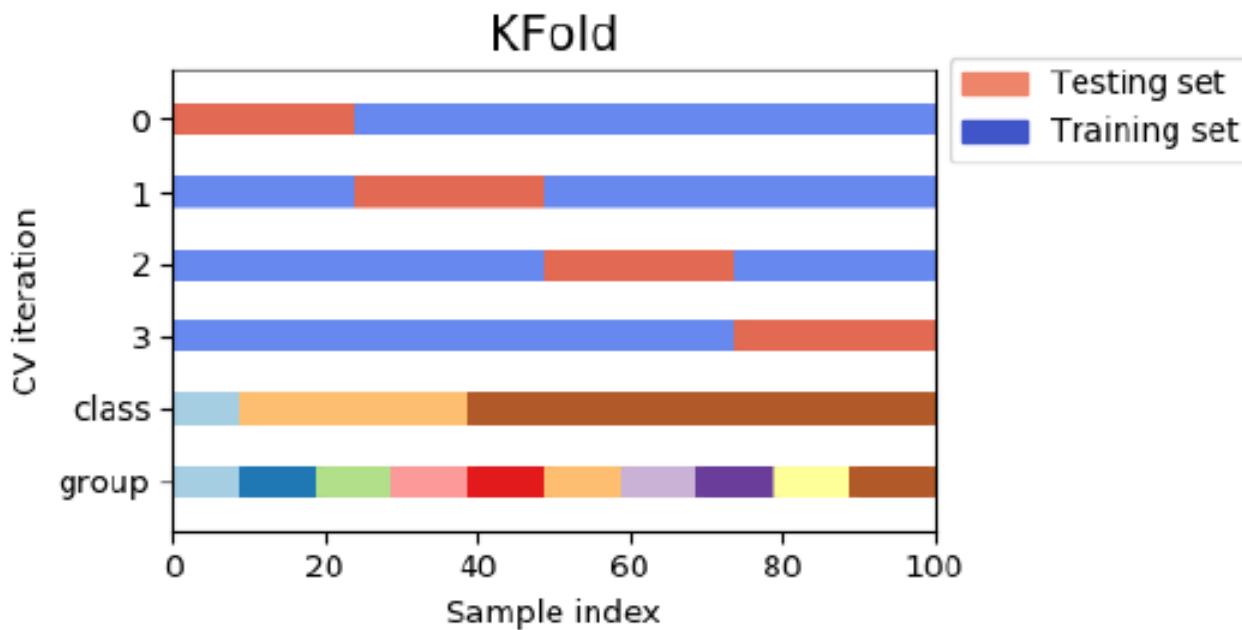
Example of 2-fold cross-validation on a dataset with 4 samples:

```
>>> import numpy as np
>>> from sklearn.model_selection import KFold

>>> X = ["a", "b", "c", "d"]
>>> kf = KFold(n_splits=2)
>>> for train, test in kf.split(X):
...     print("%s %s" % (train, test))
[2 3] [0 1]
[0 1] [2 3]
```

Cross-Validation – IID's

Behavior K-Fold



Cross-Validation – Time Series Case

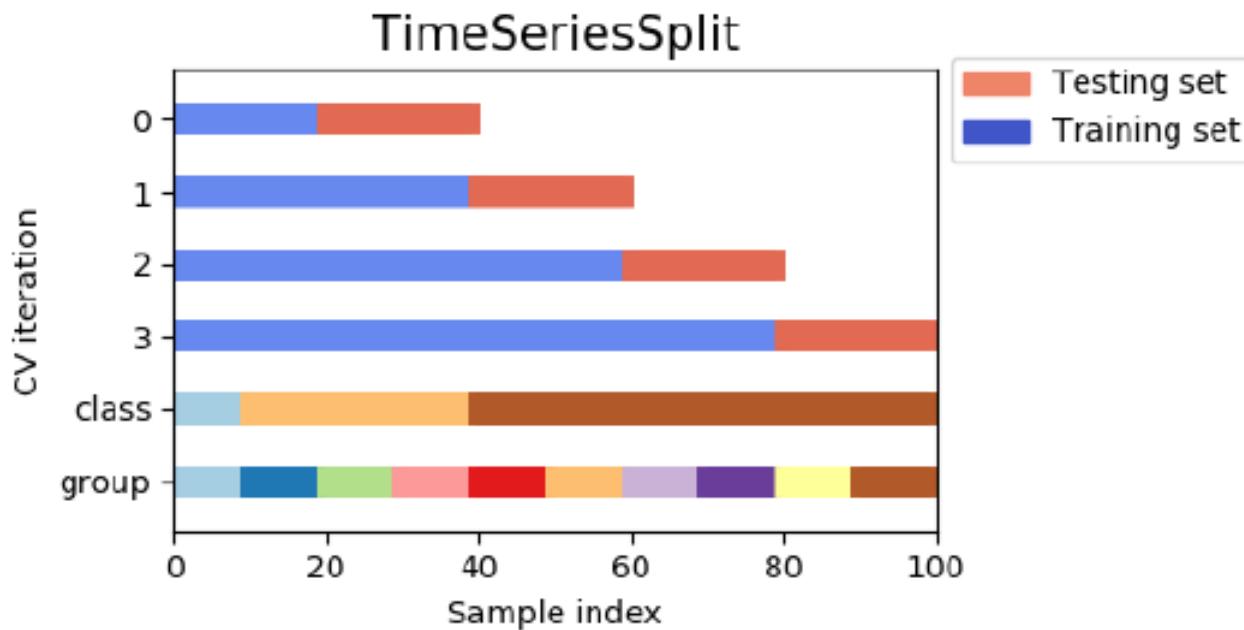
Time series data is characterised by the correlation between observations that are near in time (autocorrelation). However, classical cross-validation techniques such as KFold and ShuffleSplit assume the samples are independent and identically distributed, and would result in unreasonable correlation between training and testing instances (yielding poor estimates of generalisation error) on time series data. Therefore, it is very important to evaluate our model for time series data on the “future” observations least like those that are used to train the model. To achieve this, one solution is provided by TimeSeriesSplit.

```
>>> from sklearn.model_selection import TimeSeriesSplit

>>> X = np.array([[1, 2], [3, 4], [1, 2], [3, 4], [1, 2], [3, 4]])
>>> y = np.array([1, 2, 3, 4, 5, 6])
>>> tscv = TimeSeriesSplit(n_splits=3)
>>> print(tscv)
TimeSeriesSplit(max_train_size=None, n_splits=3)
>>> for train, test in tscv.split(X):
...     print("%s %s" % (train, test))
[0 1 2] [3]
[0 1 2 3] [4]
[0 1 2 3 4] [5]
```

Cross-Validation – Time Series Case

Behavior Time series Case



Machine Learning in Finance

Modeling 3.5 Model Selection

Machine Learning Trade-Offs

1. Prediction accuracy versus interpretability. Linear models are easy to interpret; thin-plate splines are not.
2. Good fit versus over-fit or under-fit. How do we know when the fit is just right?
3. Parsimony versus black-box. We often prefer a simpler model involving fewer variables over a black-box predictor involving them all.

Model Performance

Suppose we fit a model $\hat{f}(x)$ to some training data $Tr = \{x_i, y_i\}_1^N$, and we wish to see how well it performs.

We could compute the average squared prediction error over Tr :

$$MSE_{Tr} = \text{Ave}_{i \in Tr} [y_i - \hat{f}(x_i)]^2$$

This may be biased toward more overfit models.

Instead we should, if possible, compute it using fresh test data $T_e = \{x_i, y_i\}_1^M$:

$$MSE_{Te} = \text{Ave}_{i \in Te} [y_i - \hat{f}(x_i)]^2$$

Bias – Variance Trade - OFF

Suppose we have fitted a model $\hat{f}(x)$ to some training data Tr , and let $(x_0; y_0)$ be a test observation drawn from the population. If the true model is $Y = f(X) + \epsilon$ (with $f(x) = E(Y|X = x)$), then

$$E(y_0 - \hat{f}(x_0))^2 = Var(\hat{f}(x_0)) + [Bias(\hat{f}(x_0))]^2 + Var(\epsilon)$$

The expectation averages over the variability of y_0 as well as the variability in Tr .

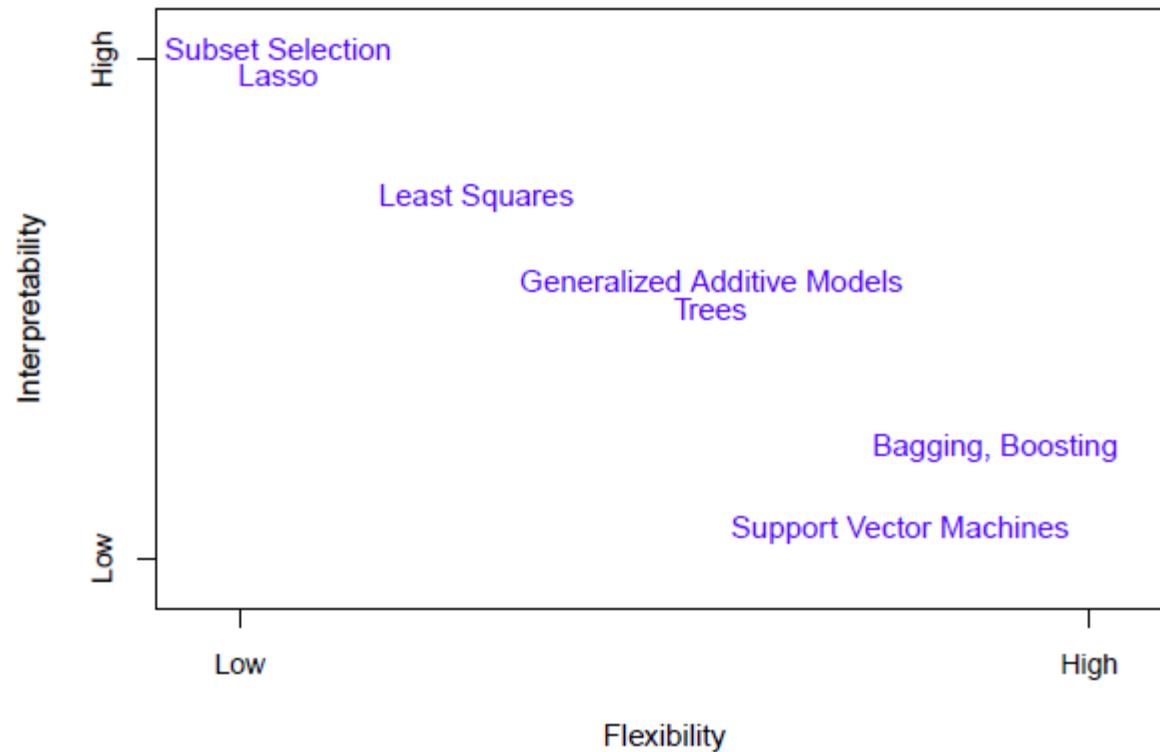
Note that $Bias(\hat{f}(x_0)) = E[\hat{f}(x_0)] - f(x_0)$.

Typically as the flexibility of \hat{f} increases, its variance increases, and its bias decreases. So choosing the flexibility based on average test error amounts to a bias-variance trade-off.

Training Error versus Test error

- Recall the distinction between the test error and the training error:
- The test error is the average error that results from using a statistical learning method to predict the response on a new observation, one that was not used in training the method.
- In contrast, the training error can be easily calculated by applying the statistical learning method to the observations used in its training.
- But the training error rate often is quite different from the test error rate, and in particular the former can dramatically underestimate the latter.

Machine Learning Modeling – Model Features



Machine Learning in Finance

Non Linearity Tests

Lee-White-Granger Test

Lee, White, and Granger (1992) proposed the use of artificially generated neural networks for testing for the presence of neglected nonlinearity in the regression residuals of any estimated model.

The test works with the regressions residuals and the inputs of the model, and seeks to find out if any of the residuals can be explained by nonlinear transformations of the input variables. If they can be explained, there is **neglected nonlinearity**.

Since the precise form of the nonlinearity is unspecified, Lee, White, and Granger propose a neural network approach, but they leave aside the time-consuming estimation process for the neural network. Instead, the coefficients or weights linking the inputs to the neurons are generated randomly.

Neural Network Test for Neglected Nonlinearity: Lee-White-Granger Test

The **Lee, White, Granger (L-W-G)** test is rather straightforward, and proceeds in six steps:

1. From the initial model, obtain the residuals and the input variables.
2. Generate a set of neuron regressors from the inputs, with randomly generated weights for the input variables.
3. Regress the residuals on the neurons, and obtain the multiple correlation coefficients.
4. Repeat this process 1000 times.
5. Assess the significance of the multiple correlation coefficients by F statistics.
6. If these coefficients are significant more than 5% of the time, there is a case for neglected nonlinearity.

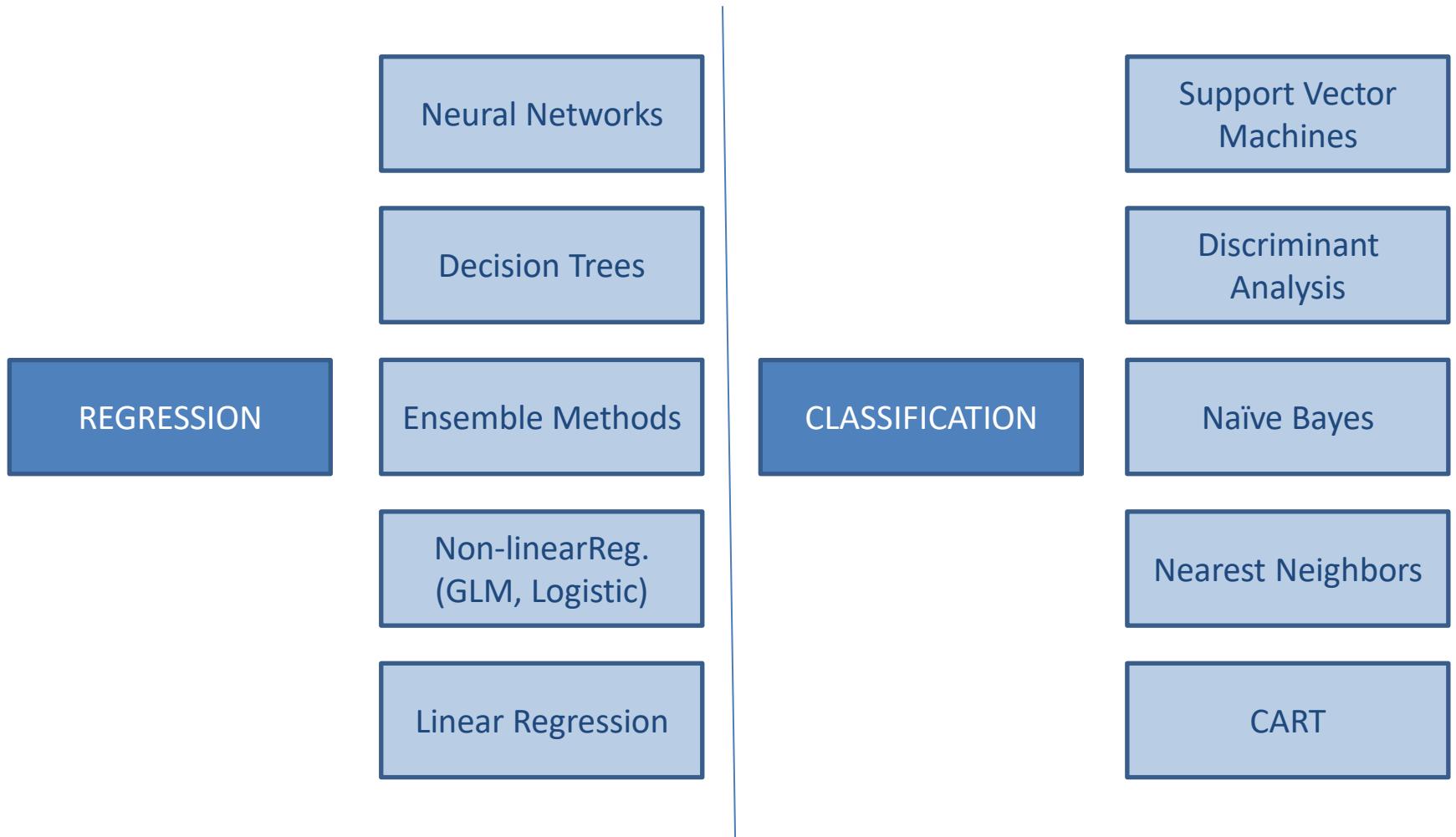
Brock-Deckert-Scheinkman Test for Nonlinear Patterns

Definition	Operation
Form m -dimensional vector, x_t^m	$x_t^m = x_t, \dots, x_{t+m}, t = 1, \dots, T_{m-1}, T_{m-1} = T - m$
Form m -dimensional vector, x_s^m	$x_s^m = x_s, \dots, x_{s+m}, s = t + 1, \dots, T_m, T_m = T - m + 1$
Form indicator function	$I_\varepsilon(x_t^m, x_s^m) = \max_{i=0,1,\dots,m-1} x_{t+1} - x_{s+i} < \varepsilon$
Calculate correlation integral	$C_{m,T}(\varepsilon) = 2 \sum_{t=1}^{T_{m-1}} \sum_{s=t+1}^{T_m} \frac{I_\varepsilon(x_t^m, x_s^m)}{T_m(T_{m-1}-1)}$
Calculate correlation integral	$C_{1,T}(\varepsilon) = 2 \sum_{t=1}^{T-1} \sum_{s=t+1}^T \frac{I_\varepsilon(x_t^1, x_s^1)}{T(T-1)}$
Form Numerator	$\sqrt{T} [C_{m,T}(\varepsilon) - C_{1,T}(\varepsilon)^m]$
Sample Standard Dev. of Numerator	$\sigma_{m,T}(\varepsilon)$
Form BDS Statistic Distribution	$BDS_{m,T}(\varepsilon) = \frac{\sqrt{T}[C_{m,T}(\varepsilon) - C_{1,T}(\varepsilon)^m]}{\sigma_{m,T}(\varepsilon)}$ $BDS_{m,T}(\varepsilon) \sim N(0, 1)$

Machine Learning In Finance

4. Supervised Learning

SUPERVISED LEARNING OVERVIEW



Supervised Learning

- Notations: $x_i \in \mathbf{R}^d$ i-th training example
- **Model:** how to make prediction \hat{y}_i given x_i
 - Linear model: $\hat{y}_i = \sum_j w_j x_{ij}$ (include linear/logistic regression)
 - The prediction score \hat{y}_i can have different interpretations depending on the task
 - Linear regression: \hat{y}_i is the predicted score
 - Logistic regression: $1/(1 + \exp(-\hat{y}_i))$ is predicted the probability of the instance being positive
 - Others... for example in ranking \hat{y}_i can be the rank score
- **Parameters:** the things we need to learn from data
 - Linear model: $\Theta = \{w_j | j = 1, \dots, d\}$

Supervised Learning

- Objective function that is everywhere

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

Training Loss measures how well model fit on training data

Regularization, measures complexity of model

- Loss on training data: $L = \sum_{i=1}^n l(y_i, \hat{y}_i)$
 - Square loss: $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
 - Logistic loss: $l(y_i, \hat{y}_i) = y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})$
- Regularization: how complicated the model is?
 - L2 norm: $\Omega(w) = \lambda \|w\|^2$
 - L1 norm (lasso): $\Omega(w) = \lambda \|w\|_1$

Supervised Learning

- Ridge regression: $\sum_{i=1}^n (y_i - w^T x_i)^2 + \lambda \|w\|^2$
 - Linear model, square loss, L2 regularization
- Lasso: $\sum_{i=1}^n (y_i - w^T x_i)^2 + \lambda \|w\|_1$
 - Linear model, square loss, L1 regularization
- Logistic regression:
$$\sum_{i=1}^n [y_i \ln(1 + e^{-w^T x_i}) + (1 - y_i) \ln(1 + e^{w^T x_i})] + \lambda \|w\|^2$$
 - Linear model, logistic loss, L2 regularization
- The conceptual separation between model, parameter, objective also gives you **engineering benefits**.
 - Think of how you can implement SGD for both ridge regression and logistic regression

Supervised Learning

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

Training Loss measures how well model fit on training data

Regularization, measures complexity of model

- Why do we want to contain two component in the objective?
- Optimizing training loss encourages **predictive** models
 - Fitting well in training data at least get you close to training data which is hopefully close to the underlying distribution
- Optimizing regularization encourages **simple** models
 - Simpler models tends to have smaller variance in future predictions, making prediction **stable**

Machine Learning In Finance

Supervised Learning

4.1 Classification

Supervised Learning Classification

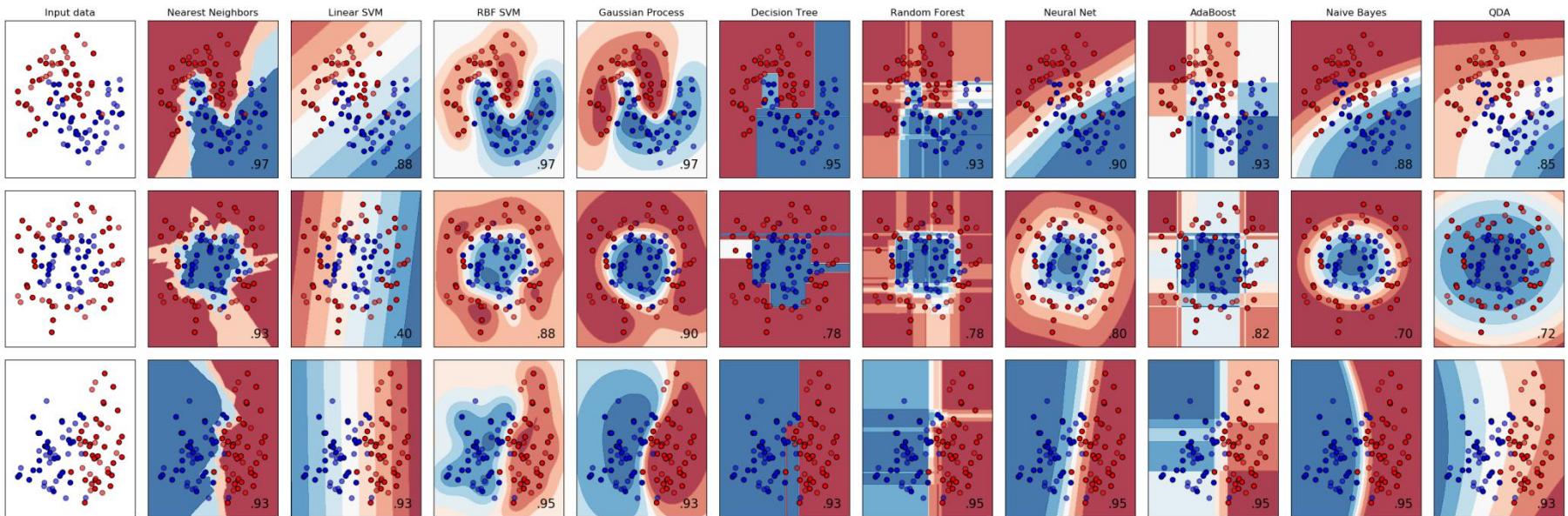
Classification methods of supervised learning have a goal of classifying observations into distinct categories. For instance, we may want the output of a model to be a binary action such as ‘buy market’ or ‘sell market’ based on a number of macro, fundamental, or market input variables. Similarly, we may want to classify asset volatility regimes as “high”, “medium”, and “low volatility”. In this section, we cover the following classification algorithms: logistic regression, Support Vector Machines (SVM), decision trees and random forests, and Hidden Markov Models. Logistic regression is a classification algorithm that produces output as a binary decision, e.g. “buy” or “sell”. It is the simplest adaptation of linear regression to a specific case when the output variable is binary (0 or 1).

Support Vector Machine is one of the most commonly used off-the shelf classification algorithms. It separates data into classes via abstract mathematical processes (mapping a dataset into a higher dimensional space, and then fitting a linear classifier).

Decision trees try to find the optimal rule to forecast an outcome based on a sequence of simple decision steps. Random Forest is a classification Machine Learning method that is based on ‘decision trees’.

Random Forests are averaging simple decision tree models (e.g. calibrated over different historical episodes) and they often yield better and more reliable forecasts as compared to decision trees. The fourth class of algorithms is Hidden Markov Models (HMM). HMMs originate in signal processing theory, and are used in finance to classify asset regimes.

Supervised Learning - Classification



SUPERVISED LEARNING CLASSIFICATION

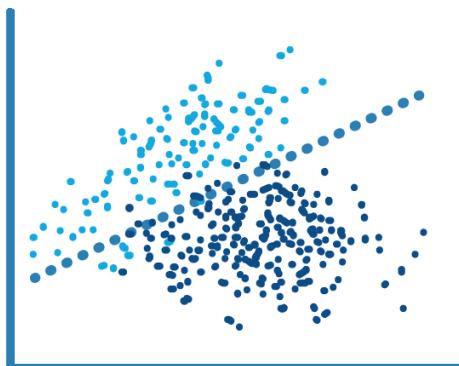
Logistic Regression

How it Works

Fits a model that can predict the probability of a binary response belonging to one class or the other. Because of its simplicity, logistic regression is commonly used as a starting point for binary classification problems.

Best Used...

- When data can be clearly separated by a single, linear boundary
- As a baseline for evaluating more complex classification methods

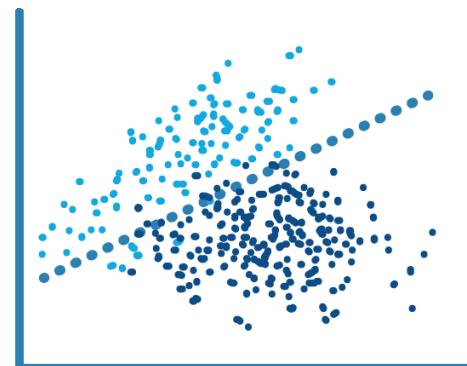


Logistic Regression

Logistic regression can be binomial or multinomial. The **binomial logistic regression** model has the following form

$$p(y|x, w) = \text{Ber}(y|\text{sigm}(w^T x)) \quad (8.1)$$

where w and x are extended vectors, i.e., $w = (b, w_1, w_2, \dots, w_D)$, $x = (1, x_1, x_2, \dots, x_D)$.



SUPERVISED LEARNING CLASSIFICATION

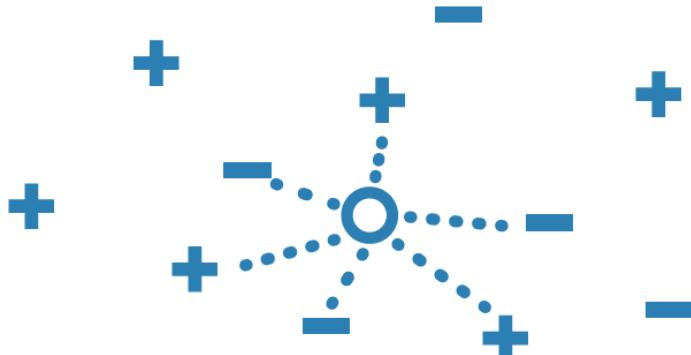
k Nearest Neighbor (kNN)

How it Works

kNN categorizes objects based on the classes of their nearest neighbors in the dataset. kNN predictions assume that objects near each other are similar. Distance metrics, such as Euclidean, city block, cosine, and Chebychev, are used to find the nearest neighbor.

Best Used...

- When you need a simple algorithm to establish benchmark learning rules
- When memory usage and speed of the trained model is not a concern



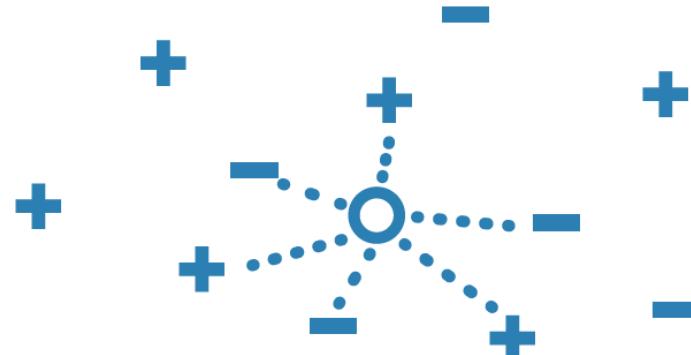
k Nearest Neighbor (kNN)

1.3.2.1 Representation

$$y = f(x) = \arg \min_c \sum_{\mathbf{x}_i \in N_k(\mathbf{x})} \mathbb{I}(y_i = c) \quad (1.6)$$

where $N_k(\mathbf{x})$ is the set of k points that are closest to point \mathbf{x} .

Usually use **k-d tree** to accelerate the process of finding k nearest points.



SUPERVISED LEARNING CLASSIFICATION

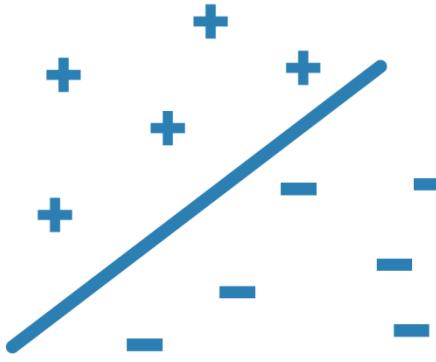
Support Vector Machine (SVM)

How It Works

Classifies data by finding the linear decision boundary (hyperplane) that separates all data points of one class from those of the other class. The best hyperplane for an SVM is the one with the largest margin between the two classes, when the data is linearly separable. If the data is not linearly separable, a loss function is used to penalize points on the wrong side of the hyperplane. SVMs sometimes use a kernel transform to transform nonlinearly separable data into higher dimensions where a linear decision boundary can be found.

Best Used...

- For data that has exactly two classes
- For high-dimensional, nonlinearly separable data
- When you need a classifier that's simple, easy to interpret, and accurate



Support Vector Machine (SVM)

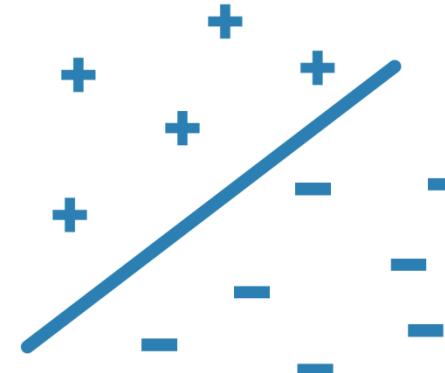
Representation

$$\mathcal{H} : y = f(x) = \text{sign}(wx + b) \quad (14.27)$$

Evaluation

$$\min_{w,b} \quad \frac{1}{2} \|w\|^2 \quad (14.28)$$

$$s.t. \quad y_i(wx_i + b) \geq 1, i = 1, 2, \dots, N \quad (14.29)$$



SUPERVISED LEARNING CLASSIFICATION

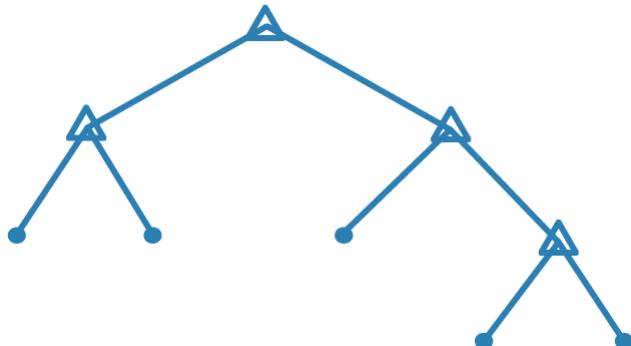
Decision Tree

How it Works

A decision tree lets you predict responses to data by following the decisions in the tree from the root (beginning) down to a leaf node. A tree consists of branching conditions where the value of a predictor is compared to a trained weight. The number of branches and the values of weights are determined in the training process. Additional modification, or pruning, may be used to simplify the model.

Best Used...

- When you need an algorithm that is easy to interpret and fast to fit, to minimize memory usage and when high predictive accuracy is not a requirement



Bagged and Boosted Decision Trees

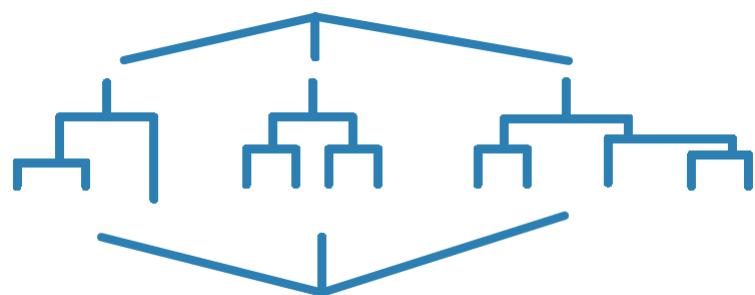
How They Work

In these ensemble methods, several “weaker” decision trees are combined into a “stronger” ensemble.

A bagged decision tree consists of trees that are trained independently on data that is bootstrapped from the input data. Boosting involves creating a strong learner by iteratively adding “weak” learners and adjusting the weight of each weak learner to focus on misclassified examples.

Best Used...

- When predictors are categorical (discrete) or behave nonlinearly
- When the time taken to train a model is less of a concern



SUPERVISED LEARNING CLASSIFICATION

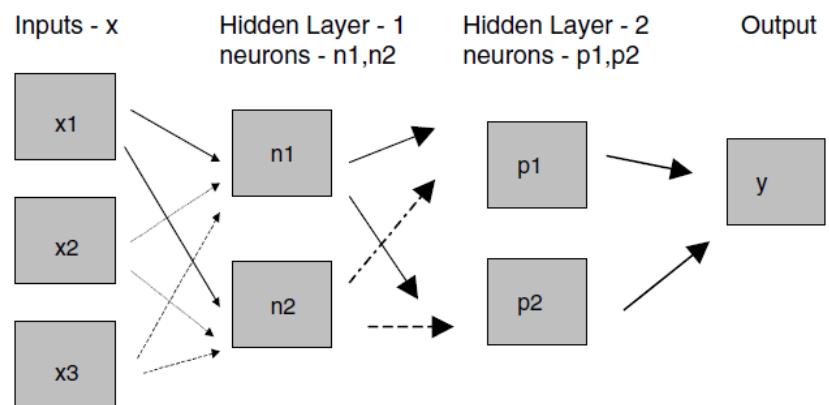
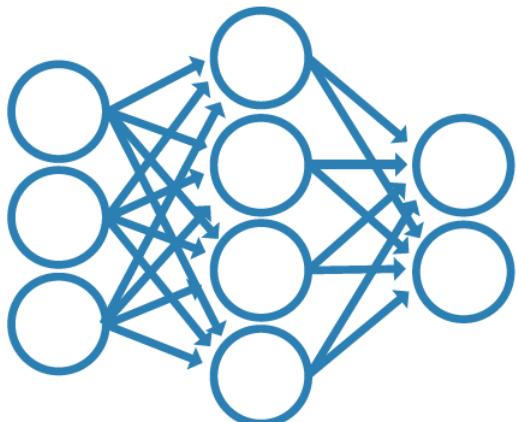
Neural Networks

How it Works

Inspired by the human brain, a neural network consists of highly connected networks of neurons that relate the inputs to the desired outputs. The network is trained by iteratively modifying the strengths of the connections so that given inputs map to the correct response.

Best Used...

- For modeling highly nonlinear systems
- When data is available incrementally and you wish to constantly update the model
- When there could be unexpected changes in your input data
- When model interpretability is not a key concern



$$n_{k,t} = \omega_{k,0} + \sum_{i=1}^{i^*} \omega_{k,i} x_{i,t}$$

$$N_{k,t} = \frac{1}{1 + e^{-n_{k,t}}}$$

$$p_{l,t} = \rho_{l,0} + \sum_{k=1}^{k^*} \rho_{l,k} N_{k,t}$$

$$P_{l,t} = \frac{1}{1 + e^{-p_{l,t}}}$$

$$y_t = \gamma_0 + \sum_{l=1}^{l^*} \gamma_l P_{l,t}$$

SUPERVISED LEARNING CLASSIFICATION

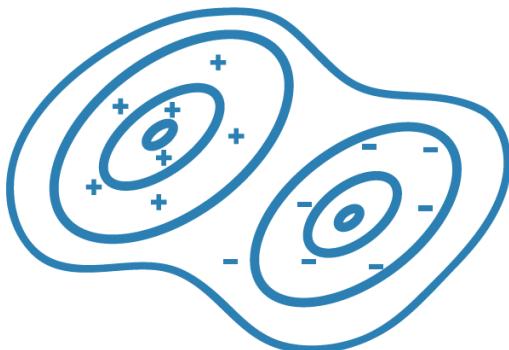
Naïve Bayes

How It Works

A naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. It classifies new data based on the highest probability of its belonging to a particular class.

Best Used...

- For a small dataset containing many parameters
- When you need a classifier that's easy to interpret
- When the model will encounter scenarios that weren't in the training data, as is the case with many financial and medical applications



Naïve Bayes

Assume the features are **conditionally independent** given the class label, then the class conditional density has the following form

$$p(x|y=c, \theta) = \prod_{j=1}^D p(x_j|y=c, \theta_{jc}) \quad (3.33)$$

The resulting model is called a **naïve Bayes classifier**(NBC).

The form of the class-conditional density depends on the type of each feature. We give some possibilities below:

- In the case of real-valued features, we can use the Gaussian distribution: $p(x|y, \theta) = \prod_{j=1}^D \mathcal{N}(x_j|\mu_{jc}, \sigma_{jc}^2)$, where μ_{jc} is the mean of feature j in objects of class c , and σ_{jc}^2 is its variance.
- In the case of binary features, $x_j \in \{0, 1\}$, we can use the Bernoulli distribution: $p(x|y, \theta) = \prod_{j=1}^D \text{Ber}(x_j|\mu_{jc})$, where μ_{jc} is the probability that feature j occurs in class c . This is sometimes called the **multivariate Bernoulli naïve Bayes** model. We will see an application of this below.
- In the case of categorical features, $x_j \in \{a_{j1}, a_{j2}, \dots, a_{js_j}\}$, we can use the multinomial distribution: $p(x|y, \theta) = \prod_{j=1}^D \text{Cat}(x_j|\mu_{jc})$, where μ_{jc} is a histogram over the K possible values for x_j in class c .

Obviously we can handle other kinds of features, or use different distributional assumptions. Also, it is easy to mix and match features of different types.



Statistical decision theory

Let C be a random variable giving the class label of an observation in our data set. A natural rule would be to classify according to

$$f(x) = \operatorname{argmax}_{j=1,\dots,K} P(C = j | X = x)$$

This predicts the most likely class, given the feature measurements $X = x \in \mathbb{R}^p$

This is called the Bayes classifier, and it is the best that we can do (think of overlapping classes).

We can use the Bayes rule to write

$$P(C = j | X = x) = \frac{P(X = x | C = j) \cdot P(C = j)}{P(X = x)}$$

The denominator is always the same

$$f(x) = \operatorname{argmax}_{j=1,\dots,K} P(X = x | C = j) \cdot \pi_j$$

Machine Learning in Finance

4.1 Supervised Learning

Classification – Logistic Regression

SUPERVISED LEARNING CLASSIFICATION

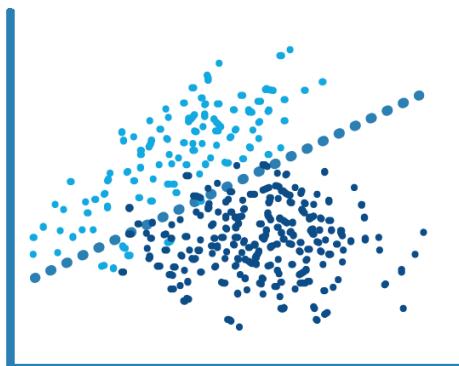
Logistic Regression

How it Works

Fits a model that can predict the probability of a binary response belonging to one class or the other. Because of its simplicity, logistic regression is commonly used as a starting point for binary classification problems.

Best Used...

- When data can be clearly separated by a single, linear boundary
- As a baseline for evaluating more complex classification methods

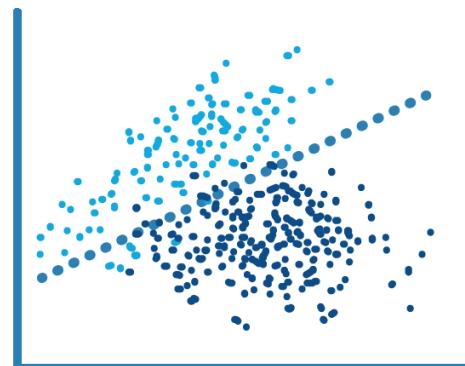


Logistic Regression

Logistic regression can be binomial or multinomial. The **binomial logistic regression** model has the following form

$$p(y|x, w) = \text{Ber}(y|\text{sigm}(w^T x)) \quad (8.1)$$

where w and x are extended vectors, i.e., $w = (b, w_1, w_2, \dots, w_D)$, $x = (1, x_1, x_2, \dots, x_D)$.



4.1.1 Logistic Regression

Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function.

The implementation of logistic regression in scikit-learn can be accessed from class LogisticRegression. This implementation can fit binary, One-vs- Rest, or multinomial logistic regression with optional L2 or L1 regularization.

As an optimization problem, binary class L2 penalized logistic regression minimizes the following cost function:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^n \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Similarly, L1 regularized logistic regression solves the following optimization problem

$$\min_{w,c} \|w\|_1 + C \sum_{i=1}^n \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Note that, in this notation, it's assumed that the observation y_i takes values in the set $-1, 1$ at trial i .

Logistic Regression

Logistic Regression: Logistic regression (or logit in econometrics) is used to forecast the probability of an event given historical sample data. For example, we may want to forecast the direction of next-day price movement given the observation of other market returns or value of quant signals today. By mapping the probability to either 0 or 1, logistic regression can also be used for classification problems, where we have to choose to either buy (mapped to 0) or sell (mapped to 1).

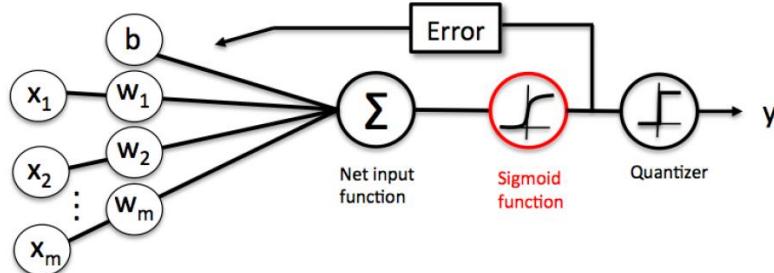
Logistic regression is derived via a simple change to ordinary linear regression.

We first form a linear combination of the input variables (as in conventional regression), and then apply a function that maps this number to a value between 0 and 1.

The mapping function is called the logistic function, and has the simple shape shown below (left). An alternative to the logistic regression would have to simply 'quantize' the output of a linear regressor to either 0 or 1, i.e. if the value was less than 0.5, we declare the output to be 0; and if its greater than 0.5, we declare it to be 1.

It turns out that such an approach is very sensitive to outliers. As shown in the figure below (right), the outliers cause the line to have an extremely low slope, which in turn leads to many points in the set being classified as 0, instead of being classified correctly as 1.

Logistic vs Softmax Regression



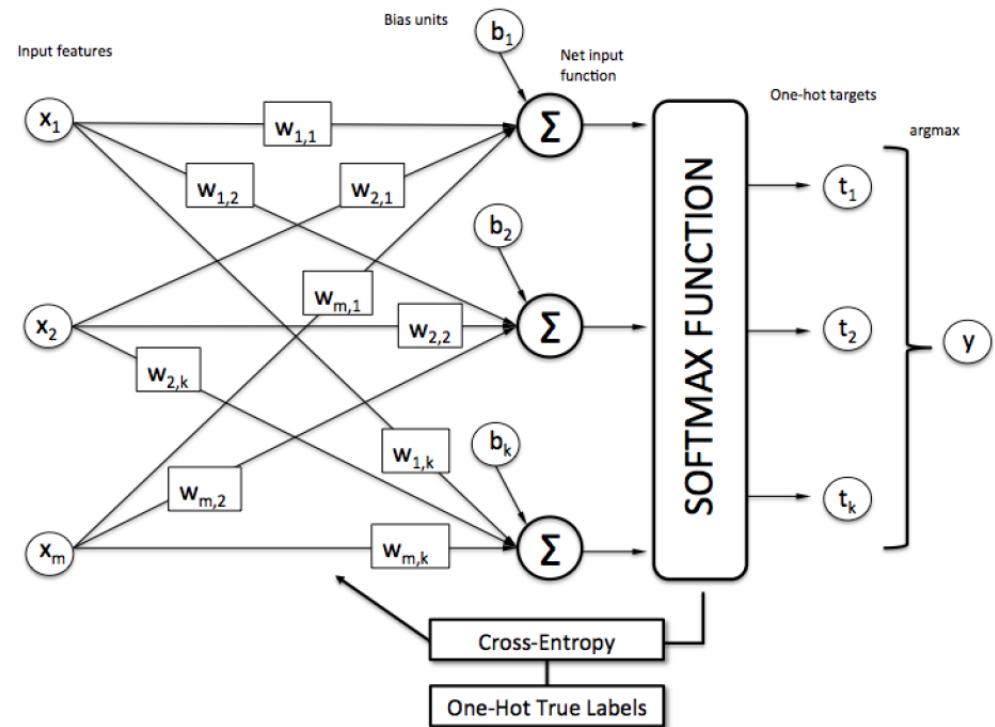
$$\phi(z) = \frac{1}{1 + e^{-z}},$$

where z is defined as the net input

$$z = w_0 x_0 + w_1 x_1 + \dots + w_m x_m = \sum_{j=0}^m w_j x_j = \mathbf{w}^T \mathbf{x}.$$

The net input is in turn based on the logit function

$$\text{logit}(p(y = 1 \mid \mathbf{x})) = z.$$



In Softmax Regression (SMR), we replace the sigmoid logistic function by the so-called softmax function $\phi_{\text{softmax}}(\cdot)$.

$$P(y = j \mid z^{(i)}) = \phi_{\text{softmax}}(z^{(i)}) = \frac{e^{z^{(i)}}}{\sum_{k=0}^k e^{z_k^{(i)}}},$$

where we define the net input z as

$$z = w_1 x_1 + \dots + w_m x_m + b = \sum_{l=1}^m w_l x_l + b = \mathbf{w}^T \mathbf{x} + b.$$

Machine Learning In Finance

Supervised Learning

Classification

4.1.2 K Nearest

SUPERVISED LEARNING CLASSIFICATION

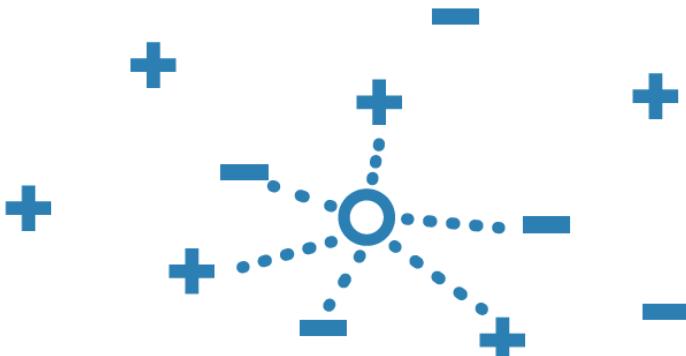
k Nearest Neighbor (kNN)

How it Works

kNN categorizes objects based on the classes of their nearest neighbors in the dataset. kNN predictions assume that objects near each other are similar. Distance metrics, such as Euclidean, city block, cosine, and Chebychev, are used to find the nearest neighbor.

Best Used...

- When you need a simple algorithm to establish benchmark learning rules
- When memory usage and speed of the trained model is not a concern



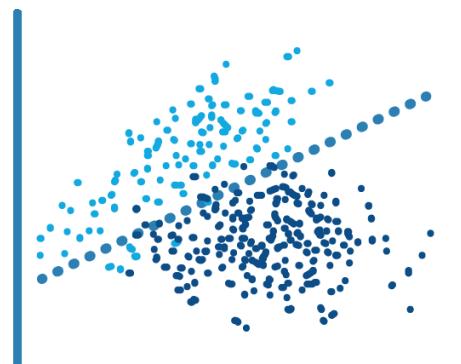
Logistic Regression

How it Works

Fits a model that can predict the probability of a binary response belonging to one class or the other. Because of its simplicity, logistic regression is commonly used as a starting point for binary classification problems.

Best Used...

- When data can be clearly separated by a single, linear boundary
- As a baseline for evaluating more complex classification methods



4.1.2 Nearest Neighbors

Neighbors-based classification is a type of instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

scikit-learn implements two different nearest neighbors classifiers: `KNeighborsClassifier` implements learning based on the nearest neighbors of each query point, where k is an integer value specified by the user. `RadiusNeighborsClassifier` implements learning based on the number of neighbors within a fixed radius of each training point, where r is a floating-point value specified by the user.

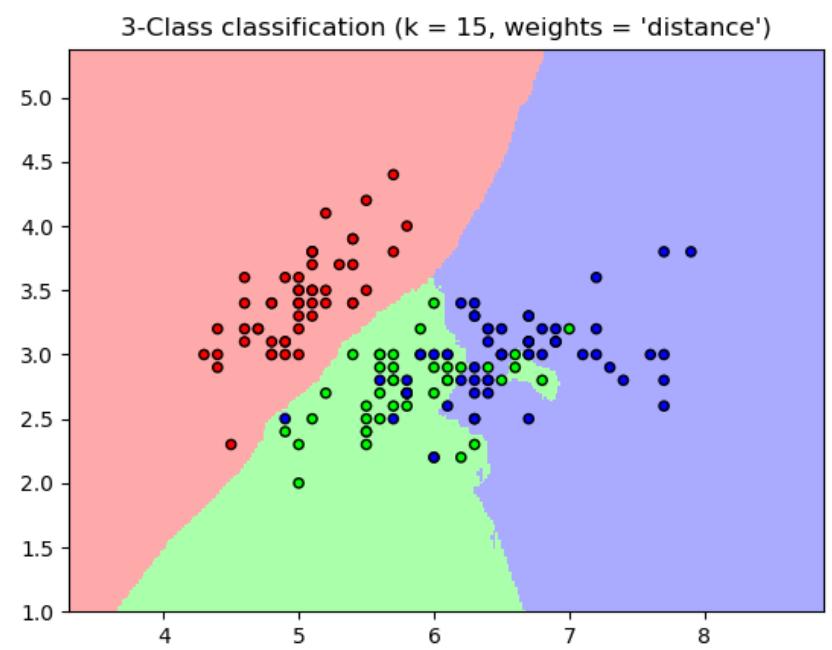
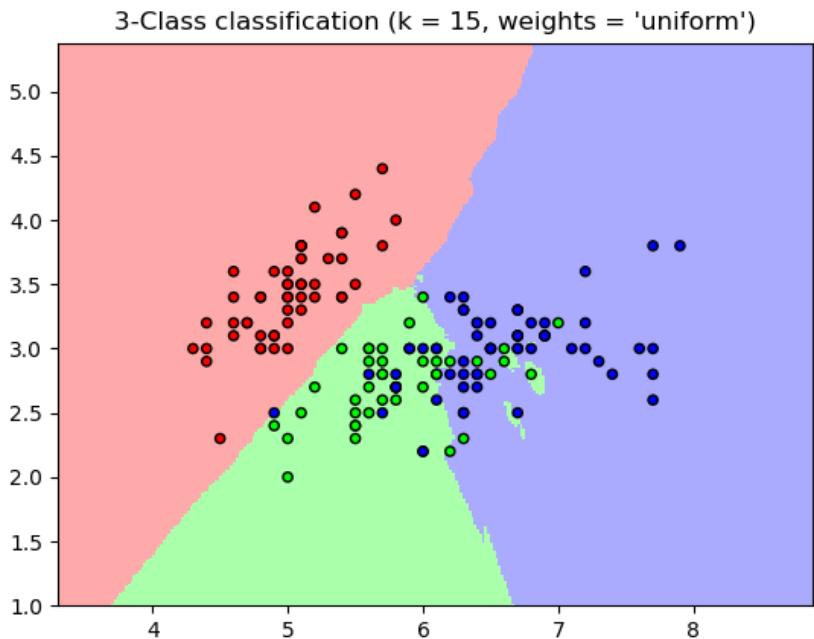
4.1.2 Nearest Neighbors

The K-neighbors classification in `KNeighborsClassifier` is the most commonly used technique. The optimal choice of the value `n_neighbors` is highly data-dependent: in general a larger `n_neighbors` suppresses the effects of noise, but makes the classification boundaries less distinct.

In cases where the data is not uniformly sampled, radius-based neighbors classification in `RadiusNeighborsClassifier` can be a better choice. The user specifies a fixed radius `r`, such that points in sparser neighborhoods use fewer nearest neighbors for the classification. For high-dimensional parameter spaces, this method becomes less effective due to the so-called “curse of dimensionality”.

The basic nearest neighbors classification uses uniform weights: that is, the value assigned to a query point is computed from a simple majority vote of the nearest neighbors. Under some circumstances, it is better to weight the neighbors such that nearer neighbors contribute more to the fit. This can be accomplished through the `weights` keyword. The default value, `weights = 'uniform'`, assigns uniform weights to each neighbor. `weights = 'distance'` assigns weights proportional to the inverse of the distance from the query point. Alternatively, a user-defined function of the distance can be supplied to compute the weights.

4.1.2 Nearest Neighbors

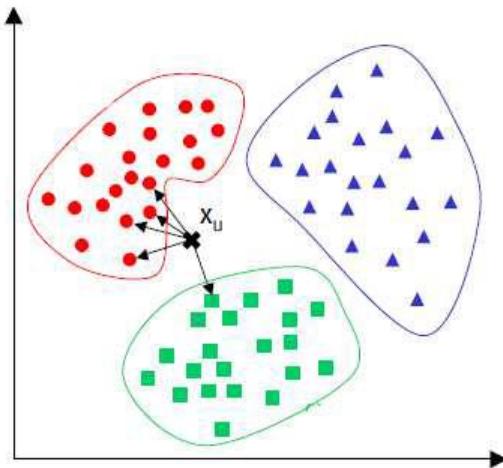


4.1.2 Nearest Neighbors

```
>>> # Split iris data in train and test data
>>> # A random permutation, to split the data randomly
>>> np.random.seed(0)
>>> indices = np.random.permutation(len(iris_X))
>>> iris_X_train = iris_X[indices[:-10]]
>>> iris_y_train = iris_y[indices[:-10]]
>>> iris_X_test = iris_X[indices[-10:]]
>>> iris_y_test = iris_y[indices[-10:]]
>>> # Create and fit a nearest-neighbor classifier
>>> from sklearn.neighbors import KNeighborsClassifier
>>> knn = KNeighborsClassifier()
>>> knn.fit(iris_X_train, iris_y_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
>>> knn.predict(iris_X_test)
array([1, 2, 1, 0, 0, 0, 2, 1, 2, 0])
>>> iris_y_test
array([1, 1, 1, 0, 0, 0, 2, 1, 2, 0])
```

K-Nearest Neighbor (K-NN)

- Given training data $D = \{(x_1, y_1), \dots, (x_N, y_N)\}$ and a test point
- Prediction Rule: Look at the K most similar training examples



- For classification: assign the majority class label (majority voting)
- For regression: assign the average response
- The algorithm requires:
 - Parameter K: number of nearest neighbors to look for
 - Distance function: To compute the similarities between examples
- Special Case: 1-Nearest Neighbor

K-Nearest Neighbors : Computing the distances

- The K-NN algorithm requires computing distances of the test example from each of the training examples
- Several ways to compute distances
- The choice depends on the type of the features in the data
- Real-valued features ($\mathbf{x}_i \in \mathbb{R}^D$) : Euclidean distance is commonly used

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{m=1}^D (x_{im} - x_{jm})^2} = \sqrt{\|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2 - 2\mathbf{x}_i^T \mathbf{x}_j}$$

- Generalization of the distance between points in 2 dimensions
- $\|\mathbf{x}_i\| = \sqrt{\sum_{m=1}^D x_{im}^2}$ is called the norm of \mathbf{x}_i . Norm of a vector \mathbf{x} is also its length
- $\mathbf{x}_i^T \mathbf{x}_j = \sum_{m=1}^D x_{im} x_{jm}$ is called the dot (or inner) product of \mathbf{x}_i and \mathbf{x}_j
 - Dot product measures the similarity between two vectors (orthogonal vectors have dot product=0, parallel vectors have high dot product)

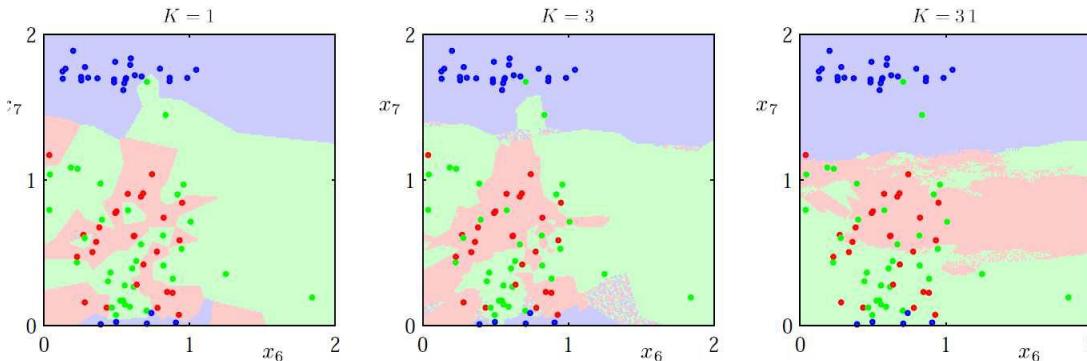
K-Nearest Neighbors : Some other distance measures

- Binary-valued features
 - Use Hamming distance: $d(x_i, x_j) = \sum_{m=1}^D \mathbb{I}(x_{im} \neq x_{jm})$
 - Hamming distance counts the number of features where the two examples disagree
- Mixed feature types (some real-valued and some binary-valued)?
 - Can use mixed distance measures
 - E.g., Euclidean for the real part, Hamming for the binary part
- Can also assign weights to features: $d(x_i, x_j) = \sum_{m=1}^D w_m d(x_{im}, x_{jm})$

K-Nearest Neighbors Feature Normalization

- Note: Features should be on the same scale
- Example: if one feature has its values in millimeters and another has in centimeters, we would need to normalize
- One way is:
- Replace x_{im} by $z_{im} = \frac{(x_{im} - \bar{x}_m)}{\sigma_m}$ (make them zero mean, unit variance)
- $\bar{x}_m = \frac{1}{N} \sum_{i=1}^N x_{im}$: empirical mean of mth feature
- $\sigma_m^2 = \frac{1}{N} \sum_{i=1}^N (x_{im} - \bar{x}_m)^2$: empirical variance of mth feature

K-Nearest Neighbors : Neighborhood Size



- **Small K**
 - Creates many small regions for each class
 - May lead to non-smooth) decision boundaries and overfit
- **Large K**
 - Creates fewer larger regions
 - Usually leads to smoother decision boundaries (caution: too smooth decision boundary can underfit)
- **Choosing K**
 - Often data dependent and heuristic based
 - Or using cross-validation (using some held-out data)
 - In general, a K too small or too big is bad!

K-Nearest Neighbors : Conclusions

- Pros
 - Simple and intuitive; easily implementable
 - Asymptotically consistent (a theoretical property)
 - With infinite training data and large enough K, K-NN approaches the best possible classifier (Bayes optimal)
- Cons
- Store all the training data in memory even at test time
 - Can be memory intensive for large training datasets
 - An example of non-parametric, or memory/instance-based methods
 - Different from parametric, model-based learning models
- Expensive at test time: $O(ND)$ computations for each test point
 - Have to search through all training data to find nearest neighbors
 - Distance computations with N training points (D features each)
- Sensitive to noisy features
- May perform badly in high dimensions (curse of dimensionality)
 - In high dimensions, distance notions can be counter-intuitive

Machine Learning In Finance

Supervised Learning

Classification

4.1.3 Decision Trees

SUPERVISED LEARNING CLASSIFICATION

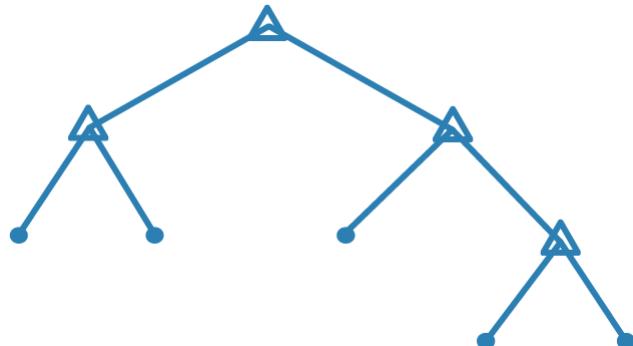
Decision Tree

How it Works

A decision tree lets you predict responses to data by following the decisions in the tree from the root (beginning) down to a leaf node. A tree consists of branching conditions where the value of a predictor is compared to a trained weight. The number of branches and the values of weights are determined in the training process. Additional modification, or pruning, may be used to simplify the model.

Best Used...

- When you need an algorithm that is easy to interpret and fast to fit, to minimize memory usage and when high predictive accuracy is not a requirement



Bagged and Boosted Decision Trees

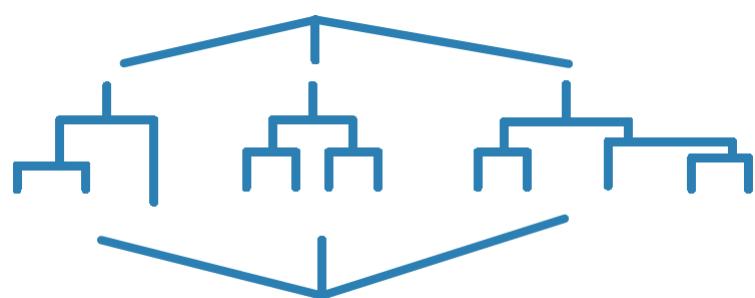
How They Work

In these ensemble methods, several “weaker” decision trees are combined into a “stronger” ensemble.

A bagged decision tree consists of trees that are trained independently on data that is bootstrapped from the input data. Boosting involves creating a strong learner by iteratively adding “weak” learners and adjusting the weight of each weak learner to focus on misclassified examples.

Best Used...

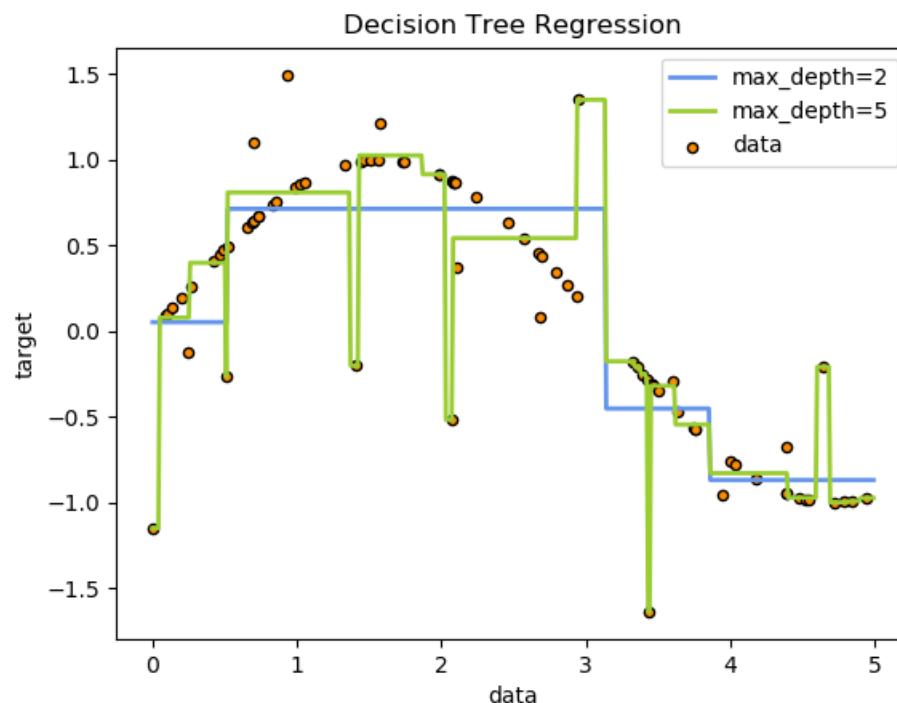
- When predictors are categorical (discrete) or behave nonlinearly
- When the time taken to train a model is less of a concern



Decision Trees

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

For instance, in the example below, decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model.



Decision Trees - Disadvantages

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning (not currently supported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Decision Trees - Advantages

- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data. Other techniques are usually specialised in analysing datasets that have only one type of variable. See algorithms for more information.
- Able to handle multi-output problems.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

CART Example

The explanatory variables are a set of technical indicators derived from the initial daily OHLC dataset. Each indicator represents a well-documented market behavior. In order to reduce the noise in the data and to try to identify robust relationships, each independent variable is considered to have a binary outcome.

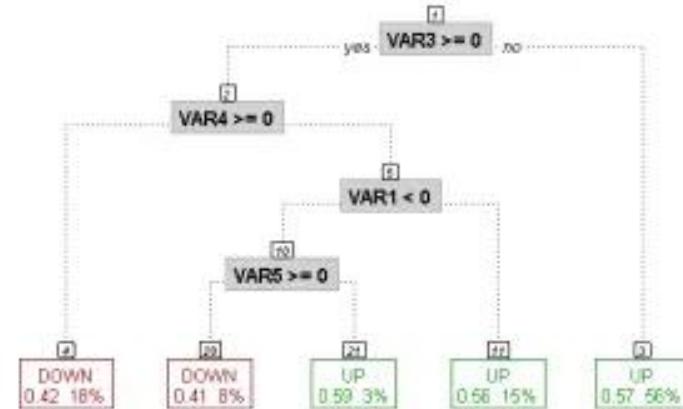
- **Volatility (VAR1):** High volatility is usually associated with a down market and low volatility with an up market. Volatility is defined as the 20 days raw ATR (Average True Range) spread to its moving average (MA). If raw ATR > MA then VAR1 = 1, else VAR1 = -1.

- **Short term momentum (VAR2):** The equity market exhibits short term momentum behavior captured here by a 5 days simple moving averages (SMA). If Price > SMA then VAR2 = 1 else VAR2 = -1

- **Long term momentum (VAR3):** The equity market exhibits long term momentum behavior captured here by a 50 days simple moving averages (LMA). If Price > LMA then VAR3 = 1 else VAR3 = -1

- **Short term reversal (VAR4):** This is captured by the CRTDR which stands for Close Relative To Daily Range and calculated as following: . If CRTDR > 0.5, then VAR4 = 1 else VAR4 = -1

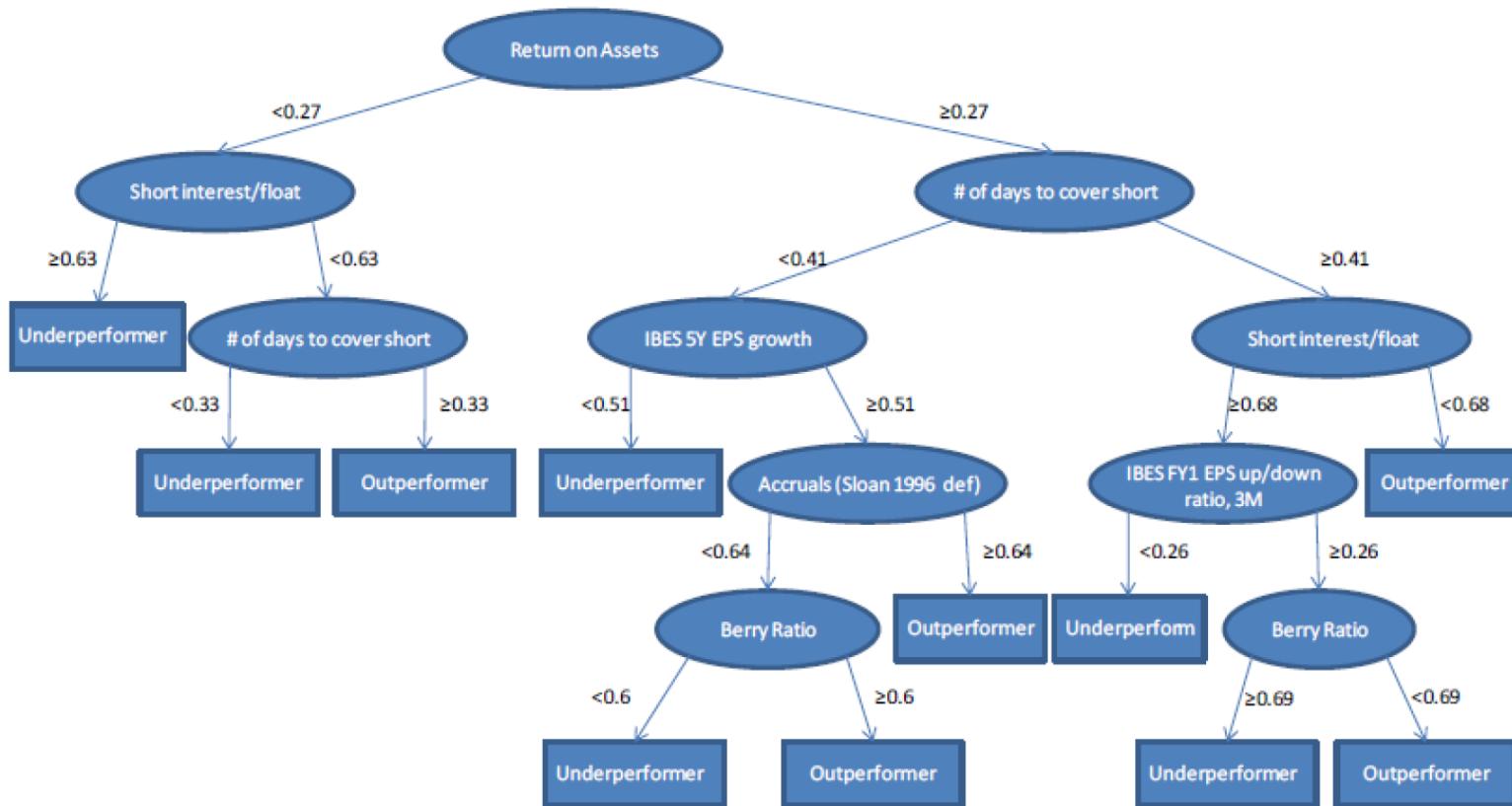
- **Autocorrelation regime (VAR5):** The equity market tends to go through periods of negative and positive autocorrelation regimes. If returns autocorrelation over the last 5 days > 0 then VAR5 = 1 else VAR5 = -1



In the tree above, the path to reach node #4 is: VAR3 ≥ 0 (Long Term Momentum ≥ 0) and VAR4 ≥ 0 (CRTDR ≥ 0). The red rectangle indicates this is a DOWN leaf (e.g., terminal node) with a probability of 58% ($1 - 0.42$). In market terms this means that if Long Term Momentum is Up and CRTDR is > 0.5 then the probability of a positive return next week is 42% based on the in sample sample data. 18% indicates the proportion of the data set that falls into that terminal node (e.g., leaf)

Classification and Regression Trees

CART stands for classification and regression tree. It is a simple machine learning technique that seeks to classify data into binary branches or trees, by applying hierarchical splits based on a set of explanatory variables.



Decision Trees

- Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. It is one of the predictive modeling approaches used in statistics, data mining and machine learning.
- Tree models where the target variable can take a finite set of values are called classification trees. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.
- In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining, a decision tree describes data but not decisions; rather the resulting classification tree can be an input for decision making. This page deals with decision trees in data mining.

Decision Trees

- Some techniques, often called ensemble methods, construct more than one decision tree:
- A Random Forest classifier uses a number of decision trees, in order to improve the classification rate.
- Boosted Trees can be used for regression-type and classification-type problems.
- Rotation forest - in which every decision tree is trained by first applying principal component analysis (PCA) on a random subset of the input features.

Decision Trees

Decision trees are formed by a collection of rules based on variables in the modeling data set:

- Rules based on variables' values are selected to get the best split to differentiate observations based on the dependent variable
- Once a rule is selected and splits a node into two, the same process is applied to each "child" node (i.e. it is a recursive procedure)
- Splitting stops when CART detects no further gain can be made, or some pre-set stopping rules are met. (Alternatively, the data are split as much as possible and then the tree is later pruned.)

Each branch of the tree ends in a terminal node. Each observation falls into one and exactly one terminal node, and each terminal node is uniquely defined by a set of rules.

Classification and regression trees (CART) are a non-parametric decision tree learning technique that produces either classification or regression trees, depending on whether the dependent variable is categorical or numeric, respectively.

Decision Trees

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represents classification rules.

In decision analysis a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.

A decision tree consists of 3 types of nodes:

- Decision nodes - commonly represented by squares
- Chance nodes - represented by circles
- End nodes - represented by triangle

Decision Trees

- implementations developed by Ross Quinlan (Quinlan, J. R. 1986. Induction of Decision Trees. *Mach. Learn.* 1, 1 (Mar. 1986), 81-106.)
- CART, or Classification And Regression Trees is often used as a generic acronym for the term Decision Tree, though it apparently has a more specific meaning. In sum, the CART implementation is very similar to C4.5; the one notable difference is that CART constructs the tree based on a numerical splitting criterion recursively applied to the data, whereas C4.5 includes the intermediate step of constructing *rule set*s.

Decision Trees

- **ID3, or Iterative Dichotomizer**, was the first of three Decision Tree implementations developed by Ross Quinlan (Quinlan, J. R. 1986. Induction of Decision Trees. *Mach. Learn.* 1, 1 (Mar. 1986), 81-106.)
- **CART, or Classification And Regression Trees** is often used as a generic acronym for the term Decision Tree, though it apparently has a more specific meaning. In sum, the CART implementation is very similar to **C4.5**; the one notable difference is that CART constructs the tree based on a numerical splitting criterion recursively applied to the data, whereas C4.5 includes the intermediate step of constructing *rule set*s.

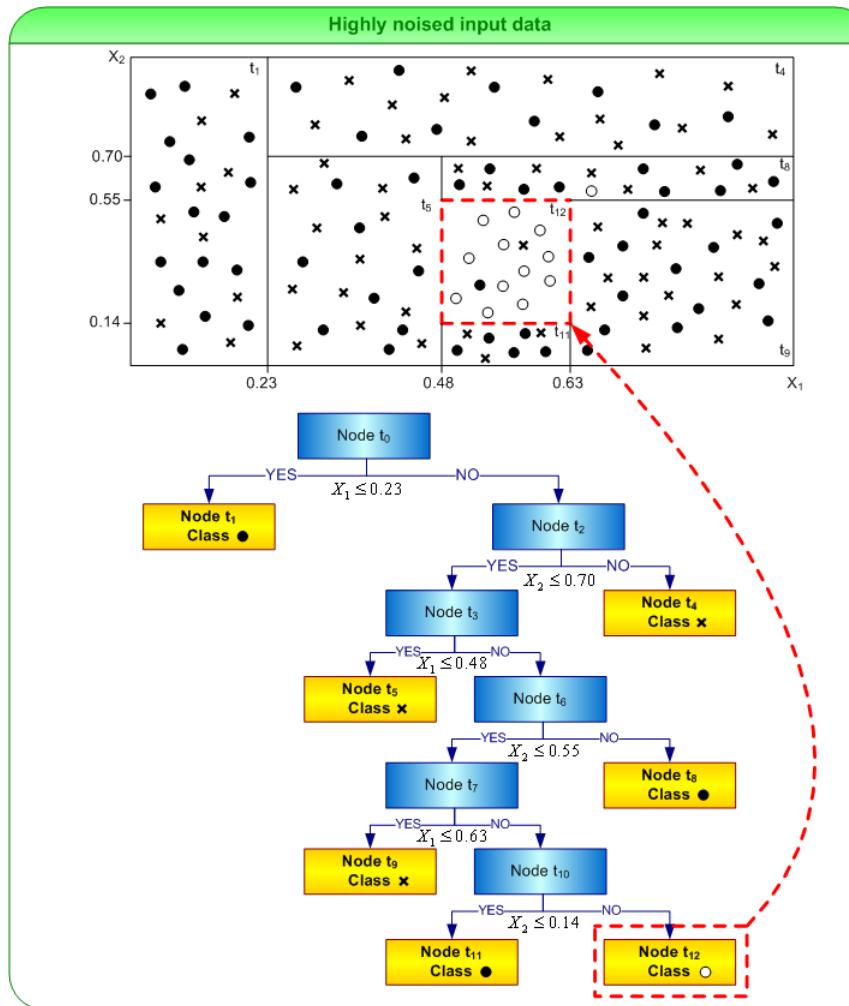
Decision Trees

- **C4.5, Quinlan's next iteration.** The new features (versus ID3) are: (i) accepts both continuous and discrete features; (ii) handles incomplete data points; (iii) solves over-fitting problem by (very clever) bottom-up technique usually known as "pruning"; and (iv) different weights can be applied the features that comprise the training data. Of these, the first three are very important-- and i would suggest that any DT implementation you choose have all three. The fourth (differential weighting) is much less important
- **C5.0**, the most recent Quinlan iteration. This implementation is covered by patent and probably as a result, is rarely implemented (outside of commercial software packages).

Decision Trees

- **CHAID (chi-square automatic interaction detector)** actually predates the original ID3 implementation by about six years (published in a Ph.D thesis by Gordon Kass in 1980). I know very little about this technique. The R Platform has a Package called CHAID which includes excellent documentation
- **MARS (multi-adaptive regression splines)** is actually a term trademarked by the original inventor of MARS, Salford Systems. As a result, MARS clones in libraries not sold by Salford are named something other than MARS--e.g., in R, the relevant function is polymars in the polyspline library. Matlab and Statistica also have implementations with MARS-functionality

Financial Applications of Classification and Regression Trees Paper - Anton Andriyashin



Financial Applications of Classification and Regression Trees Paper - Anton Andriyashin

$$\left\{ \begin{array}{l} \sum_{t=\tau}^T \sum_{i=1}^{N(t)} w_{it} [E(R_{it} | F_{i,t-1}) (1 - I\{R_{it} > 0\} \cdot I\{\pi_{it} = \pi_{i,t-1} \neq 0\} + \\ + I\{R_{it} < 0\} \cdot I\{\pi_{it} = \pi_{i,t-1} \neq 0\})] \rightarrow \max_{\pi_{it}} \\ f_{it} : F_{i,(t-1)} \rightarrow R_{it} \\ \pi_{it} \in \mathbb{S} = \{-1; 0; 1\} \\ R_{it} \in \mathbb{A} \\ \sum_{i=1}^{N(t)} [1 + 0.001 \cdot I\{\pi_{it} = \pi_{i,t-1} \neq 0\}] \cdot \pi_{it} P_{it} Q_{it} = \bar{M} \\ w_{it} = \frac{1}{N(t)} \pi_{it}^* (F_{i,t-1}) \\ F_{i,T} \supset F_{i,T-1} \supset \dots \supset F_{i,\tau} \quad \forall i = \overline{1, N(t)} \end{array} \right.$$

Financial Applications of Classification and Regression Trees Paper - Anton Andriyashin

Variable name	Variable type	Regularity estimate
$\frac{P_{t-1} - P_{t-2}}{P_{t-2}}$	Fund./Tech.	1 day
$Sales_{it}$	Fundamental	1 day
$\frac{P_{it}}{CF_{it}}$	Fundamental	1 day
$\frac{P_{it}}{EPS_{it}}$	Fundamental	1 day
$\Delta_{12} \frac{EPS_{it}}{P_{it}}$	Fundamental	1 day
ROE_{it}	Technical	1 day
$Momentum_{it}$	Technical	1 day
$Stochastic_{it}$	Technical	1 day
$\frac{MA_{it}}{P_{it}}$	Technical	1 day
$MACD_{it}$	Technical	1 day
$\sigma\left(\frac{MA_{it}}{P_{it}}\right)$	Technical	1 day
ROC_{it}	Technical	1 day
$TRIX_{it}$	Technical	1 day

Machine Learning In Finance

Supervised Learning

Classification

4.1.4 Support Vector Machines

SUPERVISED LEARNING CLASSIFICATION

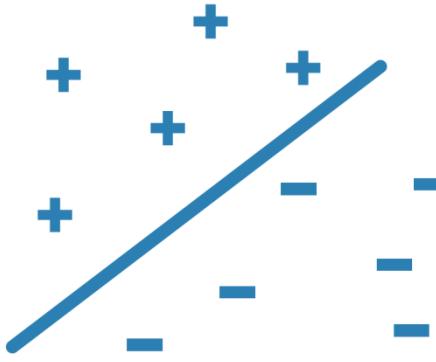
Support Vector Machine (SVM)

How It Works

Classifies data by finding the linear decision boundary (hyperplane) that separates all data points of one class from those of the other class. The best hyperplane for an SVM is the one with the largest margin between the two classes, when the data is linearly separable. If the data is not linearly separable, a loss function is used to penalize points on the wrong side of the hyperplane. SVMs sometimes use a kernel transform to transform nonlinearly separable data into higher dimensions where a linear decision boundary can be found.

Best Used...

- For data that has exactly two classes
- For high-dimensional, nonlinearly separable data
- When you need a classifier that's simple, easy to interpret, and accurate



Support Vector Machine (SVM)

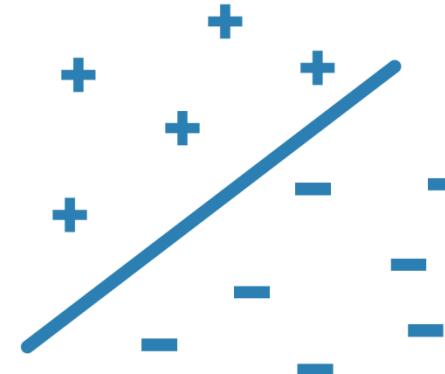
Representation

$$\mathcal{H} : y = f(x) = \text{sign}(wx + b) \quad (14.27)$$

Evaluation

$$\min_{w,b} \quad \frac{1}{2} \|w\|^2 \quad (14.28)$$

$$s.t. \quad y_i(wx_i + b) \geq 1, i = 1, 2, \dots, N \quad (14.29)$$



Support Vector Machines

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

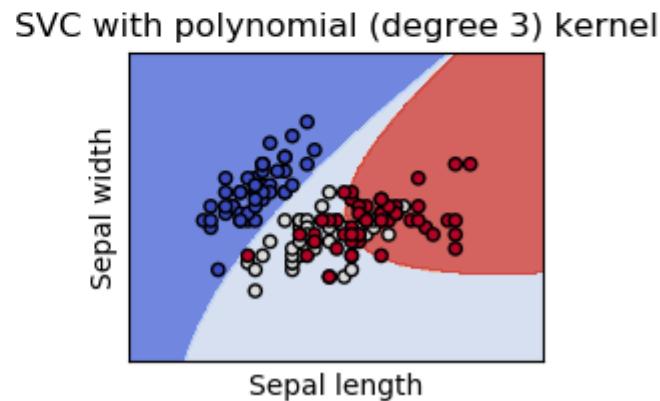
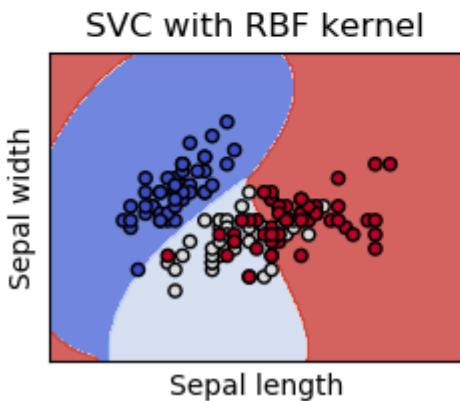
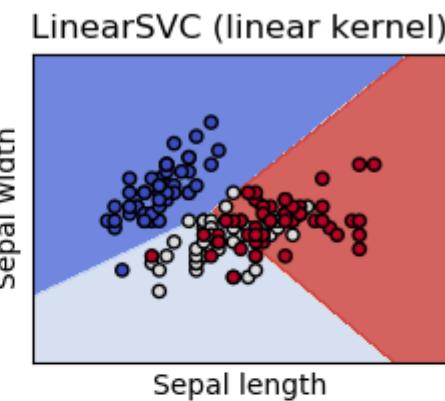
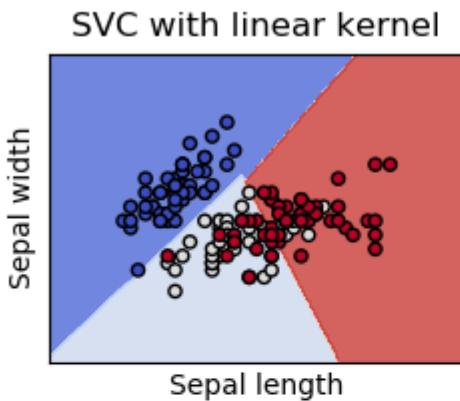
- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

The support vector machines in scikit-learn support both dense (`numpy.ndarray` and convertible to that by `numpy.asarray`) and sparse (any `scipy.sparse`) sample vectors as input. However, to use an SVM to make predictions for sparse data, it must have been fit on such data. For optimal performance, use C-ordered `numpy.ndarray` (dense) or `scipy.sparse.csr_matrix` (sparse) with `dtype=float64`.

Support Vector Machines



Support Vector Classifier – Mathematics

Given training vectors $x_i \in \mathbb{R}^p, i = 1, \dots, n$, in two classes, and a vector $y \in \{1, -1\}^n$ SVC solves the following primal problem:

$$\min_{w, b, \zeta} \frac{1}{2} w^T w + C \sum_{i=1}^n \zeta_i$$

$$\text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i$$

$$\zeta_i \geq 0, i = 1, \dots, n$$

Its dual is

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \text{ subject to } y^T \alpha = 0, 0 \leq \alpha_i \leq C, i = 1, \dots, n$$

Where e is the vector of all ones, $C > 0$ is the upper bound, Q is n by n positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i x_j)$ where $K(x_i x_j) = \phi(x_i)^T \phi(x_j)$ is the kernel. Here training vectors are implicitly mapped into a higher (maybe infinite) dimensional space by the function ϕ .

The decision function is

$$\operatorname{sgn}\left(\sum_{i=1}^n y_i \alpha_i K(x_i, x) + \rho\right)$$

Support Vector Classifier – Mathematics

The kernel function can be any of the following:

- linear: $\langle x, x' \rangle$.
- polynomial: $(\gamma \langle x, x' \rangle + r)^d$. d is specified by keyword degree, r by coef0.
- Radiab basis function: $\exp(-\gamma \|x - x'\|^2)$. γ is specified by keyword gamma, must be greater than 0.
- Sigmoid : $(\tanh(\gamma \langle x, x' \rangle + r))$, where r is specified by coef0.

Different kernels are specified by keyword kernel at initialization:

```
>>> linear_svc = svm.SVC(kernel='linear')
>>> linear_svc.kernel
'linear'
>>> rbf_svc = svm.SVC(kernel='rbf')
>>> rbf_svc.kernel
'rbf'
```

Support Vector Regression – Mathematics

Given training vectors $x_i \in \mathbb{R}^p, i = 1, \dots, n$, in two classes, and a vector $y \in \{1, -1\}^n$ ε -SVC solves the following primal problem:

$$\min_{w, b, \zeta, \zeta^*} \frac{1}{2} w^T w + C \sum_{i=1}^n (\zeta_i + \zeta_i^*)$$

$$\text{subject to } y_i - w^T \phi(x_i) + b \leq \varepsilon + \zeta_i,$$

$$w^T \phi(x_i) + b - y_i \leq \varepsilon + \zeta_i^*$$

$$\zeta_i + \zeta_i^* \geq 0, i = 1, \dots, n$$

Support Vector Regression – Mathematics

Its dual is

$$\min_{\alpha, \alpha^*} \frac{1}{2} (\alpha - \alpha^*)^T Q (\alpha - \alpha^*) + \epsilon \epsilon^T (\alpha - \alpha^*) - y^T (\alpha - \alpha^*)$$

$$\text{subject to } \epsilon^T (\alpha - \alpha^*) = 0$$

$$0 \leq \alpha_i, \alpha_1^* \leq C, i = 1, \dots, n$$

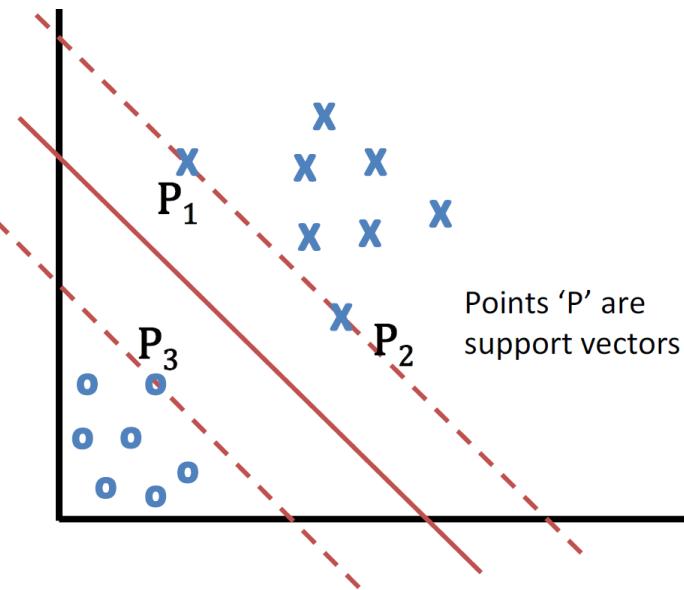
$$\text{subject to } y^T \alpha = 0, 0 \leq \alpha_i \leq C, i = 1, \dots, n$$

Where e is the vector of all ones , $C > 0$ is the upper bound, Q is n by n positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i x_j)$ where $K(x_i x_j) = \phi(x_i)^T \phi(x_j)$ is the kernel. Here training vectors are implicitly mapped into a higher (maybe infinite) dimensional space by the function ϕ .

The decision function is

$$\sum_{i=1}^n (\alpha_i, \alpha_i^*) K(x_i, x) + \rho$$

Support Vector Machines



The line “l” can be determined just from the border points of the two sets. Points P₁ and P₂ create a border line for the crosses and we drew a parallel line through P₃ as well. These border points P₁, P₂ and P₃ which alone determine the optimal separating line “l” are called ‘support vectors’. This was a very simple illustration of a linear classification when the input variables can be visualized on a chart. Mathematics and visualization become significantly more complex when we are dealing with a large number of input variables. One can improve performance of a linear classifier by providing it more features; instead of just variable x, we could give additional inputs like x₂ and x₃.

Support Vector Machines

Computer scientists found an efficient way – called the ‘kernel trick’ - of mapping the data to higher-dimensional spaces (mathematical box below). The second idea to improve is to allow for a small number of errors to account for cases where the data is not linearly separable. This is accomplished by adding a regularization term which penalizes the optimizer for each misclassification. Putting these two ideas together, leads to Support Vector Machines.

Support Vector Machines

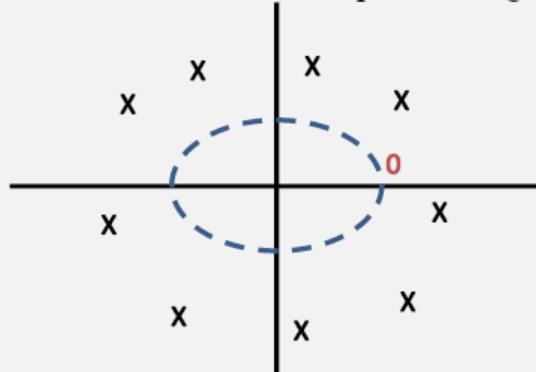
We describe the kernel trick in this section.

Given a set of input *attributes*, say $\underline{x} = [\underline{x}_1, \underline{x}_2, \underline{x}_3]$, one often finds it useful to map these to a set of *features*, say

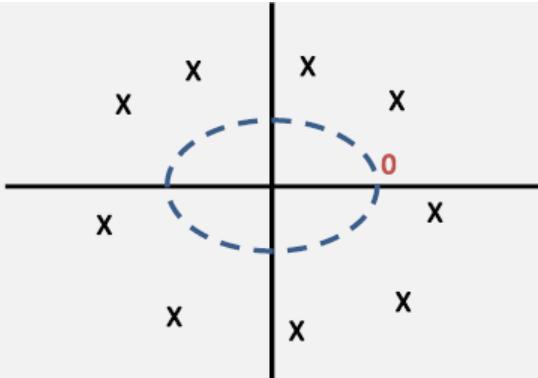
$$\phi(\underline{x}) = \begin{bmatrix} \underline{x}_1 \underline{x}_1 \\ \underline{x}_1 \underline{x}_2 \\ \vdots \\ \underline{x}_3 \underline{x}_2 \\ \underline{x}_3 \underline{x}_3 \end{bmatrix}.$$

Such a mapping into higher dimensions can often improve the performance of algorithms for function approximation. In many algorithms, one can reduce the functional dependence on input attributes to just inner products, i.e. the optimization function depends on the training example $\underline{x}^{(i)}$ only through the form of $\langle \underline{x}^{(i)}, \underline{c} \rangle$, where \underline{c} is another vector. In this case, the mapping to the higher-dimensional feature space creates the inner product $\langle \phi(\underline{x}^{(i)}), \phi(\underline{c}) \rangle$. The kernel trick refers to the observation that the inner product $k(\underline{x}^{(i)}, \underline{c}) = \langle \phi(\underline{x}^{(i)}), \phi(\underline{c}) \rangle$ can often be computed very efficiently.

As an example, consider the mapping ϕ as defined above. A naïve computation of $\langle \phi(\underline{x}^{(i)}), \phi(\underline{c}) \rangle$ would be an $O(n^2)$ operation. But noting that $\langle \phi(\underline{x}^{(i)}), \phi(\underline{c}) \rangle = (\underline{x}^{(i)T} \underline{c})^2 = k(\underline{x}^{(i)}, \underline{c})$, we can compute the value in just $O(n)$ time. This implies that an algorithm designed for use in a low-dimensional space of attributes can function even in a high-dimensional space of features, merely by replacing all inner products of the type $\langle \underline{x}^{(i)}, \underline{c} \rangle$ by $k(\underline{x}^{(i)}, \underline{c})$. As an example of such use, consider the points in figure.



Support Vector Machines



These points are not linearly separable in two dimensions, but mapping them to higher dimensions, say three, will make them so. In practice, this kernel trick is widely used to improve the performance of algorithms as diverse as the perceptron to support vector machines to Principal Component Analysis. The kernel trick enables linear learning algorithms to learn a non-linear boundary without specifying an explicit mapping. One need not stop at a finite-dimensional mapping. The

commonly used kernel function is the Gaussian Radial Basis Function, $k(\underline{x} - \underline{y}) = e^{-\frac{\|\underline{x} - \underline{y}\|^2}{2}}$, which corresponds to an infinite-dimensional mapping function ϕ and yet is computable in just $O(n)$ time.

To decide what forms are acceptable for the kernel function $k(\underline{x}, \underline{y})$, one defers to Mercer's theorem. Simply put, it states that for a function $k: \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ to be a valid kernel function, it is necessary and sufficient that for any set of vectors $\{\underline{x}^{(1)}, \underline{x}^{(2)}, \dots, \underline{x}^{(n)}\}$, the corresponding kernel matrix $K_{ij} = k(\underline{x}^{(i)}, \underline{x}^{(j)})$ is symmetric and positive definite.

Machine Learning In Finance

Supervised Learning

Classification

Support Vector Machines Application

SVM model

Nonlinear support vector machines can systematically identify stocks with high and low future returns

SSRN-id1930709

A Non-Linear SVM classifies a stock that has M features or input variables. In other words, the feature vector x has M components. Typically, the number of features varies from 7 to 51 depending on whether we use technical data, fundamental data, or both. An SVM classification function is

$$f(x) = \sum_{i=1}^N \alpha_i y_i K(x, x_i) - b$$

where

- x_i is all the vectors of the training set. Given a training history of tail sets, the SVM keeps them in memory because they will be used for prediction purposes.
- N is the number of training examples used to fit the SVM parameters, which varies from 10,000 to 100,000 examples depending on the history, the sector, and the quantile threshold on the distribution of volatility-adjusted returns.

SVM model

$$f(x) = \sum_{i=1}^N \alpha_i y_i K(x, x_i) - b$$

where

- α_i is a scalar, that is a real number, that takes values between 0 and C. The value of C is important because it indicates how much emphasis we give to fitting the model closely to the training data. If we make it high we might incur overfitting: even though all data points might be well classified in the training set, the model might lose its ability to generalize in out-of-sample tests.
- y_i identifies whether the feature vector x_i of the stock i belongs to the tail set +1 or not 1. We use +1 to label positive stock returns and 1 to label the opposite ones. The stocks that lie in the middle of distribution are disregarded for training.
- b is obtained by training the SVM, and is a scalar that shifts the output of the SVM by a constant.
- Finally, $K(x; x_i)$ is the kernel function. A kernel is a function that takes two vectors as inputs and produces a single scalar value which is positive. The kernel function has a series of interesting properties that are required to make the SVM work (Muller et al., 2001). For our investigation, we use the Gaussian kernel

SVM model

The SVM function is trained such that $f(x)$ is larger or equal than 1 if x belongs to class +1, and smaller or equal than -1 when it belongs to class -1. The α values and the b value are selected to match these requirements.

An important area of active research in machine learning is meta parameter search. These two meta parameters are chosen according to past performance on a training window, which emulates the way this system would be operated in real conditions today: evaluate the system performance in a given history for different pairs C, γ , choose the best set of parameters and use them to train the model for tomorrow's opening. One needs to replicate this procedure carefully to validate the model without incurring forward-looking bias. The software used to train the SVMs is based on Muezzinoglu et al. (2010), and is made available by Huerta (2010).

Technical indicators used

Feature	Type	Reference
α_{3m}	Momentum 3 months	Jegadeesh and Titman (2012) and Rouwenhorst (2002)
α_{1y}	Momentum 1 year	Jegadeesh and Titman (2012) and Rouwenhorst (2002)
ΔV_{3m}	Volume change 3 months	Lee and Swaminathan (2000) and Chordia and Swaminathan (2002)
ΔV_{1m}	Volume change 1 month	Lee and Swaminathan (2000) and Chordia and Swaminathan (2002)
$N_{high,low}$	Contrarian	Mizrach and Weerts (2009)
$maxR$	Contrarian	Bali et al. (2011)
RL	Resistance levels	

Fundamental Indicators

Feature	Formula or variable	Reference
Total Revenue	TR	
Gross Profit		
Operating Income		
Income before Tax		
Income after Tax		
Net Income before extraordinary items		
Net Income	NI	
Dividends	D	
Diluted normalized Earnings per Share		
Cash and Equivalents	CE	
Short Term Investments		
Accounts Receivable	AR	
Total Inventory	TI	
Total Current Assets	TCA	
Total Assets	TA	
Short Term Liabilities		
Total Current Liabilities	CL	
Total Long Term Debt		
Total Debt	TD	
Total Liabilities		
Total Equity	TE	
Total Shares	SO	
Depreciation		

Fundamental Indicators (2)

Feature	Formula or variable	Reference
Cash from operating activities		
Capital expenditures		
Cash from investing activities		
Cash from financing activities		
Net change in cash		
Snapshot accrual	SAC=TCA-CE-CL+TD	Sloan (1996)
Accrual based on balance sheet	SAC(quarter)-SAC(quarter-4)	Sloan (1996)
Accrual based on cash flow		Bradshaw et al. (2002)
Financial Health		Piotroski (2000)
Working capital	TCA-CL	
Quick ratio	(TCA-TI)/CL	
Dividend payout ratio	D/NI	
Book value	BV	
Book value -Total Debt	BV-TD	
Receivables to sales	AR/TR	
Debt to Assets	TD/TA	
Debt to Equity	TD/TE	
Cash to Assets	CE/TA	
Liabilities to Income	TL/NI	
Return on Equity	ROE = NI/TE	
Sales per shares	TR/SO	

Portfolios

We form portfolios of 10 equally weighted long and 10 equally weighted short positions. Each position is closed at the end of the last trading day in the following 91 days. Every 28 days we open an additional 20 positions. Therefore, on most days we have 60 total positions for each sector. The reason we take 20 new positions every 28 days, instead of opening 60 every 91 days, is to decorrelate temporally the natural fluctuations of the SVM classifier. Positions could be opened more frequently, to decrease the correlations between portfolios further, but that would require extra effort in maintaining the portfolios through time.

An important characteristic of our approach is that a fresh model is trained every time a new portfolio is to be formed, in order to adapt to changing environments.

The stocks with the highest values are chosen as long positions and the stocks with the lowest values are short sales. Note that, even though the system is trained for a particular quantile B, the SVM evaluation for time + d is run over the list of all tradable stocks obtained after running the filters. The net dollar value of the long positions is equal to the dollar value of the short ones. Since the trend of the longs is to increase in value and the trend of the shorts is to decrease in value, the portfolios become slightly unbalanced at the end of the rebalancing period. However, the correlation of the portfolio with the S&P 500 index is almost zero.

Thus, it is not critical to actively manage the portfolio to keep the long and short positions balanced.

Results Yearly Returns

B	Technicals			Fundamentals			Combined			
	r	σ	r/σ	r	σ	r/σ	r	σ	r/σ	D
5	15.14%	11.39%	1.32	13.05%	12.86%	1.01	20.51%	12.37%	1.65	-24.24%
10	15.44%	11.40%	1.35	12.99%	13.01%	0.99	18.74%	12.70%	1.47	-33.19%
15	15.84%	12.39%	1.27	14.23%	12.72%	1.11	20.53%	12.59%	1.63	-36.57%
20	13.12%	12.46%	1.05	15.13%	12.48%	1.21	21.92%	12.79%	1.71	-24.19%
25	14.51%	12.55%	1.15	14.78%	12.75%	1.15	23.15%	12.67%	1.83	-22.83%
30	14.30%	12.50%	1.14	14.45%	12.96%	1.11	21.22%	12.42%	1.70	-22.03%
35	17.33%	11.95%	1.44	11.30%	12.85%	0.87	21.55%	12.28%	1.75	-17.42%
40	19.48%	11.95%	1.62	13.19%	13.28%	0.99	21.85%	12.28%	1.77	-17.57%
45	20.22%	11.96%	1.69	11.93%	13.28%	0.89	20.26%	12.98%	1.60	-14.42%
50	16.53%	12.62%	1.30	9.69%	13.61%	0.71	20.00%	12.65%	1.58	-21.18%

Results Yearly Returns

B	All sectors except Utilities and Telecommunications					
	Jensen α	β	Sharpe ratio	volatility	max drawdown	<i>IR</i>
5	11.31%	0.0041	1.26	8.96%	-26.32%	0.63
10	13.06%	-0.0047	1.64	8.14%	-21.73%	0.73
15	13.96%	-0.0035	1.74	7.97%	-18.62%	0.73
20	13.86%	-0.0096	1.80	7.71%	-20.02%	0.69
25	14.86%	-0.0167	2.06	7.29%	-12.27%	0.75
30	13.69%	-0.0173	1.89	7.18%	-12.54%	0.64
35	14.06%	-0.0236	1.88	7.41%	-11.18%	0.67
40	12.70%	-0.0210	1.77	7.27%	-14.86%	0.59
45	13.76%	-0.0222	1.79	7.73%	-14.08%	0.62
50	12.76%	-0.0199	1.81	7.01%	-9.03%	0.56

Machine Learning In Finance

Supervised Learning

4.1.5 Ensembles

Ensemble Methods

- The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator.
- Two families of ensemble methods are usually distinguished:
- In averaging methods, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.
- Examples: Bagging methods, Forests of randomized trees, ...
- By contrast, in boosting methods, base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.

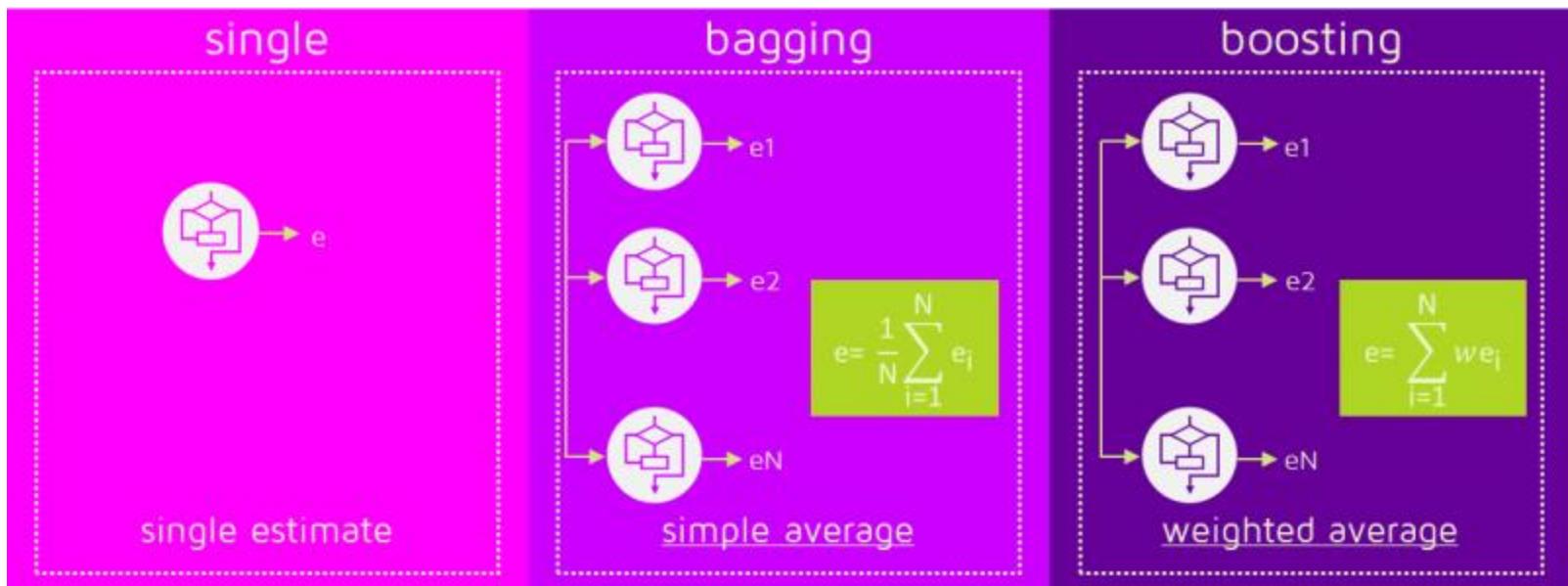
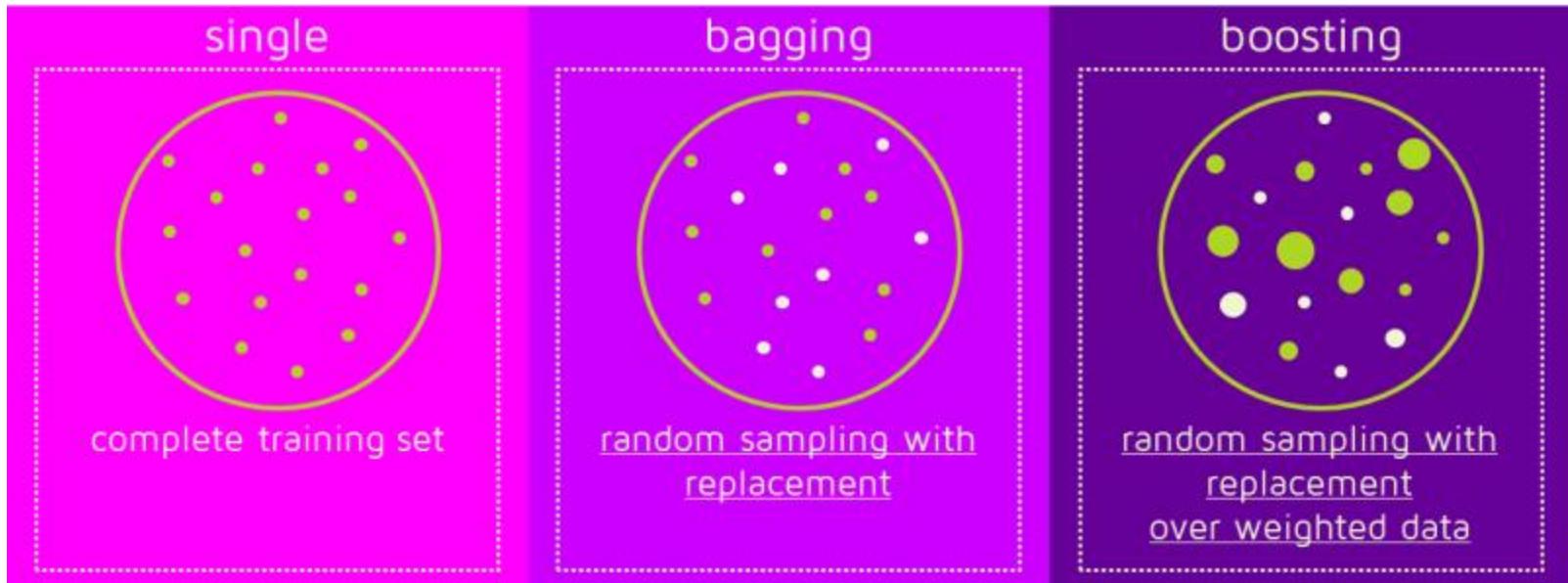
Ensemble Methods

- Ensemble is a Machine Learning concept in which the idea is to train multiple models using the same learning algorithm. The ensembles take part in a bigger group of methods, called multiclassifiers, where a set of hundreds or thousands of learners with a common objective are fused together to solve the problem.
- The second group of multiclassifiers contain the hybrid methods. They use a set of learners too, but they can be trained using different learning techniques. Stacking is the most well-known. If you want to learn more about Stacking, you can read my previous post, “Dream team combining classifiers”.

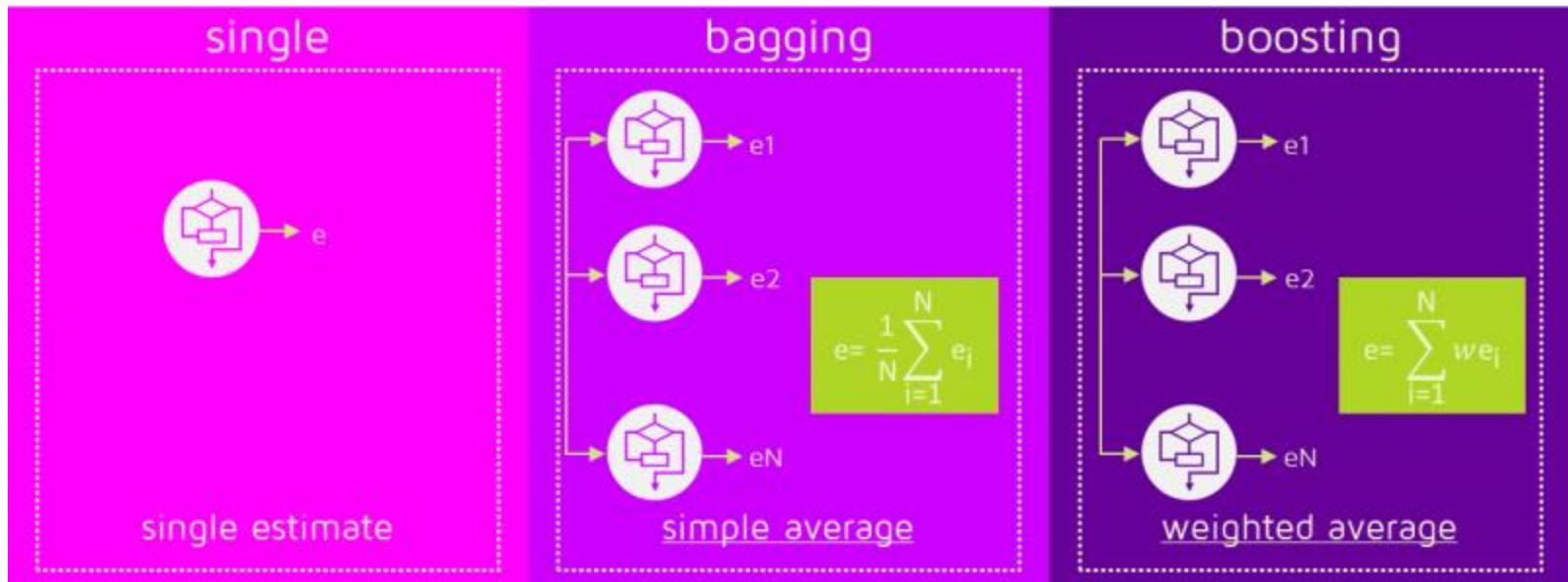
Ensemble Methods

- The main causes of error in learning are due to noise, bias and variance. Ensemble helps to minimize these factors. These methods are designed to improve the stability and the accuracy of Machine Learning algorithms. Combinations of multiple classifiers decrease variance, especially in the case of unstable classifiers, and may produce a more reliable classification than a single classifier.
- To use Bagging or Boosting you must select a base learner algorithm. For example, if we choose a classification tree, Bagging and Boosting would consist of a pool of trees as big as we want.

Ensemble Methods



Ensemble Methods



Ensemble Theory

An ensemble is itself a supervised learning algorithm, because it can be trained and then used to make predictions. The trained ensemble, therefore, represents a single hypothesis. This hypothesis, however, is not necessarily contained within the hypothesis space of the models from which it is built. Thus, ensembles can be shown to have more flexibility in the functions they can represent. This flexibility can, in theory, enable them to over-fit the training data more than a single model would, but in practice, some ensemble techniques (especially bagging) tend to reduce problems related to over-fitting of the training data.

Empirically, ensembles tend to yield better results when there is a significant diversity among the models.

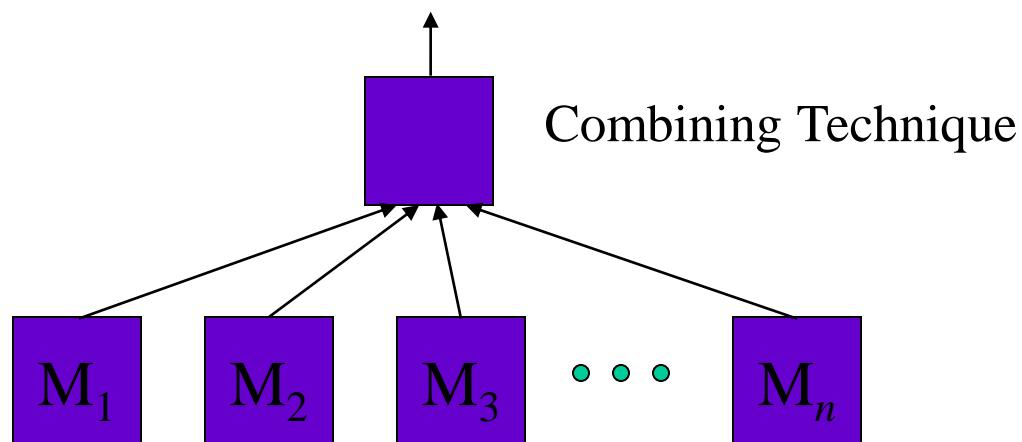
Many ensemble methods, therefore, seek to promote diversity among the models they combine.

Although perhaps non-intuitive, more random algorithms (like random decision trees) can be used to produce a stronger ensemble than very deliberate algorithms (like entropy-reducing decision trees).

Using a variety of strong learning algorithms, however, has been shown to be more effective than using techniques that attempt to dumb-down the models in order to promote diversity

Ensembles

- Multiple diverse models (Inductive Biases) are trained on the same problem and then their outputs are combined to come up with a final output
- The specific overfit of each learning model can be averaged out
- If models are diverse (uncorrelated errors) then even if the individual models are weak generalizers, the ensemble can be very accurate
- Many different Ensemble approaches
 - Stacking, Gating/Mixture of Experts, Bagging, Boosting, Wagging, Mimicking, Heuristic Weighted Voting, Combinations



Bias vs. Variance

- Learning models can have error based on two basic issues: Bias and Variance
 - "Bias" measures the basic capacity of a learning approach to fit the task
 - "Variance" measures the extent to which different hypothesis trained using a learning approach will vary based on initial conditions, training set, etc.
- MLPs trained with backprop have lower bias error because they can potentially fit most tasks well, but have relatively high variance error because each model might fall into odd nuances (overfit) based on training set choice, initial weights, and other parameters – Typical with the more complex models that we want
- Naïve Bayes has high bias error (doesn't fit that well), but has no variance error
- We would like low bias error and low variance error
- Ensembles using multiple trained (high variance/low bias) models can average out the variance, leaving just the bias
 - Less worry about overfit (stopping criteria, etc.) with the base models

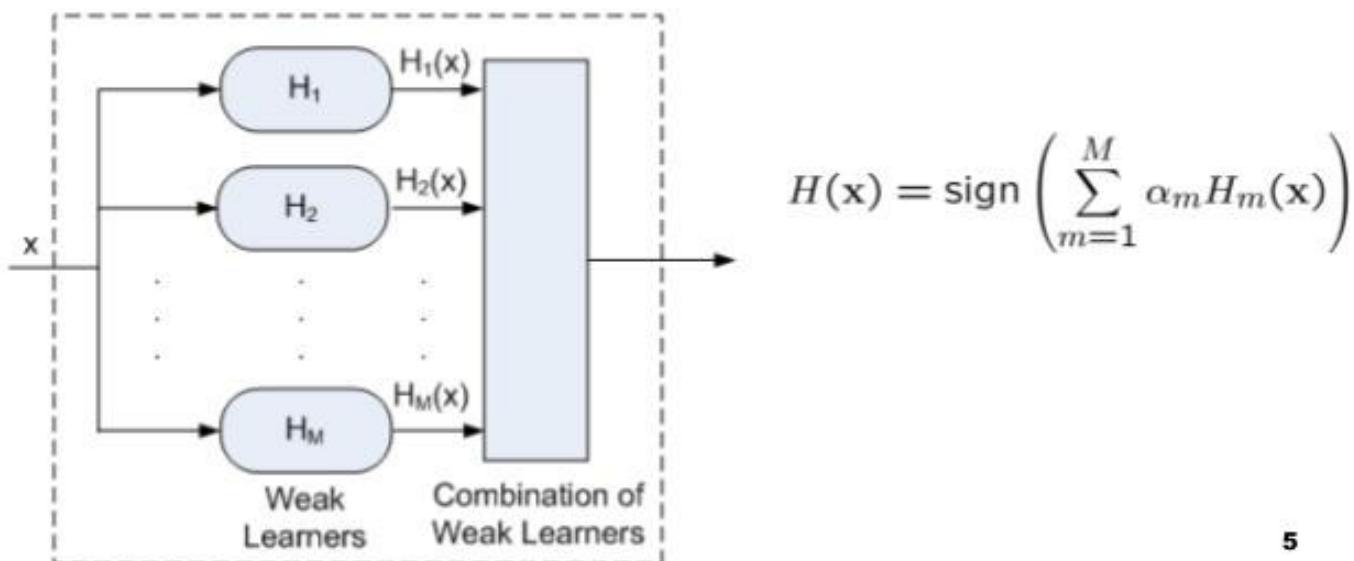
Combining Weak Learners

- Combining weak learners
 - Assume n induced models which are independent of each other with each having accuracy of 60% on a two class problem. If all n give the same class output then you can be confident it is correct with probability $1-(1-.6)^n$. For $n=10$, confidence would be 99.4%.
 - Normally not independent. If all n were the same model, then no advantage could be gained.
 - Also, unlikely that all n would give the same output, but if a majority did, then we can still get an overall accuracy better than the base accuracy of the models
 - If m models say class 1 and w models say class 2, then
 $P(\text{majority_class}) = 1 - \text{Binomial}(n, \min(m, w), .6)$

$$P(r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$$

Boosting

- Boosting builds an ensemble of classifiers
- Combine simple classifiers to build a robust classifier
- Each classifier H_m has an associated contribution α_m



Boosting

- Boosting by resampling - Each TS is chosen randomly with distribution D_t with replacement from the original data set. D_1 has all instance equally likely to be chosen. Typically each TS is the same size as the original data set.
 - Induce first model with TS_1 drawn using D_1 . Create D_{t+1} so that instances which are mis-classified by the most recent model on TS_1 have a higher probability of being chosen for future training sets.
 - Keep training new models until stopping criteria met
 - M models induced
 - Overall Accuracy levels out or most recent model has accuracy less than .5 on its TS
 - Etc.
- All models vote but each model's vote is scaled by its accuracy on the training set it was trained on
- Boosting is more aggressive than bagging on accuracy but in some cases can overfit and do worse – can theoretically converge to training set
 - On average better than bagging, but worse for some tasks
 - In rare cases can be worse than the non-ensemble approach
- Many variations

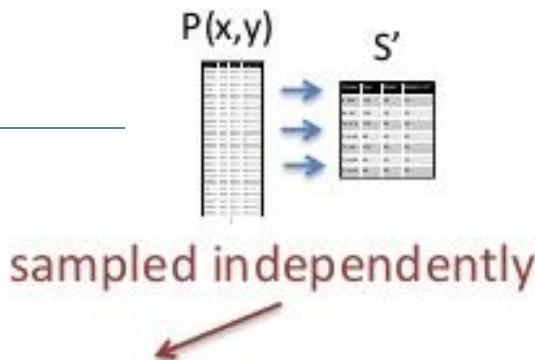
Bagging

Bagging (Bootstrap aggregating) was proposed by Leo Breiman in 1994 to improve the classification by combining classifications of randomly generated training sets. See Breiman, 1994. Technical Report No. 421.

- Given a standard training set D of size n , bagging generates m new training sets D_i , each of size n' , by sampling from D uniformly and with replacement. By sampling with replacement, some observations may be repeated in each D_i .
- If $n'=n$, then for large n the set D_i is expected to have the fraction $(1 - 1/e)$ ($\approx 63.2\%$) of the unique examples of D , the rest being duplicates.
- This kind of sample is known as a bootstrap sample. The m models are fitted using the above m bootstrap samples and combined by averaging the output (for regression) or voting (for classification).

Bagging

- **Goal:** reduce variance
- **Ideal setting:** many training sets S'
 - Train model using each S'
 - Average predictions



Variance reduces linearly
Bias unchanged

$$E_S[(h(x|S) - y)^2] = \underbrace{E_S[(Z - \bar{z})^2]}_{\text{Expected Error}} + \underbrace{\bar{z}^2}_{\text{Variance}} + \underbrace{y^2}_{\text{Bias}}$$

$$\begin{aligned} Z &= h(x|S) - y \\ \bar{z} &= E_S[Z] \end{aligned}$$

“Bagging Predictors” [Leo Breiman, 1994]
<http://statistics.berkeley.edu/sites/default/files/tech-reports/421.pdf>

Bagging

- Bootstrap aggregating (Bagging)
- Great way to improve overall accuracy by decreasing variance
- Often used with the same learning algorithm and thus best for those which tend to give more diverse hypotheses based on initial conditions
- Induce m learners starting with same initial parameters with each training set chosen uniformly at random with replacement from the original data set, training sets might be $2/3^{\text{rds}}$ of the data set – still need to save some separate data for testing
- All m hypotheses have an equal vote for classifying novel instances
- Consistent significant empirical improvement
- Does not overfit (whereas boosting may), but may be more conservative overall on accuracy improvements
- Could use other schemes to improve the diversity between learners
 - Different initial parameters, sampling approaches, etc.
 - Different learning algorithms
 - The more diversity the better - (yet often used with the same learning algorithm and just different training sets)

Ensemble Creation Approaches

- A good goal is to get less correlated errors between models
- Injecting randomness – initial weights, different learning parameters, etc.
- Different Training sets – Bagging, Boosting, different features, etc.
- Forcing differences – different objective functions, auxiliary tasks
- Different machine learning models
 - Obvious, but surprisingly it is less used

Ensemble Combining Approaches

- Unweighted Voting (e.g. Bagging)
- Weighted voting – based on accuracy (e.g. Boosting), Expertise, etc.
- Stacking - Learn the combination function
 - Higher order possibilities
 - Which algorithm should be used for the stacker
 - Stacking the stack, etc.
- Gating function/Mixture of Experts – The gating function uses the input features to decide which combination (weights) of expert voting to use
- Heuristic Weighted Voting

Ensemble Summary

- Efficiency
 - Wagging (Weight Averaging) - Multi-layer?
 - Mimicking - Oracle Learning
- Other Models - Cascading, Arbitration, Delegation, PDDAGS (Parallel Decision DAGs), etc.
- Almost always gain accuracy improvements by decreasing variance
- Still lots of potential work to be done regarding the best ways to create and combine models for ensembles
- Which algorithms are most different and thus most appropriate to ensemble:
COD (Classifier Output Distance) research

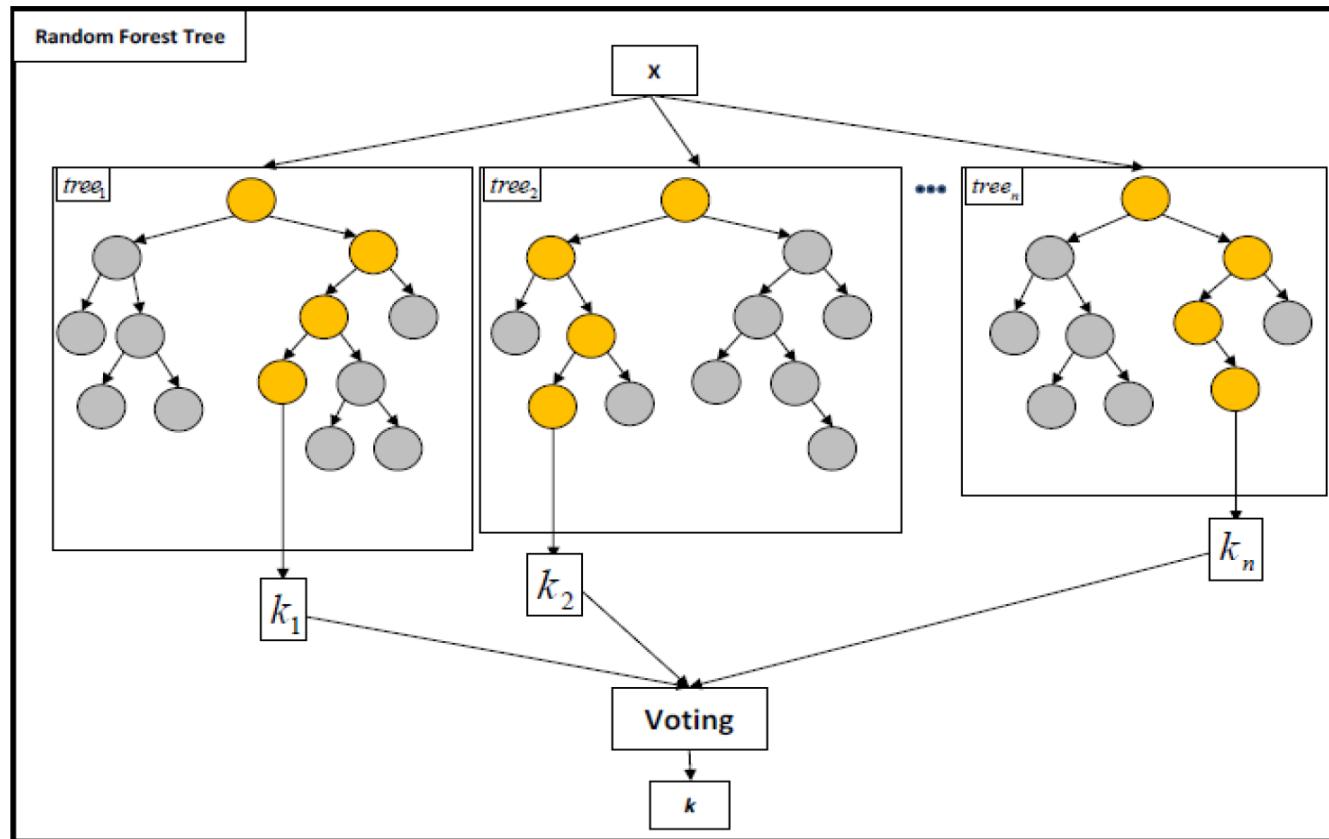
Machine Learning In Finance

Supervised Learning

Random Forest

Random Forest

A natural extension of the CART model is the random forest algorithm. This approach incorporates the idea of “bagging” into the decision tree, i.e., randomly selecting subsets of data and building a tree



Random Forest

In random forests (see `RandomForestClassifier` and `RandomForestRegressor` classes), each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features.

Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.

In contrast to the original publication [B2001], the scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class.

Random Forest

In extremely randomized trees (see `ExtraTreesClassifier` and `ExtraTreesRegressor` classes), randomness goes one step further in the way splits are computed. As in random forests, a random subset of candidate features is used, but instead of looking for the most discriminative thresholds, thresholds are drawn at random for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule. This usually allows to reduce the variance of the model a bit more, at the expense of a slightly greater increase in bias.

Random Forest

```
>>> from sklearn.model_selection import cross_val_score
>>> from sklearn.datasets import make_blobs
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.ensemble import ExtraTreesClassifier
>>> from sklearn.tree import DecisionTreeClassifier

>>> X, y = make_blobs(n_samples=10000, n_features=10, centers=100,
...                     random_state=0)

>>> clf = DecisionTreeClassifier(max_depth=None, min_samples_split=2,
...                                random_state=0)
>>> scores = cross_val_score(clf, X, y, cv=5)
>>> scores.mean()
0.98...

>>> clf = RandomForestClassifier(n_estimators=10, max_depth=None,
...                               min_samples_split=2, random_state=0)
>>> scores = cross_val_score(clf, X, y, cv=5)
>>> scores.mean()
0.999...
```

```
>>> clf = ExtraTreesClassifier(n_estimators=10, max_depth=None,
...                             min_samples_split=2, random_state=0)
>>> scores = cross_val_score(clf, X, y, cv=5)
>>> scores.mean() > 0.999
True
```

Random Forest

The main parameters to adjust when using these methods is `n_estimators` and `max_features`. The former is the number of trees in the forest. The larger the better, but also the longer it will take to compute. In addition, note that results will stop getting significantly better beyond a critical number of trees. The latter is the size of the random subsets of features to consider when splitting a node. The lower the greater the reduction of variance, but also the greater the increase in bias.

Empirical good default values are `max_features=n_features` for regression problems, and `max_features=sqrt(n_features)` for classification tasks (where `n_features` is the number of features in the data). Good results are often achieved when setting `max_depth=None` in combination with `min_samples_split=2` (i.e., when fully developing the trees). Bear in mind though that these values are usually not optimal, and might result in models that consume a lot of RAM. The best parameter values should always be cross-validated. In addition, note that in random forests, bootstrap samples are used by default (`bootstrap=True`) while the default strategy for extra-trees is to use the whole dataset (`bootstrap=False`). When using bootstrap sampling the generalization accuracy can be estimated on the left out or out-of-bag samples. This can be enabled by setting `oob_score=True`.

Random Forest - Algorithm

A natural extension of the CART model is the random forest algorithm. This approach incorporates the idea of “bagging” into the decision tree, i.e., randomly selecting subsets of data and building a tree

Algorithm 1 Random Forest Classifier

```
1: procedure RANDOMFORESTCLASSIFIER( $D$ ) ▷  $D$  is the labeled training data
2:    $forest = \text{new Array}()$ 
3:   for do  $i = 0$  to  $B$ 
4:      $D_i = \text{Bagging}(D)$                                 ▷ Bootstrap Aggregation
5:      $T_i = \text{new DecisionTree}()$ 
6:      $features_i = \text{RandomFeatureSelection}(D_i)$ 
7:      $T_i.\text{train}(D_i, features_i)$ 
8:      $forest.add(T_i)$ 
9:   end for
10:  return  $forest$ 
11: end procedure
```

Ensemble Methods

Some of the Boosting techniques include an extra-condition to keep or discard a single learner. For example, in AdaBoost, the most renowned, an error less than 50% is required to maintain the model; otherwise, the iteration is repeated until achieving a learner better than a random guess.

The previous image shows the general process of a Boosting method, but several alternatives exist with different ways to determine the weights to use in the next training step and in the classification stage.

Some of the most important ones are

AdaBoost, LPBoost, XGBoost, GradientBoost, BrownBoost.

Machine Learning In Finance

Supervised Learning

Adaboost

Adaboost

The module `sklearn.ensemble` includes the popular boosting algorithm AdaBoost, introduced in 1995 by Freund and Schapire [FS1995].

The core principle of AdaBoost is to fit a sequence of weak learners (i.e., models that are only slightly better than random guessing, such as small decision trees) on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote (or sum) to produce the final prediction. The data modifications at each so-called boosting iteration consist of applying weights w_1, w_2, \dots, w_N to each of the training samples. Initially, those weights are all set to $w_i = 1/N$, so that the first step simply trains a weak learner on the original data. For each successive iteration, the sample weights are individually modified and the learning algorithm is reapplied to the reweighted data.

At a given step, those training examples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased for those that were predicted correctly. As iterations proceed, examples that are difficult to predict receive ever-increasing influence. Each subsequent weak learner is thereby forced to concentrate on the examples that are missed by the previous ones in the sequence [HTF].

Adaboost

The following example shows how to fit an AdaBoost classifier with 100 weak learners:

```
>>> from sklearn.model_selection import cross_val_score  
>>> from sklearn.datasets import load_iris  
>>> from sklearn.ensemble import AdaBoostClassifier
```

```
>>> iris = load_iris()  
>>> clf = AdaBoostClassifier(n_estimators=100)  
>>> scores = cross_val_score(clf, iris.data, iris.target, cv=5)  
>>> scores.mean()  
0.9...
```

The number of weak learners is controlled by the parameter `n_estimators`. The `learning_rate` parameter controls the contribution of the weak learners in the final combination. By default, weak learners are decision stumps. Different weak learners can be specified through the `base_estimator` parameter. The main parameters to tune to obtain good results are `n_estimators` and the complexity of the base estimators (e.g., its depth `max_depth` or minimum required number of samples to consider a split `min_samples_split`).

Adaboost

Given weak learners, how do strengthen their combined predictive ability? **AdaBoost** [4]

- ▶ Iterative construction of a weighted ensemble of weak learners
- ▶ At each iteration overweight misclassified cases
- ▶ Stop until total misclassification error is not further reduced

The concept is illustrated below, where four iterations are used to separate the shapes.

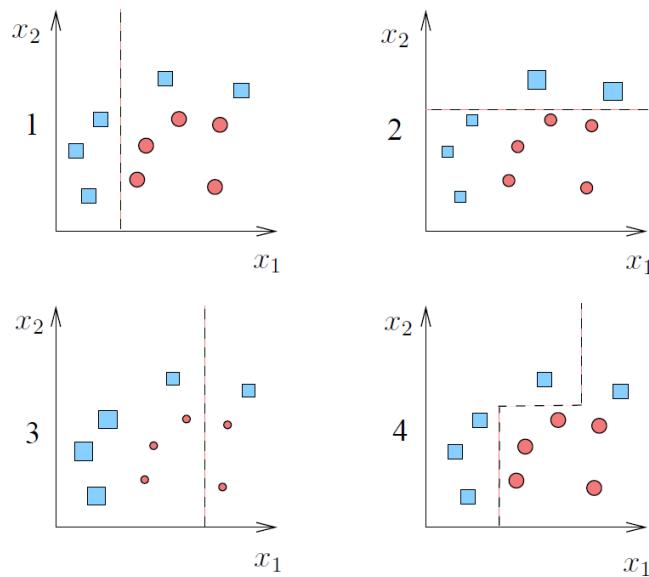


Figure: AdaBoost iterations. The final classification at stage 4 comes from majority voting.

Adaboost

Setup: The column vector \mathbf{y} contains N observed class labels, i.e. $y_i = \pm 1$.

The $N \times M$ matrix \mathbf{X} contains M predictor variables.

The size- M (row) vector of predictors for the i -th observation is denoted by x_i .

There are L weak learners $h_l : \mathbf{X} \rightarrow \hat{\mathbf{y}}$, $l = 1 \dots L$, $\hat{y}_i = \pm 1$.

The weight vector \mathbf{w} contains N elements, $w_i = 1/N$.

For t in $1 \dots T$:

- ▶ Choose h_t , i.e. find h_l , $l = 1 \dots L$ that minimizes the weighted misclassification error:

$$\epsilon_t = \sum_{i=1}^N w_{i,t} I(h_t(x_i) \neq y_i). \quad (18)$$

- ▶ Compute the coefficient:

$$\alpha_t = \frac{1}{2} \log \left(\frac{1 - \epsilon_t}{\epsilon_t} \right). \quad (19)$$

- ▶ Update the weights for the next iteration:

$$w_{i,t+1} = w_{i,t} e^{-\alpha_t y_i h_t(x_i)}. \quad (20)$$

- ▶ Renormalize the weights so that they sum up to one:

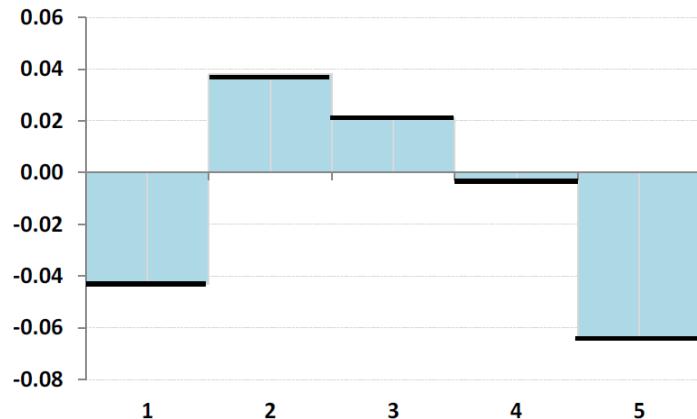
$$w_{i,t+1} \leftarrow \frac{w_{i,t+1}}{\sum_{i=1}^N w_{i,t+1}}. \quad (21)$$

The final prediction is

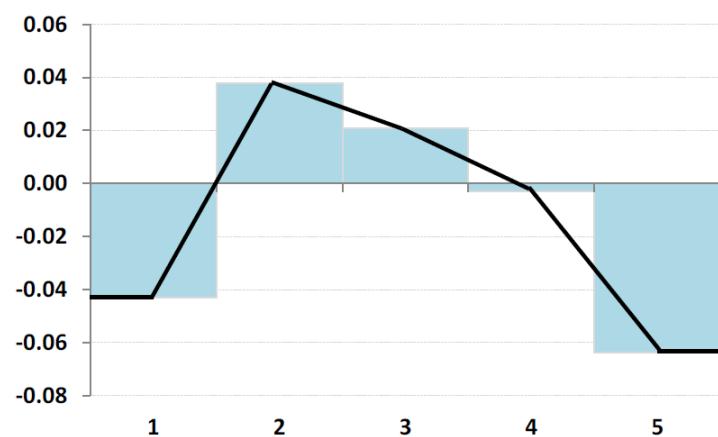
$$\hat{\mathbf{y}} = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(\mathbf{X}) \right). \quad (22)$$

Adaboost

The accruals factor – generic AdaBoost



The accruals factor – linearized AdaBoost



Modified AdaBoost Algorithm

1. Normalize all the factor value to a real value (0,1] according to the rank:

$$\text{Normalized_factor} = \text{factor_rank}/\text{total_number_of_factors_in_the_month}$$

2. Given stock performance set $S = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ and the factor pool F , where $y_i \in \{-1, 1\}$, N is the number of stocks in the training sample.

3. Initially equally weighted all the stocks $w_1(x) = 1/N$

4. For $l = 1, \dots, L$

- 1) For each factor f^* from the factor pool we build a weak classifier h^* .

- a) Divide the training data into Q quantiles, X_1, X_2, \dots, X_Q

- b) For each quantile j we calculate the total weight of outperformers and underperformers

$$W_{\pm}^j = \sum_{y_i = \pm 1} w(x_i) \cdot \max(0, (1 - \text{dist}(f(x_i), j)))$$

where $\text{dist}(f(x_i), j)$ is the normalized distance between the centre of the quantile j and the factor score $f(x_i)$

- c) Calculate the discriminative objective function:

$$Z_l^k = \sum_{j=1}^n \sqrt{W_+^j W_-^j}$$

- d) Get a weak classifier:

$$h^k(x) = \frac{1}{2} \sum_{j=1}^n \ln\left(\frac{W_+^j + \varepsilon}{W_-^j + \varepsilon}\right) \cdot \max(0, (1 - \text{dist}(f(x), j)))$$

- 2) Find the best weak classifier according to the discriminative objective function:

$$h_l(x) = h^k(x) \text{ where } k = \arg \min_{f^* \in F} \{Z_l^k\}$$

- 3) Update the weights of each stock:

$$w_{(l+1)}(x_i) = w_l(x_i) \exp(-y_l h_l(x_i))$$

- 4) Normalize the weight $w_{l+1}(x)$ so that they add up to 1.

5. The final strong classifier: $H(x) = \sum_{l=1}^L h_l(x)$

Machine Learning In Finance

Supervised Learning

XGboost

XGBoost

In contrast to bagging techniques like Random Forest, in which trees are grown to their maximum extent, boosting makes use of trees with fewer splits. Such small trees, which are not very deep, are highly interpretable.

Parameters like the number of trees or iterations, the rate at which the gradient boosting learns, and the depth of the tree, could be optimally selected through validation techniques like k-fold cross validation. Having a large number of trees might lead to overfitting. So, it is necessary to carefully choose the stopping criteria for boosting.

$$\begin{aligned}\hat{y}_i^{(0)} &= 0 \\ \hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\ &\quad \dots \\ \hat{y}_i^{(1)} &= \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)\end{aligned}$$

XGBoost

Boosting consists of three simple steps:

- An initial model F_0 is defined to predict the target variable y . This model will be associated with a residual $(y - F_0)$
- A new model h_1 is fit to the residuals from the previous step
- Now, F_0 and h_1 are combined to give F_1 , the boosted version of F_0 . The mean squared error from F_1 will be lower than that from F_0 :

$$F_1(x) \leftarrow F_0(x) + h_1(x)$$

To improve the performance of F_1 , we could model after the residuals of F_1 and create a new model F_2 :

$$F_m(x) \leftarrow F_{m-1}(x) + h_m(x)$$

XGBoost

Using gradient descent for optimizing the loss function

Gradient descent helps us minimize any differentiable function. Earlier, the regression tree for $h_m(x)$ predicted the mean residual at each terminal node of the tree. In gradient boosting, the average gradient component would be computed.

For each node, there is a factor γ with which $h_m(x)$ is multiplied. This accounts for the difference in impact of each branch of the split. Gradient boosting helps in predicting the optimal gradient for the additive model, unlike classical gradient descent techniques which reduce error in the output at each iteration.

.

XGBoost – Steps

$F_0(x)$ – with which we initialize the boosting algorithm – is to be defined:

$$F_0(x) = \operatorname{argmin}_\gamma \sum_{i=1}^n L(y_i, \gamma)$$

The gradient of the loss function is computed iteratively:

$$r_{im} = -\alpha \left[\frac{\partial(L(y_i, F(x_i)))}{\partial F(x_i)} \right] - \alpha \left[\frac{\partial(L(y_i, F(x_i)))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)},$$

Where α is the learning rate.

Each $h_m(x)$ is fit on the gradient obtained at each step

The multiplicative factor γ_m for each terminal node is derived and the boosted model $F_m(x)$ is defined:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

XG Boost Mathematics

Mathematically, we can write our model in the form

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), f_k \in \mathcal{F}$$

Where K is the number of trees, f is a function in the functional space \mathcal{F} and is the set of all possible CARTs. The objective function to be optimized is given by

$$obj(\theta) = \sum_i^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

$l(y_i, \hat{y}_i)$ is training loss $\sum_{k=1}^K \Omega(f_k)$ regularization.

So random forests and boosted trees are really the same models; the difference arises from how we train them.

This means that, if you write a predictive service for tree ensembles, you only need to write one and it should work for both random forests and gradient boosted trees.

XG Boost Mathematics

When you talk about (decision) trees, it is usually heuristics :

- Split by information gain
- Prune the tree
- Maximum depth
- Smooth the leaf values

Most heuristics maps well to objectives, taking the formal (objective) view let us know what we are learning

- Information gain -> training loss
- Pruning -> regularization defined by #nodes
- Max depth -> constraint on the function space
- Smoothing leaf values -> L2 regularization on leaf weights

XG Boost Mathematics – Additive Training

The first question we want to ask: what are the parameters of trees?

You can find that what we need to learn are those functions , each containing the structure of the tree and the leaf scores. Learning tree structure is much harder than traditional optimization problem where you can simply take the gradient. It is intractable to learn all the trees at once. Instead, we use an additive strategy: fix what we have learned, and add one new tree at a time. We write the prediction value at step t as $\hat{y}_i^{(t)}$.

$$\begin{aligned}\hat{y}_i^{(0)} &= 0 \\ \hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\ &\quad .. \\ \hat{y}_i^{(1)} &= \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)\end{aligned}$$

XG Boost Mathematics – Additive Training

It remains to ask: which tree do we want at each step? A natural thing is to add the one that optimizes our objective.

$$\begin{aligned} obj^t &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^t \Omega(f_k) = \\ &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \text{constant} \end{aligned}$$

If we used mean squared error (MSE) as our loss function, the objective becomes

$$\begin{aligned} obj^t &= \sum_{i=1}^n (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \sum_{k=1}^t \Omega(f_k) = \\ &= \sum_{i=1}^n [2(\hat{y}_i^{(t-1)} - y_i)f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + \text{constant} \end{aligned}$$

XG Boost Mathematics – Additive Training

The form of MSE is friendly, with a first order term (usually called the residual) and a quadratic term. For other losses of interest (for example, logistic loss), it is not so easy to get such a nice form. So in the general case, we take the Taylor expansion of the loss function up to the second order:

$$obj^t = \sum_{i=1}^n \left[l(y_i, \hat{y}_i^{t-1}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + constant$$

Where g_i and h_i are defined as

$$g_i = \partial_{\hat{y}_i^{t-1}} l(y_i, \hat{y}_i^{t-1})$$

$$h_i = \partial_{\hat{y}_i^{t-1}}^2 l(y_i, \hat{y}_i^{t-1})$$

XG Boost Mathematics – Additive Training

After we remove all the constants, the specific objective at step t becomes

$$\sum_{i=1}^n \left[q_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

This becomes our optimization goal for the new tree. One important advantage of this definition is that the value of the objective function only depends on g_i and h_i . This is how XGBoost supports custom loss functions. We can optimize every loss function, including logistic regression and pairwise ranking, using exactly the same solver that takes g_i and h_i as input!

XG Boost Mathematics – Additive Training

Here is the magical part of the derivation. After re-formulating the tree model, we can write the objective value with the t -th tree as:

$$\begin{aligned} obj^t &\approx \sum_{i=1}^n \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 \\ &= \sum_{j=1}^T \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

where I_j is the set of indices of data points assigned to the j -th leaf. Notice that in the second line we have changed the index of the summation because all the data points on the same leaf get the same score. We could further compress the expression by

Defining $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$

$$obj^t = \sum_{j=1}^T \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T$$

XG Boost Mathematics – Additive Training

In this equation, w_j are independent with respect to each other, the form $G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2$ is quadratic and the best w_j for a given structure $q(x)$ and the best objective reduction we can get is:

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

$$obj^* = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

Basically, for a given tree structure, we push the statistics and to the leaves they belong to, sum the statistics g_i h_i together, and use the formula to calculate how good the tree is. This score is like the impurity measure in a decision tree, except that it also takes the model complexity into account.

XG Boost Mathematics – Additive Training

Now that we have a way to measure how good a tree is, ideally we would enumerate all possible trees and pick the best one. In practice this is intractable, so we will try to optimize one level of the tree at a time. Specifically we try to split a leaf into two leaves, and the score it gains is

$$Gain = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

This formula can be decomposed as 1) the score on the new left leaf 2) the score on the new right leaf 3) The score on the original leaf 4) regularization on the additional leaf. We can see an important fact here: if the gain is smaller than γ , we would do better not to add that branch. This is exactly the pruning techniques in tree based models! By using the principles of supervised learning, we can naturally come up with the reason these techniques work

Stock Picking – Features – Tony Guida Paper XGBoost

Features 1 >>> ~ 200

	DataDate	dateReturnPerf	SecID	Fact2	Fact3	Fact4	Fact7	Fact8	Fact9	Fact10	Fact11	Fact12	Fact13	Fact14	Fact16	Fact17	Fact18	Fact20	Fact21	Fact22	Fact23	Fact25	Fact26	Fact27
1	2010-12-31	2011-12-31	1195515967	19	70	94	94	13	99	15	34	55	65	42	32	28	21	27	31	49	38	53	53	76
2	2014-08-31	2015-08-31	570191681	44	NULL	83	48	11	7	19	72	72	28	33	32	31	21	30	22	31	51	13	14	49
3	2012-01-31	2013-01-31	290849864	55	66	98	15	3	100	7	2	2	58	43	14	38	25	19	22	11	11	10	11	65
4	2012-02-29	2013-02-28	324622763	69	69	99	27	4	100	7	2	1	65	42	14	39	16	18	21	6	6	13	15	62
5	2012-08-31	2013-08-31	1528063821	73	70	99	61	4	99	14	2	2	47	41	19	38	16	25	26	43	45	83	84	50
6	2012-03-31	2013-03-31	1850474528	61	67	99	27	4	100	11	2	1	60	18	18	18	17	20	20	20	20	20	20	20
7	2012-11-30	2013-11-30	2021474168	75	74	99	60	4	NULL	13	2	10	47	26	26	26	26	26	26	26	26	26	26	26
8	2017-01-31	2018-01-31	408403206	42	31	61	68	10	2	26	94	92	14	18	3	10	47	18	18	18	18	18	18	18
9	2013-01-31	2014-01-31	1593812596	65	73	99	53	4	NULL	18	3	10	47	NULL	77	77	77	77	77	77	77	77	77	77
10	2016-05-31	2017-05-31	352206094	NULL	NULL	NULL	NULL	NULL	74	NULL	77	77	33	10	10	10	10	10	10	10	10	10	10	10
11	2009-03-31	2010-03-31	1251799406	55	65	49	30	10	100	11	3	1	83	10	10	10	10	10	10	10	10	10	10	10
12	2007-02-28	2008-02-29	1092089186	36	69	21	97	8	90	10	74	63	33	27	89	83	13	13	13	13	13	13	13	13
13	2016-09-30	2017-09-30	1171513832	30	32	60	67	10	100	30	3	1	41	10	10	10	10	10	10	10	10	10	10	10
14	2003-07-31	2004-07-31	1668572152	48	30	70	35	8	91	30	3	1	41	10	10	10	10	10	10	10	10	10	10	10
15	2010-08-31	2011-08-31	1317881264	20	4	87	96	15	99	10	29	56	62	10	10	10	10	10	10	10	10	10	10	10
16	2008-02-29	2009-02-28	31085621	46	81	19	24	7	63	5	77	62	51	10	10	10	10	10	10	10	10	10	10	10
17	2009-04-30	2010-04-30	512957258	52	62	50	32	10	100	9	3	1	84	10	10	10	10	10	10	10	10	10	10	10
18	2012-04-30	2013-04-30	2143460900	67	68	99	27	4	100	10	2	1	60	10	10	10	10	10	10	10	10	10	10	10
19	2016-10-31	2017-10-31	1415589206	20	NULL	NULL	64	11	58	35	79	73	15	12	35	55	65	12	12	12	12	12	12	12
20	2011-01-31	2012-01-31	486144046	21	70	95	95	13	99	19	1	1	71	19	1	1	71	19	1	1	71	19	1	71
21	2013-03-31	2014-03-31	156902714	54	64	99	67	1	100	19	5	35	36	12	29	56	62	12	29	56	62	12	29	56
22	2009-08-31	2010-08-31	290508352	29	3	53	94	18	100	19	1	1	71	19	1	1	71	19	1	1	71	19	1	71
23	2010-09-30	2011-09-30	1527687499	20	4	87	96	15	99	12	29	56	62	12	29	56	62	12	29	56	62	12	29	56
24	2016-06-30	2017-06-30	2006847672	NULL	NULL	NULL	NULL	NULL	80	NULL	75	74	23	NULL	41	56	37	30	26	74	64	87	82	50
25	2007-12-31	2008-12-31	156190601	47	84	12	83	7	100	8	90	77	53	72	57	60	62	64	72	31	30	18	15	29
26	2014-07-31	2015-07-31	1573508350	48	NULL	89	84	12	6	19	71	71	31	32	31	32	25	31	23	35	61	51	61	53
27	2007-06-30	2008-06-30	1049253878	38	70	26	97	6	96	10	81	70	32	18	67	23	79	75	75	34	42	71	71	30
28	2007-03-31	2008-03-31	400280762	36	70	21	97	8	90	11	74	62	32	17	68	53	81	81	74	34	51	71	77	30
29	2016-08-31	2017-08-31	1510213818	32	30	60	70	10	100	24	89	83	14	23	44	58	29	36	28	83	56	94	88	51
30	2011-02-28	2012-02-29	231369713	19	65	95	58	15	100	11	23	45	63	48	31	34	26	24	31	57	59	60	62	78
31	2008-04-30	2009-04-30	474472098	45	81	20	24	6	65	6	78	62	51	77	54	34	53	59	67	14	10	38	26	26

Some features examples

- Fundamental trailing Estimates
- Price based
- Volume based
- Composites
- Regime variables

Instances >>> ~ 400 000

Implementing XGBoost - Parameters

learning_rate: step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features. and eta actually shrinks the feature weights to make the boosting process more conservative.

min_split_loss: minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm will be.

max_depth: maximum depth of a tree, increase this value will make the model more complex / likely to be overfitting. 0 indicates no limit, limit is required for depth-wise grow policy.

scale_pos_weight: Control the balance of positive and negative weights, useful for unbalanced classes.

eXtremeGradient Boosting: introduction to the model

General objective of tree ensemble for K trees

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), \quad f_k \in \mathcal{F}$$

$$Obj = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

Training on loss

Complexity of the trees

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$

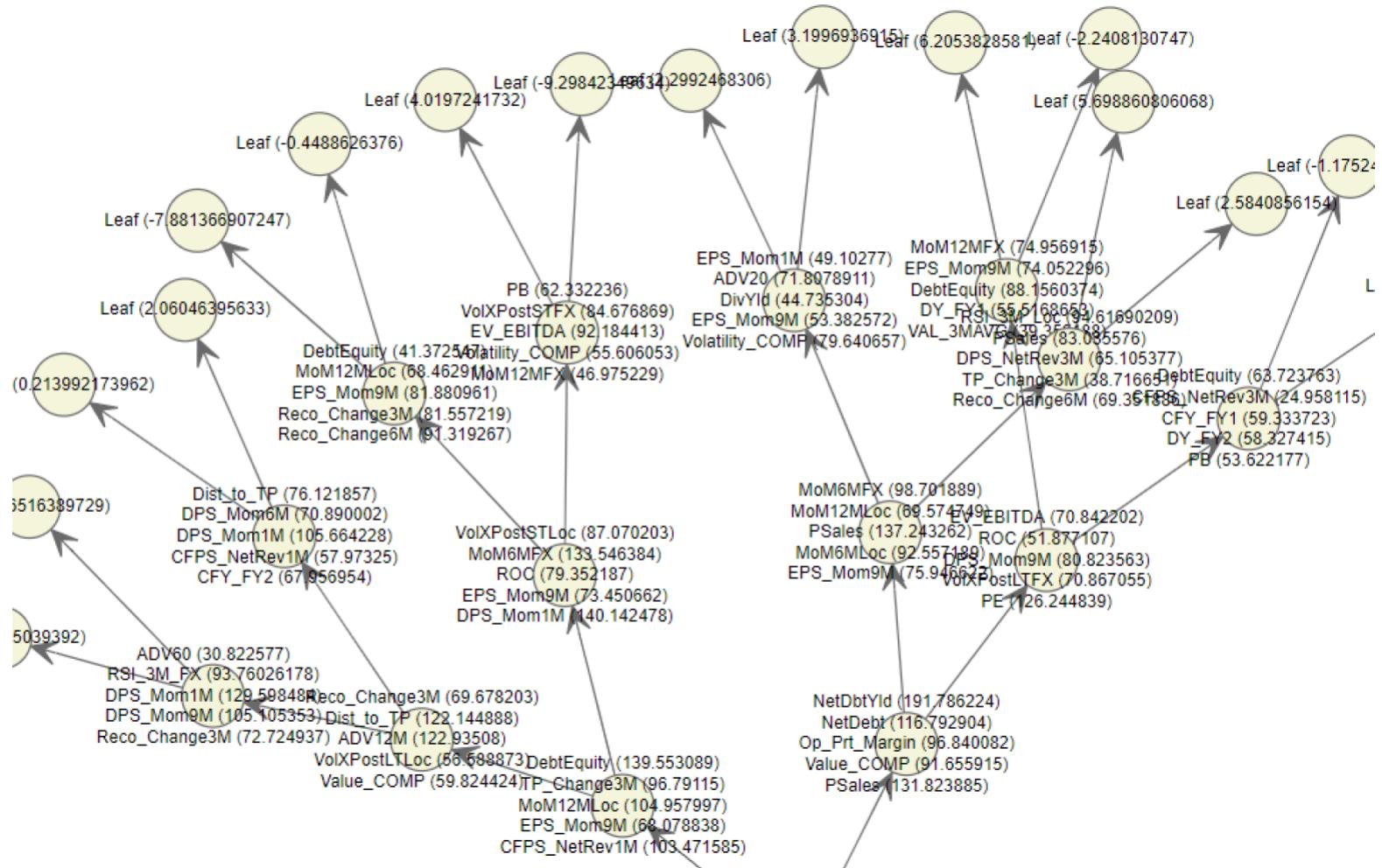
...

$$\hat{y}_i^{(t)} = \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

<http://xgboost.readthedocs.io/en/latest/model.html#>

Additive training

Boosted Tree Example



Type I Type II Examples

	OUTPERFORMED	UNDERPERFORMED
OUTPERFORM	<p>True Positive: Stock WAS classified as outperforming sector and DID outperformed</p>	<p>False negative: Stock was NOT classified as outperforming sector and DID outperformed</p>
UNDERPERF.	<p>False Positive: Stock WAS classified as outperforming sector and did NOT outperformed</p>	<p>True Negative: Stock was NOT classified as outperforming sector and did not outerperformed</p>

Machine Learning In Finance

Supervised Learning

4.2 Regressions

Machine Learning in Finance

Supervised Learning

4.2.1 Linear Regression Models

4.2.1 Generalized Linear Models

The following are a set of methods intended for regression in which the target value is expected to be a linear combination of the input variables. In mathematical notion, if \hat{y} is the predicted value.

$$\hat{y}(w, x) = w_0 + w_1x_1 + \dots + w_px_p$$

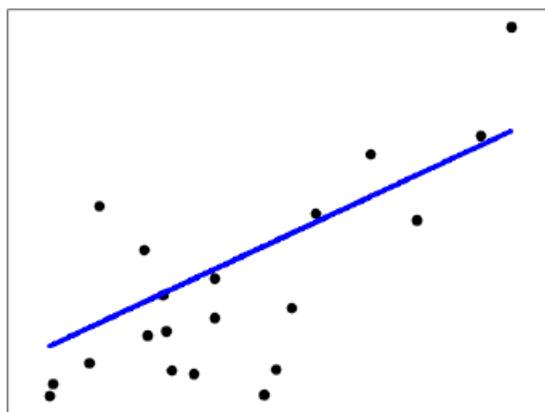
Across the module, we designate the vector $w = (w_1, \dots, w_p)$ as `coef_` and w_0 as `intercept_`.

To perform classification with generalized linear models, see Logistic regression.

4.2.1 Generalized Linear Models – Linear Regression

LinearRegression fits a linear model with coefficients $w = (w_1, \dots, w_p)$ to minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_w \|X_w - y\|_2^2$$



LinearRegression will take in its fit method arrays X, y and will store the coefficients w of the linear model in its coef_ member:

4.2.1 Generalized Linear Models – Linear Regression

```
>>> from sklearn import linear_model
>>> reg = linear_model.LinearRegression()
>>> reg.fit ([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
...
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                  normalize=False)
>>> reg.coef_
array([0.5, 0.5])
```

However, coefficient estimates for Ordinary Least Squares rely on the independence of the model terms. When terms are correlated and the columns of the design matrix X have an approximate linear dependence, the design matrix becomes close to singular and as a result, the least-squares estimate becomes highly sensitive to random errors in the observed response, producing a large variance. This situation of multicollinearity can arise, for example, when data are collected without an experimental design.

Regressions

In supervised learning, an algorithm is provided historical data (both input and output variables), and is trying to find the relationship that has the best predictive power on out-of-sample data. Methods of supervised learning are further classified in methods of regression and methods of classification. Regressions try to predict output variables based on a number of input variables. Classification methods attempt to group or classify output into categories. For instance we may want the output of a model to be a binary action as ‘buy’ or ‘sell’ based on a number of variables. One can think of regression and classification as the same methods, with the forecast of a regression being a continuous number (e.g. market will go up 1%, 1.15%, 2%, etc.) and forecast of classification being a discrete number (e.g. buy=1, sell=0, or volatility regime will be: high=+1, medium=0, low=-1).

Even a simple linear regression can be thought of as a supervised Machine Learning method. However, linear regressions may not be suitable to deal with outliers, a large number of variables, variables that are correlated, or variables that exhibit non-linear behavior. To illustrate one example of the inadequacy of simple linear regression, consider a hypothetical regression forecast of a risky asset price:

Asset Price = 0.2*US Growth - 0.05*EM Growth + 0.1* US HY Bonds + 22* US Equities
– 21* Global Equities

Regressions

Ordinary Linear regression typically produces these types of results (i.e. large opposite sign coefficients) when one includes variables that are highly correlated (such as US and Global Equities). In Big Data strategies, one is expected to include a large number of variables, without necessarily knowing which ones are correlated and which ones are not. The simple regression above would suggest an irrational trading strategy with unnecessarily high leverage and idiosyncratic risk.

One simple extension of linear regression is called a Lasso regression. Lasso regression tries to establish the relationship (a forecast) by choosing the smallest and most relevant set of input variables. In the example above, Lasso regression would ‘realize’ that US and Global Equities are highly correlated, and would penalize and eliminate the large long-short position.

K-nearest neighbors method forecasts data by simply looking at historical samples and establishing what has happened in similar situations as a best forecast of the future. While K-nearest neighbors is a simplistic method that overly relies on historical patterns, it has the advantage of including non-linear effects. In this section, we analyze the following regression methods: Lasso, Ridge, Elastic Net, and K-nearest neighbors.

Regressions - Theory

In supervised learning, we seek to determine an appropriate functional relationship between an output or target variable and a set of input or predictor variables. Given m training examples, denoted by $(\underline{x}^{(i)}, y^{(i)})$, we seek the function h such that $y = h(\underline{x})$. Defining \mathcal{X} as the input space for input variables (say, \mathbb{R}^n) and \mathcal{Y} as the space of output values (say, \mathbb{R}); it is conventional to refer to the function $h: \mathcal{X} \rightarrow \mathcal{Y}$ as the hypothesis function. The learning task is called either a regression or a classification, depending on whether the output space \mathcal{Y} is continuous (as in \mathbb{R}) or discrete (as in $\{0, 1\}$).

In the classical Ordinary Least Squares model for linear regression, one defines $h_{\underline{\theta}}(\underline{x}) = \underline{\theta}^T \underline{x}$, where $\underline{\theta}$ is the parameter or weight vector. Errors during model fitting are penalized using the cost function $J(\underline{\theta}) = \frac{1}{2} \sum_{i=1}^m (h_{\underline{\theta}}(\underline{x}^{(i)}) - y^{(i)})^2$. The

traditional approach was to stack the training examples to form $X = \begin{bmatrix} \underline{x}^{(1)T} \\ \vdots \\ \underline{x}^{(m)T} \end{bmatrix}$ and $\underline{y} = [y^{(1)} \quad \dots \quad y^{(m)}]^T$. Matrix

differentiation of the cost function $J(\underline{\theta}) = \|X\underline{\theta} - \underline{y}\|^2$ yields the normal equations $X^T X \underline{\theta} = X^T \underline{y}$ and the estimator as $\underline{\theta} = (X^T X)^{-1} X^T \underline{y}$. Unfortunately, this traditional approach of theoretical derivation and practical use is not extensible to modern Machine Learning models. So we review this question using two other approaches, namely

- A purely numerical approach, which we shall use to derive perceptron models; and
- A purely probabilistic approach, which we shall use to derive ridge and lasso regressors.

The numerical approach is to note that $J(\underline{\theta})$ can be minimized using the gradient descent algorithm, where the j^{th} element of the vector $\underline{\theta}$ is updated as $\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\underline{\theta})$. Here α is the learning rate. Repeated decrease in the direction of steepest descent of $J(\underline{\theta})$ leads to convergence to a local (global in the case of convex functions) minima. Applied to our definition of $J(\underline{\theta})$ and denoting the j^{th} element of the vector $\underline{x}^{(i)}$ as $x_j^{(i)}$, the gradient descent rule yields $\theta_j \leftarrow \theta_j + \alpha (y^{(i)} - h_{\underline{\theta}}(\underline{x}^{(i)})) x_j^{(i)}$. This rule is called the Widrow-Hoff or Least Mean Squares (LMS) rule in Machine Learning literature.

Regressions - Theory

When the LMS rule is applied – as shown below - in one shot to all training examples, it is called Batch Gradient Descent.

Repeat until convergence

$$\{ \quad \forall j \in \{1, \dots, n\}, \theta_j \leftarrow \theta_j + \alpha \sum_{i=1}^m (y^{(i)} - h_{\underline{\theta}}(\underline{x}^{(i)})) x_j^{(i)} \}$$

If the training set is very large, we can update all the weights using information from each individual training example, instead of iterating over the entire training set to update a single weight parameter. This idea leads to Incremental or Stochastic Gradient Descent (SGD).

Repeat until convergence

$$\{ \forall i \in \{1, \dots, m\} \quad \forall j \in \{1, \dots, n\}, \theta_j \leftarrow \theta_j + \alpha (y^{(i)} - h_{\underline{\theta}}(\underline{x}^{(i)})) x_j^{(i)} \}$$

While stochastic gradient descent lacks theoretical guarantees on convergence of $\underline{\theta}$ to the local minima, one finds its performance to be superior to batch gradient descent on large data sets. We shall use stochastic gradient descent to train many of the models employed in this primer. The above LMS rule shall also recur in identical form when we study logistic classifiers.

To motivate and understand many of the Machine Learning algorithms, researchers rely frequently on probabilistic interpretations. In the context of Ordinary Least Squares, one can model the output as $y^{(i)} = \underline{\theta}^T \underline{x}^{(i)} + \varepsilon^{(i)}$. If $\varepsilon^{(i)} \sim N(0, \sigma^2)$ represents independent and identically distributed (i.i.d.) noise, then the likelihood of observing the particular training set is given by the likelihood function

$$L(\underline{\theta}) = p(y|X; \underline{\theta}) = \prod_{i=1}^m p(y^{(i)}|\underline{x}^{(i)}; \underline{\theta}) = \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \underline{\theta}^T \underline{x}^{(i)})^2}{2\sigma^2}\right).$$

The principle of Maximum Likelihood mandates the choice of parameter $\underline{\theta}$ to maximize the above likelihood expression, i.e. render the appearance of the training set as likely as possible. Maximizing equivalently the log-likelihood $l(\underline{\theta})$, we recover

$$\hat{\theta}_{ML} = \arg \min_{\underline{\theta}} l(\underline{\theta}) = \arg \min_{\underline{\theta}} \log L(\underline{\theta}) = \arg \max_{\underline{\theta}} \frac{1}{2} \| X\underline{\theta} - \underline{y} \|^2.$$

This is same as the expression obtained in the traditional analysis for OLS. The advantage of this probabilistic modeling is that we will extend it using Bayesian priors to derive more stable linear regression techniques like ridge and lasso.

Machine Learning In Finance

Supervised Learning

4.2.2 Modern Regressions

Penalized Regression Techniques: Lasso, Ridge, and Elastic Net

Penalized regression techniques like Lasso (also, spelt as LASSO) and Ridge are simple modifications of ordinary linear regression aimed at creating a more robust output model in the presence of a large number of potentially correlated variables. When the number of input features is large or when the input features are correlated, classical linear regression has a tendency to overfit and yield spurious coefficients.

LASSO, Ridge, and Elastic Net regressions are also examples of ‘regularization’ – a technique in Machine Learning that is expected to reduce the out-of-sample forecasting errors (but does not help with reducing in-sample backtest errors).

In ordinary linear regression, we forecast the value y to be a linear combination of the inputs x_1, x_2, \dots, x_n . In short we assume that variable y has ‘Betas’ to a number of variables x (plus some random noise)

$$y = \beta_0 + \sum_{i=1}^n \beta_i x_i + \varepsilon.$$

To find the ‘betas’ linear regression minimizes the historical error (square of error) between actual observations of variable ‘ y ’, and predicted (or model) values of the variable. This is the reason the method is also called least-squares (since it minimizes the square of errors):

$$\text{OLS: Minimize Historical Sum of } \left(y - (\beta_0 + \sum_{i=1}^n \beta_i x_i) \right)^2.$$

Penalized Regression Techniques: Lasso, Ridge, and Elastic Net

This minimization is not stable and can yield spurious and/or large values of betas. One way to prevent that from occurring is to change the objective function in the minimization above. Instead of minimizing the least-squares objective, we can modify it by adding a penalty term that reflects our aversion towards complex models with large ‘betas’. If we change the objective to include a penalty term equal to the absolute value of the beta coefficients, i.e.

$$\text{Lasso: Minimize Historical Sum of } \left(y - (\beta_0 + \sum_{i=1}^n \beta_i x_i) \right)^2 + \alpha \sum_{i=1}^n |\beta_i|,$$

then the optimizer will set ‘unnecessary’ and very large betas to zero. The addition of a penalty term equal to the absolute value of the coefficients is called L1 regularization and the modified linear regression procedure is called **Lasso (or LASSO)**.

By concentrating only on the most relevant predictors, Lasso performs an implicit feature selection for us.

The objective function for Lasso can be understood as follows:

- If we set $\alpha = 0$, then we recover the coefficients of ordinary linear regression.
- As α increases, we choose a smaller and smaller set of predictors, concentrating only on the most important ones

Here, α is called a parameter of the model. Similarly, if we tweak our objective to include the square of the ‘betas’ we arrive at Ridge regression

Penalized Regression Techniques: Lasso, Ridge, and Elastic Net

Similarly, if we tweak our objective to include the square of the ‘betas’ we arrive at Ridge regression

$$\text{Ridge: Minimize Historical Sum of } \left(y - (\beta_0 + \sum_{i=1}^n \beta_i x_i) \right)^2 + \alpha \sum_{i=1}^n \beta_i^2 ,$$

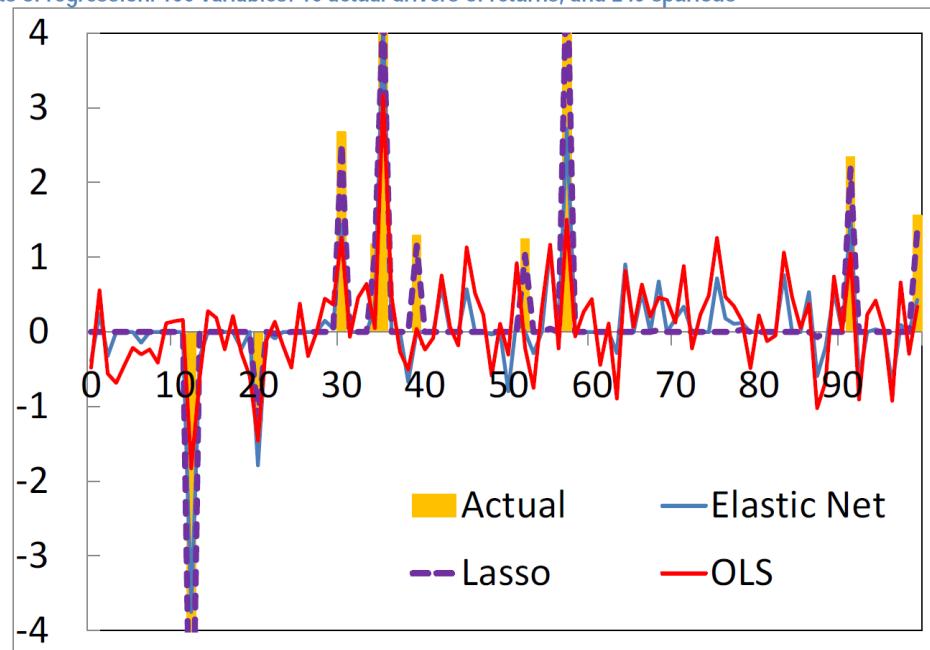
The additional penalty related to the square of the magnitude of the coefficients is called L2 regularization and the modified linear regression procedure is called Ridge. An intermediate objective between the Ridge and Lasso objectives is used by Elastic Net regression, which finds

$$\text{Elastic Net: Minimize Historical Sum of } \left(y - (\beta_0 + \sum_{i=1}^n \beta_i x_i) \right)^2 + \alpha_1 \sum_{i=1}^n |\beta_i| + \alpha_2 \sum_{i=1}^n \beta_i^2 .$$

Penalized Regression Techniques: Lasso, Ridge, and Elastic Net

We illustrate the 3 penalized regression examples with a hypothetical example (before considering an example of a realistic trading strategy). In the hypothetical example, we have chosen 10 variables that are actually driving the forecast for asset returns and added 90 spurious variables that are adding noise (throwing off the regression model). The algorithms were given all 100 variables (10 actual + 90 spurious) and asked to predict the weights for each feature. We plot the coefficients in graph below: horizontal axis ranges from 1 to 100 (reflecting the input variables) and vertical axis plots the values of beta coefficients as predicted by each algorithm. An ideal algorithm would select the 10 actual variables (light blue), and would discard the spurious ones.

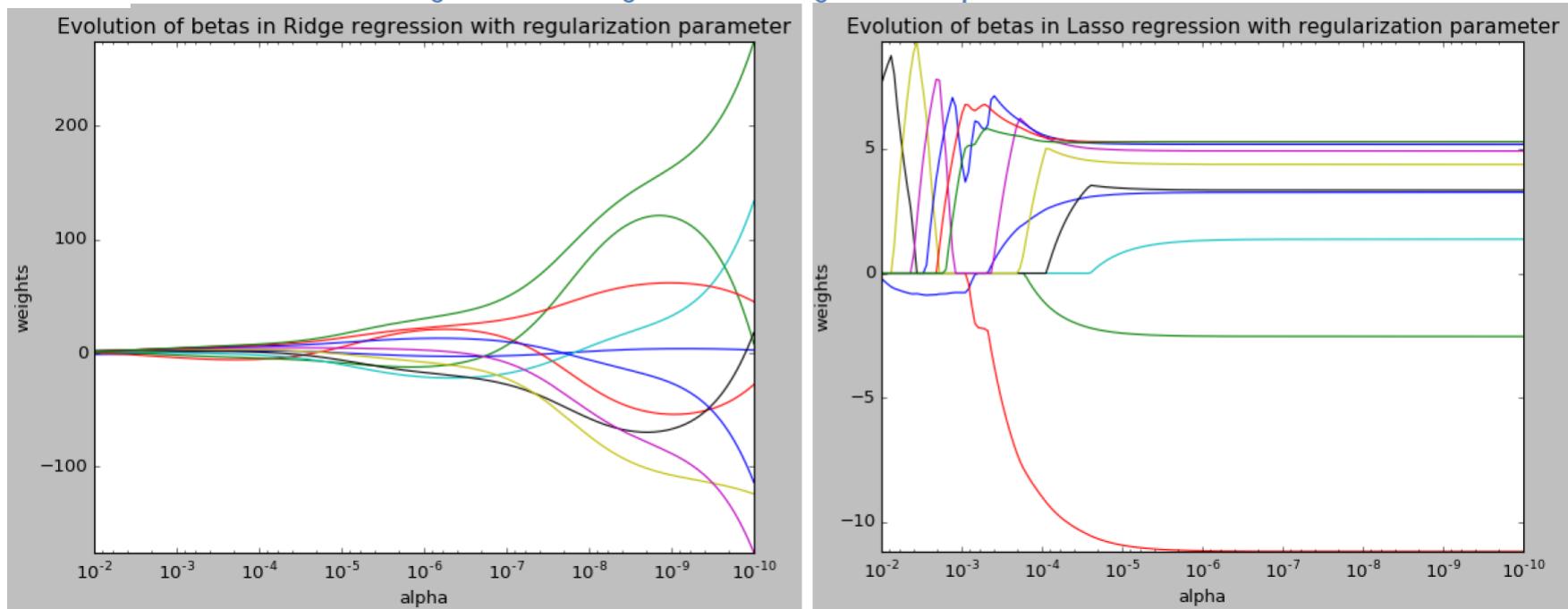
Coefficients of regression. 100 variables: 10 actual drivers of returns, and 240 spurious



Penalized Regression Techniques: Lasso, Ridge, and Elastic Net

Note that Lasso picked up a sub-set of actual coefficients. Ridge assigned weights to all coefficients, irrespective of whether they were spurious noise terms. Elastic net behaved in an intermediate way between Ridge and Lasso. As one increases the regularization parameter of the model (α), the models start suppressing spurious variables and focusing on actual ones. This is illustrated in the figures below showing the evolution of model 'betas' as we increase/decrease regularization parameter.

Evolution of betas in Ridge and Lasso regression with regularization parameter



Bayesian Interpretation Penalized Regressions

There is a natural Bayesian interpretation for both Ridge and Lasso regression³¹. We have shown earlier that for the linear model (assuming zero means) $\underline{y} = X\underline{\beta} + \underline{\varepsilon}$, $\underline{\varepsilon} \sim N(\underline{0}, I)$, that the maximum likelihood estimate for $\underline{\beta}$ yields the ordinary least squares (OLS) answer. Suppose we assume a Laplacian prior on $\underline{\beta}$, i.e. $f(\underline{\beta}) \propto e^{-\lambda|\underline{\beta}|}$, the posterior distribution of $\underline{\beta}$ is given by

$$f(\underline{\beta} | \underline{y}) \propto f(\underline{\beta}) \cdot f(\underline{y} | \underline{\beta}) = e^{-\lambda|\underline{\beta}|} \cdot e^{-\frac{1}{2}\|\underline{y} - X\underline{\beta}\|^2}.$$

This implies that the maximum a posteriori (MAP) estimate for $\underline{\beta}$ is given as

$$\hat{\underline{\beta}}_{MAP} = \arg \max f(\underline{\beta} | \underline{y}) = \arg \min \|\underline{y} - X\underline{\beta}\|^2 + 2\lambda|\underline{\beta}|.$$

This is the same optimization as used in Lasso regression. Similarly, it is easy to see that assuming a Gaussian prior on $\underline{\beta}$, $f(\underline{\beta}) \propto e^{-\frac{1}{2}\|\underline{\beta}\|^2}$ will coerce the MAP estimate to obtain the Ridge regression estimate. It is known from Bayesian analysis, that assuming a prior avoids overfitting by using our prior knowledge (in this case, knowledge of the parsimonious nature of the model); unsurprisingly, using a prior distribution and deriving the MAP estimate leads to robust regressors³².

4.2.2 Generalized Linear Models – Ridge Regression

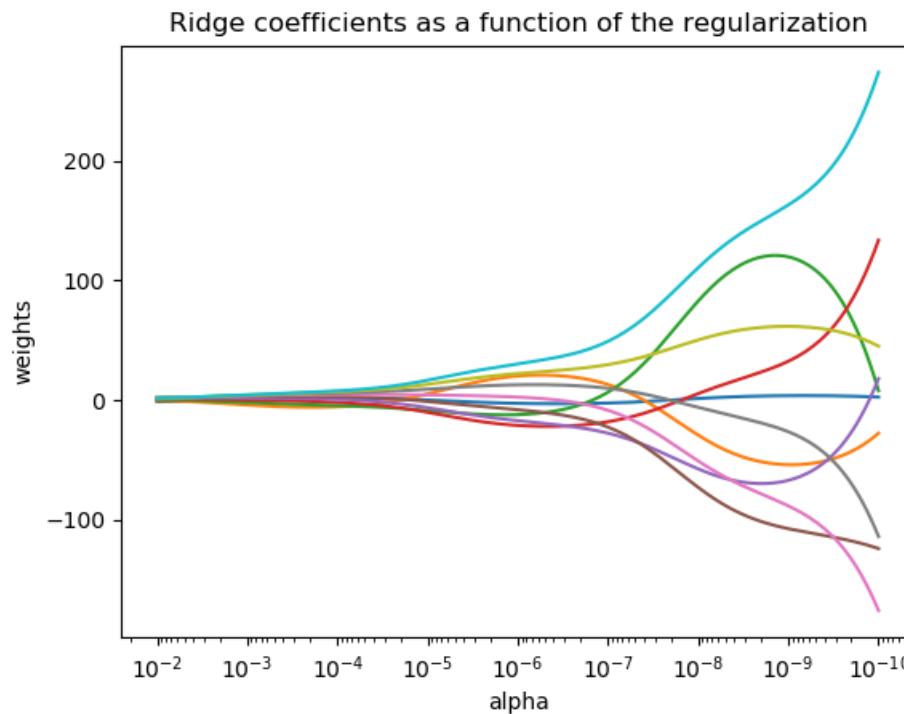
Ridge regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of coefficients. The ridge coefficients minimize a penalized residual sum of squares :

$$\min_w \|X_w - y\|_2^2 + \alpha \|w\|_2^2$$

Here, $\alpha \geq 0$ is a complexity parameter that controls the amount of shrinkage: the larger the value of α , the greater the amount of shrinkage and thus the coefficients become more robust to collinearity.

4.2.2 Generalized Linear Models – Ridge Regression

Here, $\alpha \geq 0$ is a complexity parameter that controls the amount of shrinkage: the larger the value of α , the greater the amount of shrinkage and thus the coefficients become more robust to collinearity.



4.2.2 Generalized Linear Models – Lasso

The Lasso is a linear model that estimates sparse coefficients. It is useful in some contexts due to its tendency to prefer solutions with fewer parameter values, effectively reducing the number of variables upon which the given solution is dependent. For this reason, the Lasso and its variants are fundamental to the field of compressed sensing. Under certain conditions, it can recover the exact set of non-zero weights (see Compressive sensing: tomography reconstruction with L1 prior (Lasso)).

Mathematically, it consists of a linear model trained with ℓ_1 prior as regularizer. The objective function to minimize is:

$$\min_w \frac{1}{2n_{samples}} \|X_w - y\|_2^2 + \alpha \|w\|_1$$

The lasso estimate thus solves the minimization of the least-squares penalty with $\alpha \|w\|_1$ added, where α is a constant and $\|w\|_1$ is the ℓ_1 -norm of the parameter vector.

4.2.2 Generalized Linear Models – Lasso

The implementation in the class Lasso uses coordinate descent as the algorithm to fit the coefficients.

```
>>> from sklearn import linear_model
>>> reg = linear_model.Lasso(alpha = 0.1)
>>> reg.fit([[0, 0], [1, 1]], [0, 1])
Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
>>> reg.predict([[1, 1]])
array([0.8])
```

Machine Learning In Finance

Supervised Learning

4.2.3 Non-Linear Regressions

4.2.3 Non-Linear Regressions Polynomial

One common pattern within machine learning is to use linear models trained on nonlinear functions of the data. This approach maintains the generally fast performance of linear methods, while allowing them to fit a much wider range of data.

For example, a simple linear regression can be extended by constructing polynomial features from the coefficients.

In the standard linear regression case, you might have a model that looks like this for two-dimensional data:

$$\hat{y}(w, x) = w_0 + w_1 x_1 + w_2 x_2$$

If we want to fit a paraboloid to the data instead of a plane, we can combine the features in second-order polynomials, so that the model looks like this:

$$\hat{y}(w, x) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2$$

4.2.3 Non-Linear Regressions Polynomial

The (sometimes surprising) observation is that this is still a linear model: to see this, imagine creating a new variable

$$z = [x_1, x_2, x_1x_2, x_1^2, x_2^2]$$

With this re-labeling of the data, our problem can be written

$$\hat{y}(w, x) = w_0 + w_1z_1 + w_2z_2 + w_3z_3 + w_4z_4 + w_5z_5$$

We see that the resulting polynomial regression is in the same class of linear models we'd considered above (i.e. the model is linear in w) and can be solved by the same techniques. By considering linear fits within a higher-dimensional space built with these basis functions, the model has the flexibility to fit a much broader range of data.

Here is an example of applying this idea to one-dimensional data, using polynomial features of varying degrees:

Machine Learning In Finance

Supervised Learning

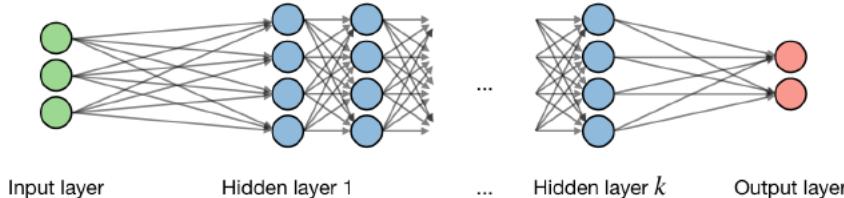
4.2.4 Neural Networks

Deep Learning

Neural Networks

Neural networks are a class of models that are built with layers. Commonly used types of neural networks include convolutional and recurrent neural networks.

□ **Architecture** – The vocabulary around neural networks architectures is described in the figure below:

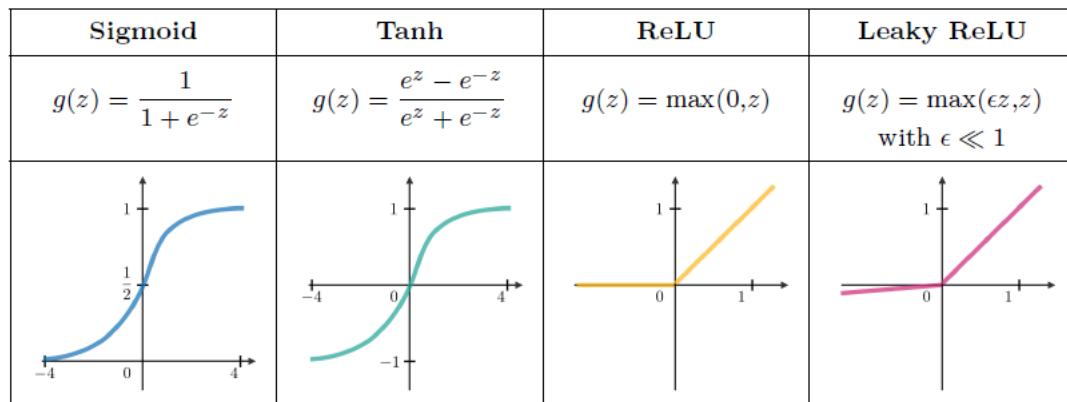


By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$z_j^{[i]} = w_j^{[i]T} x + b_j^{[i]}$$

where we note w , b , z the weight, bias and output respectively.

□ **Activation function** – Activation functions are used at the end of a hidden unit to introduce non-linear complexities to the model. Here are the most common ones:



□ **Cross-entropy loss** – In the context of neural networks, the cross-entropy loss $L(z,y)$ is commonly used and is defined as follows:

$$L(z,y) = - \left[y \log(z) + (1 - y) \log(1 - z) \right]$$

Deep Learning

□ **Learning rate** – The learning rate, often noted η , indicates at which pace the weights get updated. This can be fixed or adaptively changed. The current most popular method is called Adam, which is a method that adapts the learning rate.

□ **Backpropagation** – Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to weight w is computed using chain rule and is of the following form:

$$\frac{\partial L(z,y)}{\partial w} = \frac{\partial L(z,y)}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w}$$

As a result, the weight is updated as follows:

$$w \leftarrow w - \eta \frac{\partial L(z,y)}{\partial w}$$

□ **Updating weights** – In a neural network, weights are updated as follows:

- Step 1: Take a batch of training data.
- Step 2: Perform forward propagation to obtain the corresponding loss.
- Step 3: Backpropagate the loss to get the gradients.
- Step 4: Use the gradients to update the weights of the network.

□ **Dropout** – Dropout is a technique meant at preventing overfitting the training data by dropping out units in a neural network. In practice, neurons are either dropped with probability p or kept with probability $1 - p$.

Deep Learning

Convolutional Neural Networks

□ **Convolutional layer requirement** – By noting W the input volume size, F the size of the convolutional layer neurons, P the amount of zero padding, then the number of neurons N that fit in a given volume is such that:

$$N = \frac{W - F + 2P}{S} + 1$$

□ **Batch normalization** – It is a step of hyperparameter γ, β that normalizes the batch $\{x_i\}$. By noting μ_B, σ_B^2 the mean and variance of that we want to correct to the batch, it is done as follows:

$$x_i \leftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

It is usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

Recurrent Neural Networks

□ **Types of gates** – Here are the different types of gates that we encounter in a typical recurrent neural network:

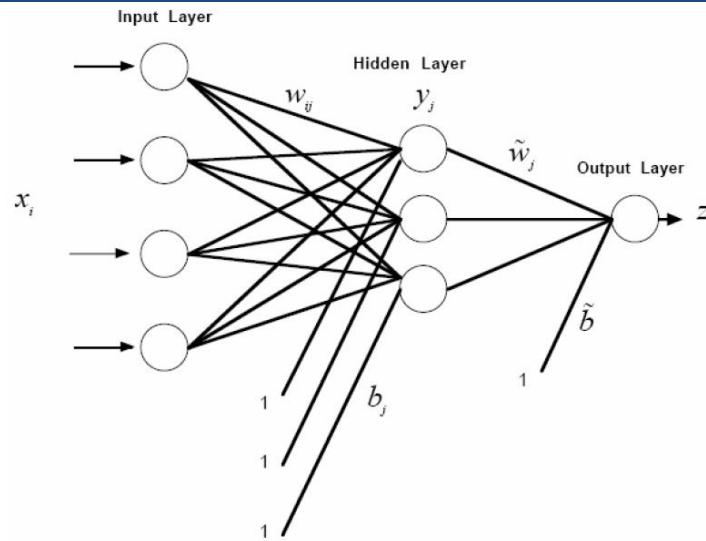
Input gate	Forget gate	Output gate	Gate
Write to cell or not?	Erase a cell or not?	Reveal a cell or not?	How much writing?

□ **LSTM** – A long short-term memory (LSTM) network is a type of RNN model that avoids the vanishing gradient problem by adding 'forget' gates.

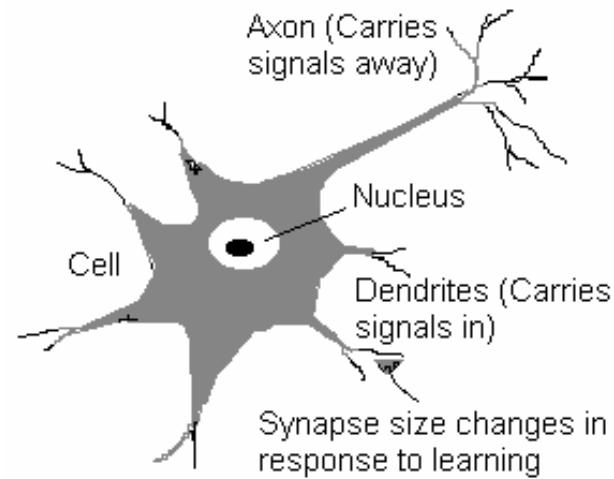
Neural Networks Concept

Artificial neural networks (ANN's) are **universal function approximators** from the point of view of financial modeling. They are made up of interconnecting artificial neurons (programming constructs that mimic the properties of biological neurons). Artificial neural networks may either be used to gain an understanding of biological neural networks, or for solving artificial intelligence problems without necessarily creating a model of a real biological system. From a statistical perspective are linear or non-linear regressions.

A Schematic Diagram of a Neuron



The Neuron Analogy

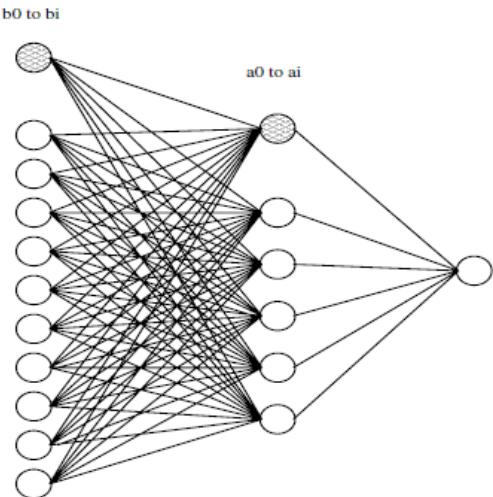


Neural Networks for Finance

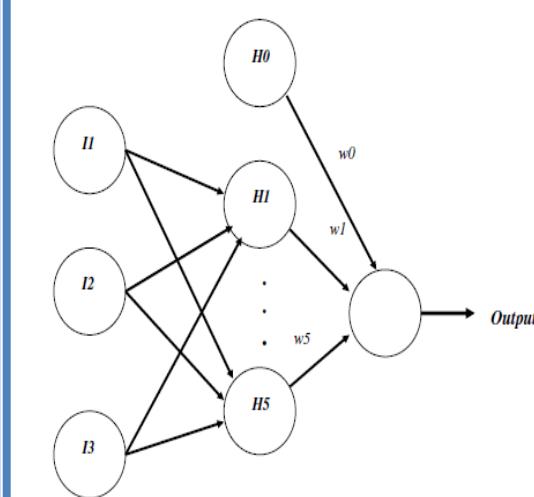
Common Neural Networks Structures

Neural Networks Structures

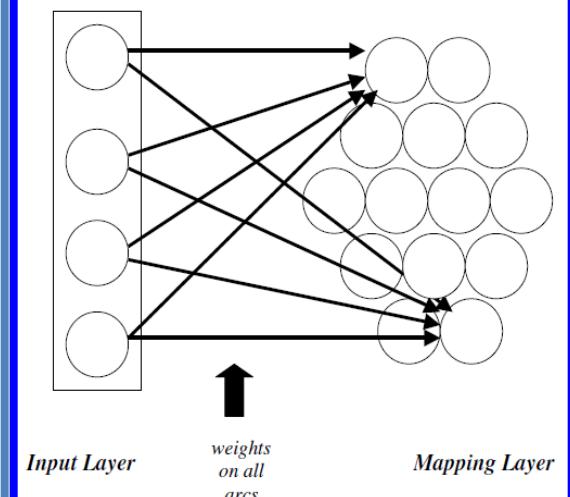
Multi Layer
Perceptron



Radial Basis
Functions Networks

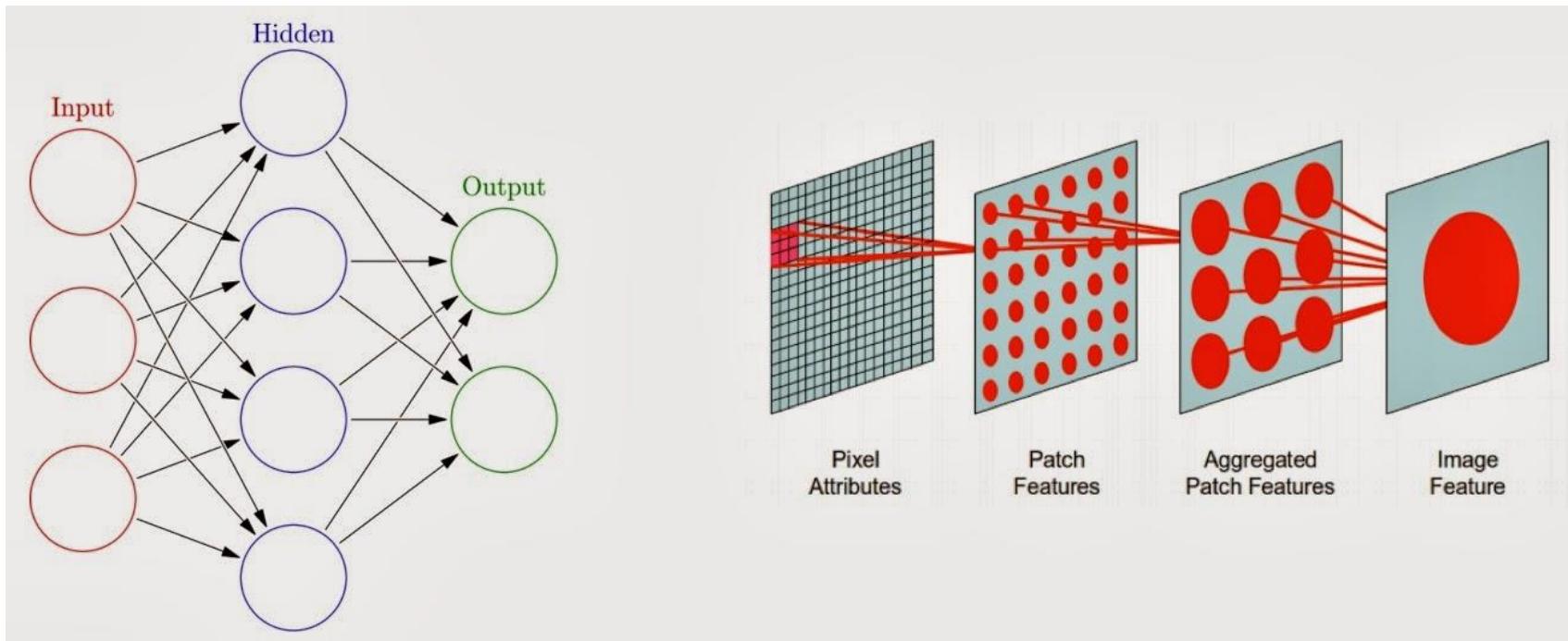


Self organising
Maps



Neural Networks for Finance – Deep Learning

Common Neural Networks Structures



What Is A Neural Network?

Like the linear and polynomial approximation methods, a neural network relates a set of input variables $\{x_i\}, i = 1, \dots, k$, to a set of one or more output variables, $\{y_j\}, j = 1, \dots, k *$.

The difference between a neural network and the other approximation methods is that the neural network makes use of one or more hidden layers, in which the input variables are squashed or transformed by a special function, known as a logistic or logsigmoid transformation.

While this hidden layer approach may seem esoteric, it represents a very efficient way to model nonlinear statistical processes.

Closed-Form Solution	Parametric	Semi-Parametric
Yes	Linear	Polynomial
No	GARCH-M	Neural Network

Feedforward Networks

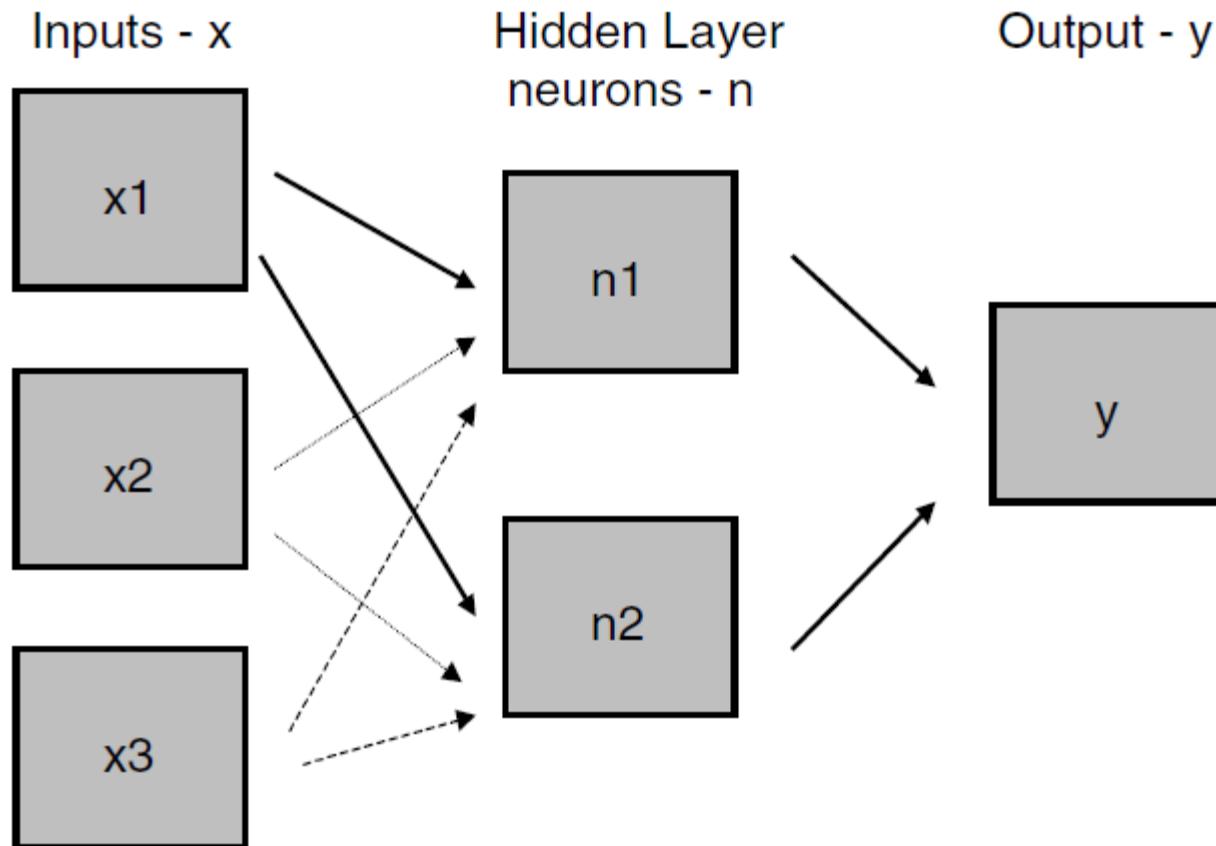
The architecture on a neural network with one hidden layer containing two neurons, three input variables $\{x_i\}, i = 1, 2, 3$, and one output y .

We see parallel processing. In addition to the sequential processing of typical linear systems, in which only observed inputs are used to predict an observed output by weighting the input neurons, the two neurons in the hidden layer process the inputs in a parallel fashion to improve the predictions.

The connectors between the input variables, often called input neurons, and the neurons in the hidden layer, as well as the connectors between the hidden-layer neurons and the output variable, or output neuron, are called synapses. Most problems we work with, fortunately, do not involve a large number of neurons engaging in parallel processing, thus the parallel processing advantage, which applies to the way the brain works with its massive number of neurons, is not a major issue.

This **single-layer feedforward or multiperceptron network** with one hidden layer is the most basic and commonly used neural network in economic and financial applications. More generally, the network represents the way the human brain processes input sensory data, received as input neurons, into recognition as an output neuron. As the brain develops, more and more neurons are interconnected by more synapses, and the signals of the different neurons, working in parallel fashion, in more and more hidden layers, are combined by the synapses to produce more nuanced insight and reaction.

Feedforward Networks



Neural Networks in Statistics

In a more general statistical framework, neural network approximation is a **sieve estimator**. In the univariate case, with one input x , an approximating function of order m , Ψ_m , is based on a non-nested sequence of approximating spaces:

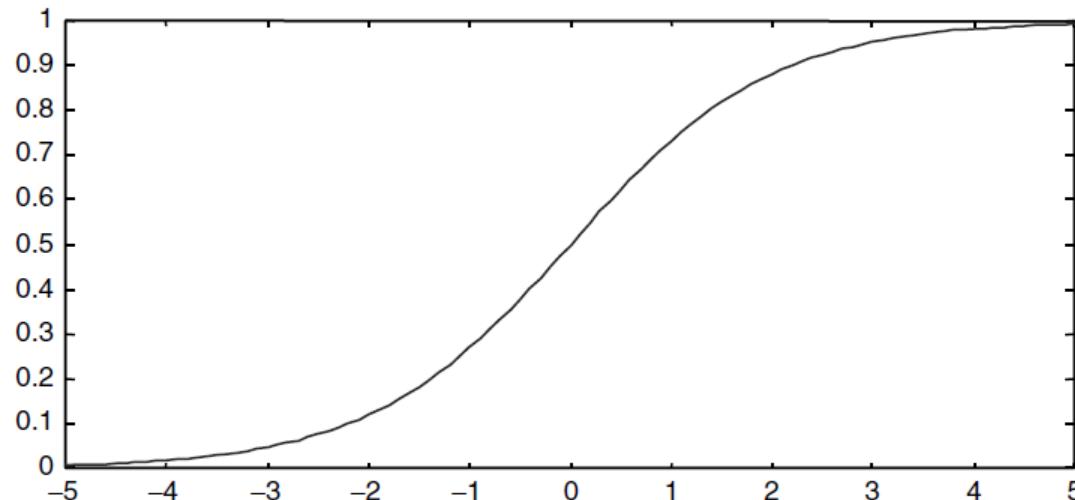
$$\Psi_m = [\psi_{m,0}(x), \psi_{m,1}(x), \dots, \psi_{m,m}(x)]$$

Beresteanu (2003) points out that each finite expansion, $\psi_{m,0}(x)$, $\psi_{m,1}(x)$, . . . $\psi_{m,m}(x)$, can potentially be based on a different set of functions [Beresteanu (2003), p. 9]. We now discuss the most commonly used functional forms in the neural network literature.

Activation / Squasher Functions

The neurons process the input data in two ways: first by forming linear combinations of the input data and then by “squashing” these linear combinations through the logsigmoid function. Figure illustrates the operation of the typical logistic or logsigmoid activation function, also known as a squasher function, on a series ranging from -5 to $+5$. The inputs are thus transformed by the squashers before transmitting their effects on the output.

The appeal of the logsigmoid transform function comes from its threshold



MLP network

The feedforward network coupled with the logsigmoid activation functions is also known as the multi-layer perception or MLP network.

It is the basic workhorse of the neural network forecasting approach, in the sense that researchers usually start with this network as the first representative network alternative to the linear forecasting model.variables.

$$n_{k,t} = \omega_{k,0} + \sum_{i=1}^{i^*} \omega_{k,i} x_{i,t}$$

$$N_{k,t} = L(n_{k,t})$$

$$= \frac{1}{1 + e^{-n_{k,t}}}$$

$$y_t = \gamma_0 + \sum_{k=1}^{k^*} \gamma_k N_{k,t}$$

MLP network Gaussian Function

The Gaussian function does not have as wide a distribution as the logsigmoid function, in that it shows little or no response when the inputs take extreme values (below -2 or above +2 in this case), whereas the logsigmoid does show some response. Moreover, within critical changes, such as [-2, 0] and [0, 2], the slope of the cumulative Gaussian function is much steeper.

The mathematical representation of the feedforward network with the Gaussian activation functions is given by the following system:

$$n_{k,t} = \omega_{k,0} + \sum_{i=1}^{i^*} \omega_{k,i} x_{i,t}$$

$$N_{k,t} = \Phi(n_{k,t})$$

$$= \int_{-\infty}^{n_{k,t}} \sqrt{\frac{1}{2\pi}} e^{-\frac{1}{2}n_{k,t}^2}$$

$$y_t = \gamma_0 + \sum_{k=1}^{k^*} \gamma_k N_{k,t}$$

Radial Basis Functions

The radial basis network function (RBF) network makes use of the radial basis or Gaussian density function as the activation function, but the structure of the network is different from the feedforward or MLP networks we have discussed so far. The input neuron may be a linear combination of regressors, as in the other networks, but there is only one input signal, only one set of coefficients of the input variables x . The signal from this input layer is the same to all the neurons, which in turn are Gaussian transformations, around k^* different means, of the input signals. Thus the input signals have different centers for the radial bases or normal distributions.

The differing Gaussian transformations are combined in a linear fashion for forecasting the output.

$$\underset{\langle \omega, \mu, \gamma \rangle}{\text{Min}} \sum_{t=0}^T (y_t - \hat{y}_t)^2$$

$$n_t = \omega_0 + \sum_{i=1}^{i^*} \omega_i x_{i,t}$$

$$R_{k,t} = \phi(n_t; \mu_k)$$

$$= \frac{1}{\sqrt{2\pi\sigma_{n-\mu_k}}} \exp\left(\frac{-[n_t - \mu_k]}{\sigma_{n-\mu_k}}\right)^2$$

$$\hat{y}_t = \gamma_0 + \sum_{k=1}^{k^*} \gamma_k N_{k,t}$$

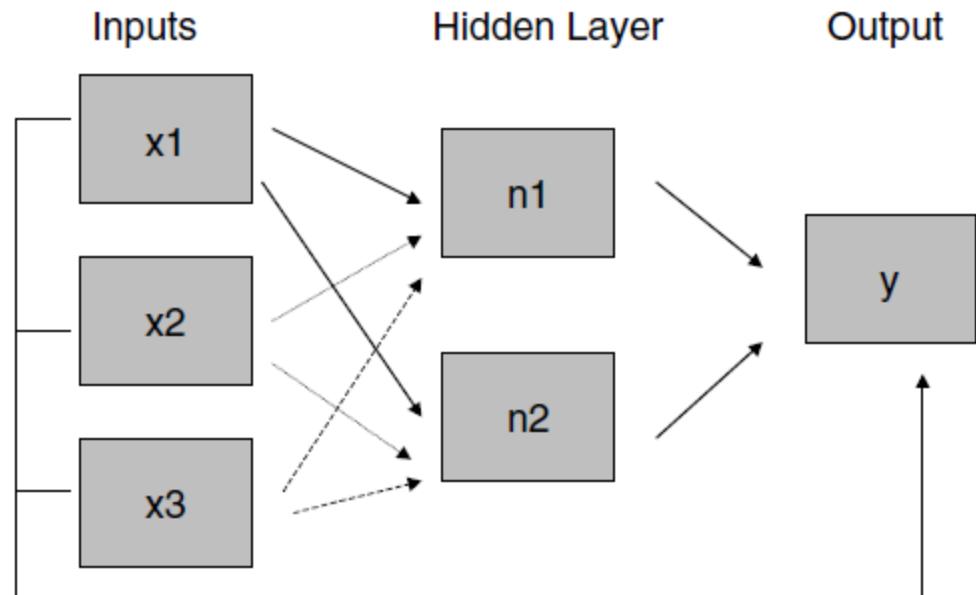
Jump Connections

One alternative to the pure feedforward network or sieve network is a feedforward network with jump connections, in which the inputs x have direct linear links to output y , as well as to the output through the hidden layer of squashed functions..

$$n_{k,t} = \omega_{k,0} + \sum_{i=1}^{i^*} \omega_{k,i} x_{i,t}$$

$$N_{k,t} = \frac{1}{1 + e^{-n_{k,t}}}$$

$$\hat{y}_t = \gamma_0 + \sum_{k=1}^{k^*} \gamma_k N_{k,t} + \sum_{i=1}^{i^*} \beta_i x_{i,t}$$



Multilayered Feedforward Networks

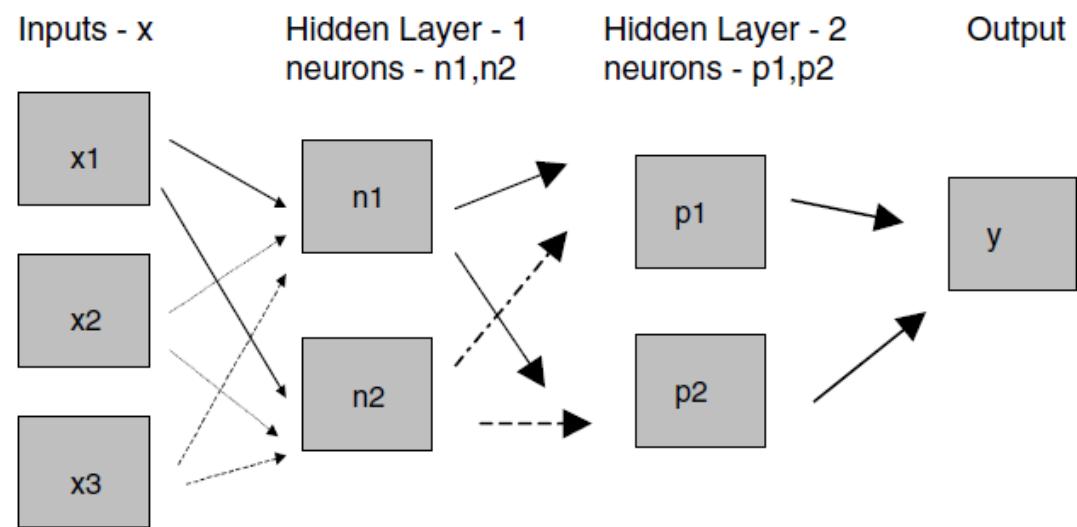
$$n_{k,t} = \omega_{k,0} + \sum_{i=1}^{i^*} \omega_{k,i} x_{i,t}$$

$$N_{k,t} = \frac{1}{1 + e^{-n_{k,t}}}$$

$$p_{l,t} = \rho_{l,0} + \sum_{k=1}^{k^*} \rho_{l,k} N_{k,t}$$

$$P_{l,t} = \frac{1}{1 + e^{-p_{l,t}}}$$

$$y_t = \gamma_0 + \sum_{l=1}^{l^*} \gamma_l P_{l,t}$$



It should be clear that adding a second hidden layer increases the number of parameters to be estimated by the factor $(k^* + 1)(l^* - 1) + (l^* + 1)$, since the feedforward network with one hidden layer, with i^* inputs and k^* neurons, has $(i^* + 1)k^* + (k^* + 1)$ parameters, while a similar network with two hidden layers, with l^* neurons in the second hidden layer, has $(i^* + 1)k^* + (k^* + 1)l^* + (l^* + 1)$ hidden layers.

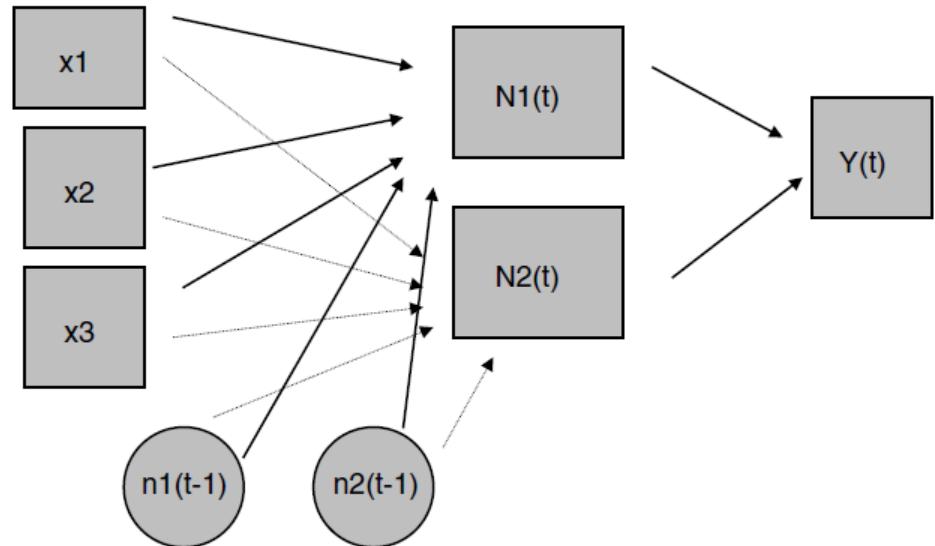
Recurrent Networks

Another commonly used neural architecture is the Elman recurrent network.

This network allows the neurons to depend not only on the input variables x , but also on their own lagged values. Thus the Elman network builds “memory” in the evolution of the neurons. This type of network is similar to the commonly used moving average (MA) process in time-series analysis. MA process :

$$y_t = \beta_0 + \sum_{i=1}^{i^*} \beta_i x_{i,t} + \epsilon_t + \sum_{j=1}^q \nu_j \hat{\epsilon}_{t-j}$$

$$\hat{\epsilon}_{t-j} = y_{t-j} - \hat{y}_{t-j}$$



$$n_{k,t} = \omega_{k,0} + \sum_{i=1}^{i^*} \omega_{k,i} x_{i,t} + \sum_{k=1}^{k^*} \phi_k n_{k,t-1}$$

$$N_{k,t} = \frac{1}{1 + e^{-n_{k,t}}}$$

$$y_t = \gamma_0 + \sum_{k=1}^{k^*} \gamma_k N_{k,t}$$

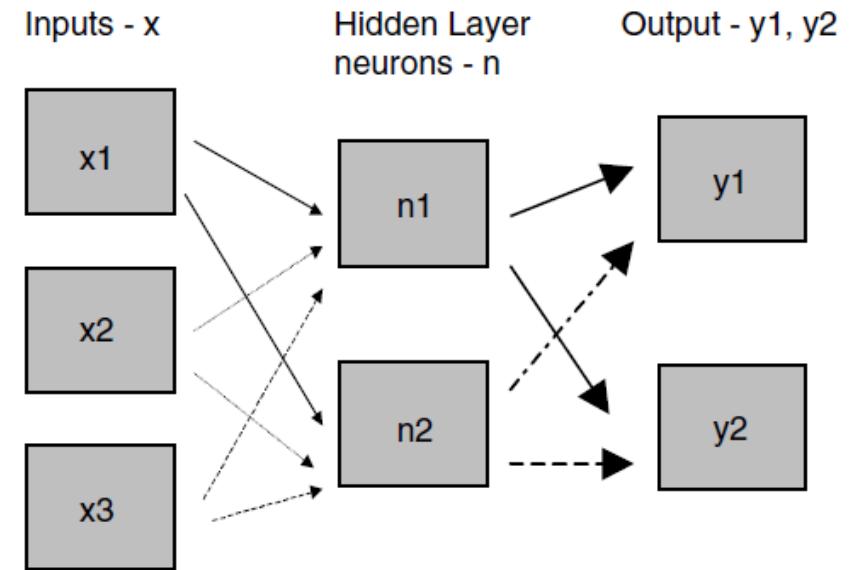
Networks with Multiple Outputs

$$n_{k,t} = \omega_{k,0} + \sum_{i=1}^{i^*} \omega_{k,i} x_{i,t}$$

$$N_{k,t} = \frac{1}{1 + e^{-n_{k,t}}}$$

$$y_{1,t} = \gamma_{1,0} + \sum_{k=1}^{k^*} \gamma_{1,k} N_{k,t}$$

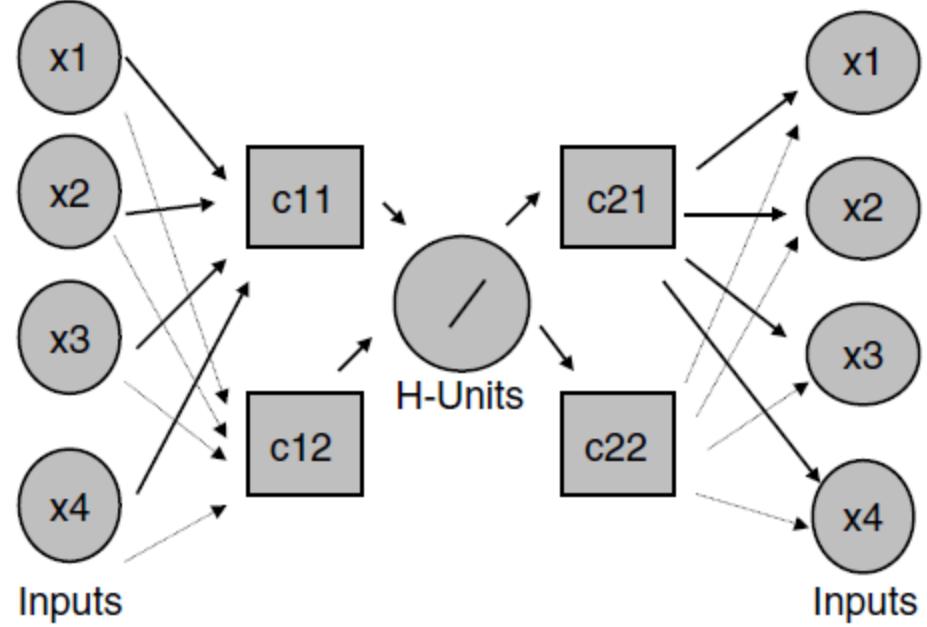
$$y_{2,t} = \gamma_{2,0} + \sum_{k=1}^{k^*} \gamma_{2,k} N_{k,t}$$



The use of a single feedforward network with multiple outputs makes sense, of course, when the outputs of the network are closely related or dependent on the same set of input variables. This type of network is especially useful, as well as economical or parsimonious in terms of parameters, when we are forecasting a specific variable, such as inflation, at different horizons. The set of input variables would be the usual determinants of inflation, such as lags of inflation, and demand and cost variables.

Nonlinear Principal Components: Auto - Encoders

Besides forecasting specific target or output variables, which are determined or predicted by specific input variables or regressors, we may wish to use a neural network for dimensionality reduction or for distilling a large number of potential input variables into a smaller subset of variables that explain most of the variation in the larger data set. Estimation of such networks is called unsupervised training, in the sense that the network is not evaluated or supervised by how well it predicts a specific readily observed target variable.

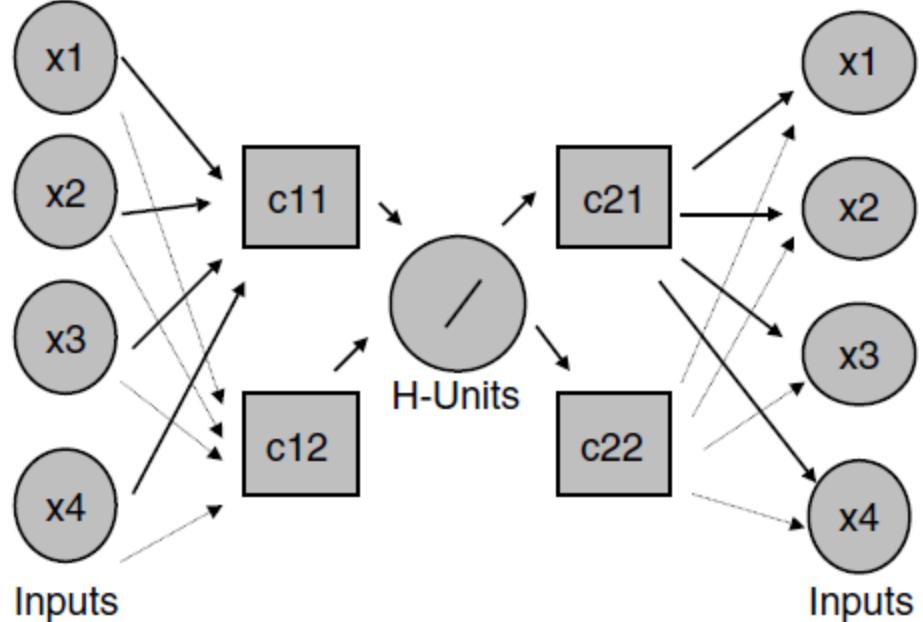


Nonlinear Principal Components: Auto - Encoders

The four input variables in this network are encoded by two intermediate logsigmoid units, C11 and C12, in a dimensionality reduction mapping.

These two encoding units are combined linearly to form H neural principal components. The H-units in turn are decoded by two decoding logsigmoid units C21 and C22, in a reconstruction mapping, which are combined linearly to regenerate the inputs as the output layers.¹¹

Such a neural network is known as an auto-associative mapping, because it maps the input variables x_1, \dots, x_4 into themselves.



Nonlinear Principal Components: Auto - Encoders

$$EN_j = \sum_{k=1}^K \alpha_{j,k} X_k$$

$$\mathbf{EN}_j = \frac{1}{1 + \exp(-EN_j)}$$

$$H_p = \sum_{j=1}^J \beta_{p,j} \mathbf{EN}_j$$

$$DN_j = \sum_{p=1}^P \gamma_{j,p} H_p$$

$$\mathbf{DN}_j = \frac{1}{1 + \exp(-DN_j)}$$

$$\hat{X}_k = \sum_{j=1}^J \delta_{k,j} \mathbf{DN}_j$$

$$\text{Min} \sum_{j=1}^k \sum_{t=1}^T [x_{jt} - \hat{x}_{jt}]^2$$

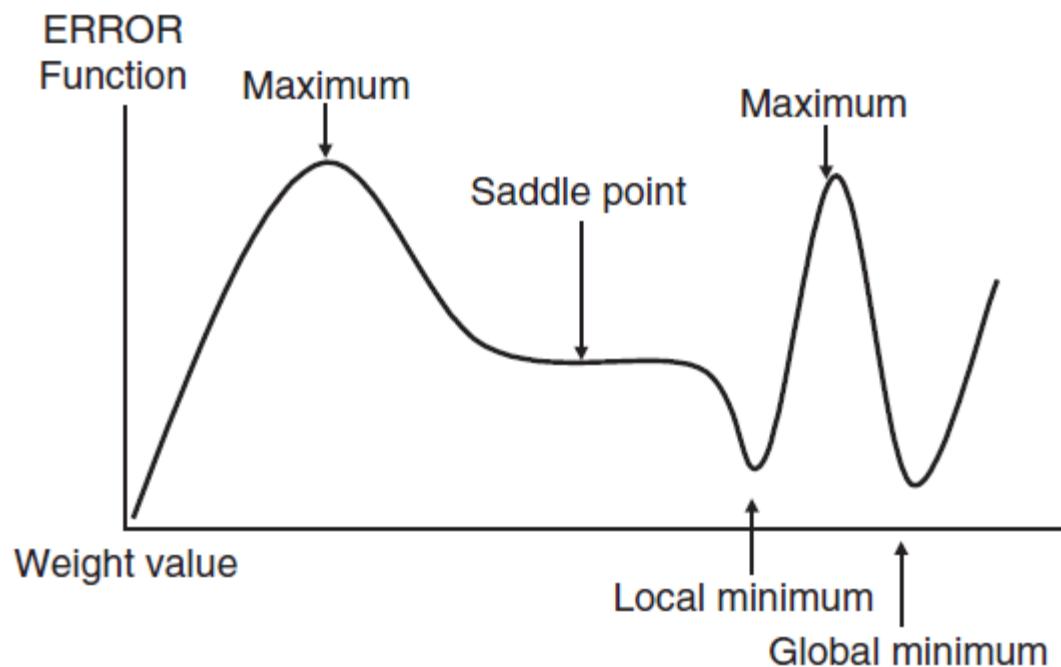
<https://blog.keras.io/building-autoencoders-in-keras.html>

Machine Learning in Finance

Learning

The Nonlinear Estimation Problem

Finding the coefficient values for a neural network, or any nonlinear model, is not an easy job—certainly not as easy as parameter estimation with a linear approximation. A neural network is a highly complex nonlinear system. There may be a multiplicity of locally optimal solutions, none of which deliver the best solution in terms of minimizing the differences between the model predictions y and the actual values of y . Thus, neural network estimation takes time and involves the use of alternative methods.



The Nonlinear Estimation Problem

Finding the coefficient values for a neural network, or any nonlinear model, is not an easy job—certainly not as easy as parameter estimation with a linear approximation. A neural network is a highly complex nonlinear system. There may be a multiplicity of locally optimal solutions, none of which deliver the best solution in terms of minimizing the differences between the model predictions y and the actual values of y . Thus, neural network estimation takes time and involves the use of alternative methods.

$$\min_{\Omega} \Psi(\Omega) = \sum_{t=1}^T (y_t - \hat{y}_t)^2$$

$$\hat{y}_t = f(x_t; \Omega)$$

Clearly, $\Psi(\Omega)$ is a nonlinear function of Ω . All nonlinear optimization starts with an initial guess of the solution, Ω_0 , and searches for better solutions, until finding the best possible solution within a reasonable amount of searching.

The Nonlinear Estimation Problem

We discuss three ways to minimize the function $\Psi(\Omega)$:

1. A **local gradient-based search**, in which we compute first- and secondorder derivatives of Ψ with respect to elements of the parameter vector Ω , and continue with updating of the initial guess of Ω , by derivatives, until stopping criteria are reached
2. A stochastic search, called **simulated annealing**, which does not rely on the use of first- and second-order derivatives, but starts with an initial guess Ω_0 , and proceeds with random updating of the initial coefficients until a “cooling temperature” or stopping criterion is reached
3. An evolutionary stochastic search, called the **genetic algorithm**, which starts with a population of p initial guesses, $[\Omega_{01}, \Omega_{02} \dots \Omega_{0p}]$, and updates the population of guesses by genetic selection, breeding, and mutation, for many generations, until the best coefficient vector is found among the last-generation population

Genetic Algorithms – Optimization in Finance

In finance we find a lot of optimization problems: asset allocation, risk management, option pricing, etc.. The **genetic algorithm (GA)** is a **stochastic search technique** that mimics the process of natural evolution. This technique is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. Genetic algorithms (GAs) are **probabilistic optimizers**. Given a function, GAs implement a systematic search for the maximum of that function by exploring the domain where the function is defined. The searching process is driven by the generation of random numbers. The search is not random; it is guided by genetic principles.

One - max Problem – Max number of ones in a string

1 - Initial random population

Candidate	String	Fitness
C	00000110	2
B	11101110	6
C	00100000	1
D	00110100	3

2 - Crossover Applied

Initial Parent	Candidate B	Candidate C
	11101110	00100000
Resulting Child	Candidate E	Candidate F
	01101110	10100000

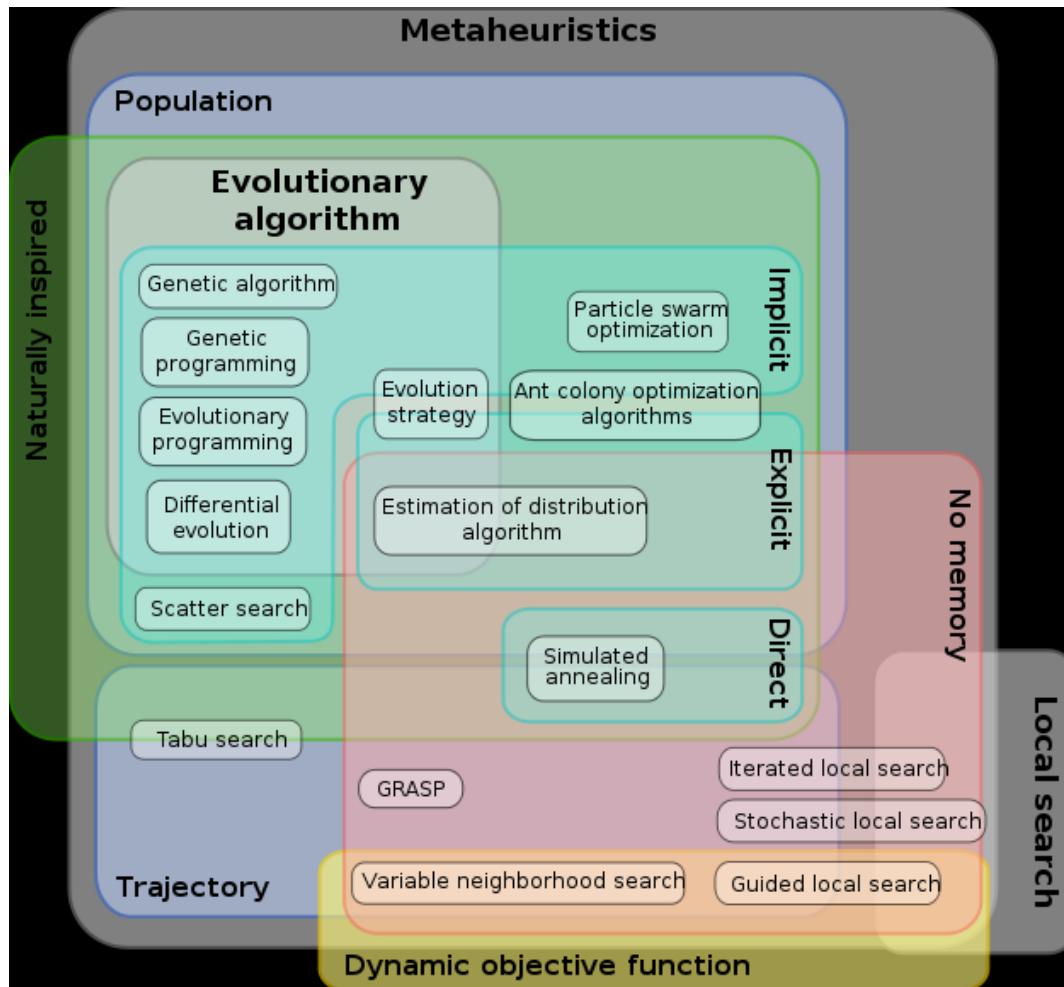
3 - No Crossover Applied

Initial Parent	Candidate B	Candidate D
	11101110	00110100
Resulting Child	Candidate G	Candidate H
	11101110	00110100

4 - Final new generation of solutions

Candidate	String	Fitness
E (Mutation)	01001110	4
F	10100000	2
G	11101110	6
H	00110100	3

Metaheuristics



Machine Learning Models

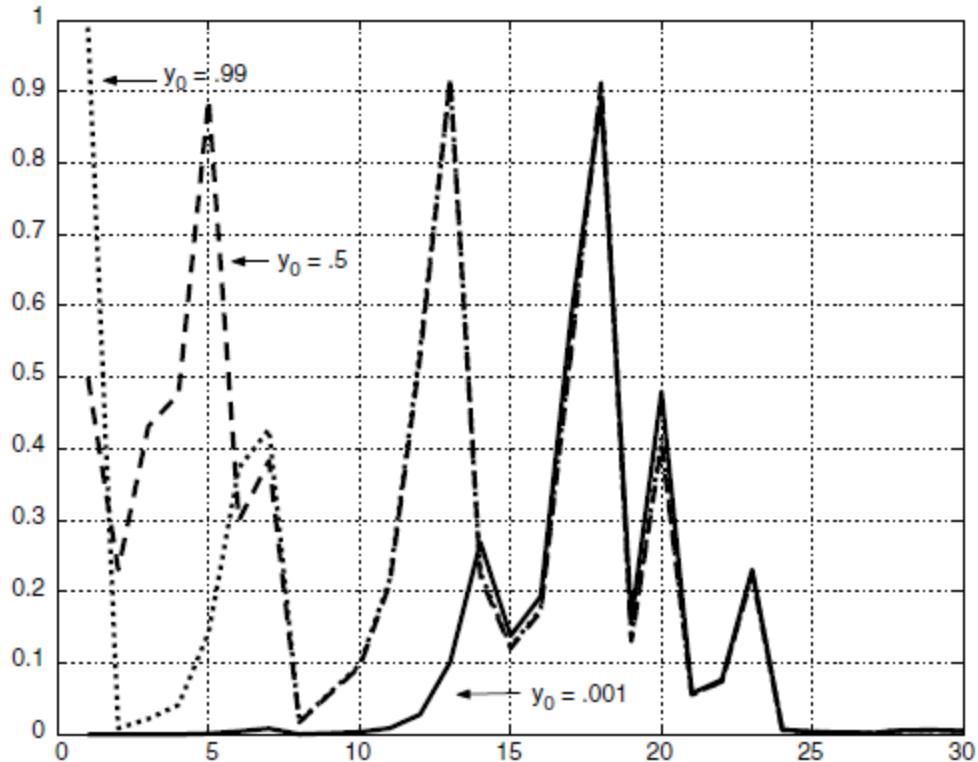
Neural Networks Applications

Neural Networks Applications

The first model we examine is the stochastic chaos (SC) model, the second is the stochastic volatility/jump diffusion (SVJD) model, the third is the Markov regime switching (MRS) model, the fourth is a volatility regime switching (VRS) model, the fifth is a distorted long-memory (DLM) model, and the last is the Black-Scholes options pricing (BSOP) model.

The SC model is widely used for testing predictive accuracy of various forecasting models, the SVJD and VRS models are commonly used models for representing volatile financial time series, and the MRS model is used for analyzing GDP growth rates. The DLM model may be used to represent an economy subject to recurring bubbles. Finally, the BSOP model is the benchmark model for calculating the arbitrage-free prices for options, under the assumption of the log normal distribution of asset returns.

Stochastic Chaos



$$y_t = 4 \cdot \varsigma_t \cdot y_{t-1} \cdot (1 - y_{t-1})$$

$$\varsigma_t \sim U(0, 1)$$

$$y_0 = .5$$

Diagnostic	Linear Model (Network Model) Estimate
R ²	.29 (.53)
HQIF	1534 (1349)
L-B*	.251
M-L*	.0001
E-N*	.0000
J-B*	.55
L-W-G	1000
B-D-S*	.0000

* marginal significance levels

Out of sample

The path of the out-of-sample prediction errors appears in last Figure. The solid path represents the forecast error of the linear model while the dotted curves are for the network forecast errors. This shows the improved performance of the network relative to the linear model, in the sense that its errors are usually closer to zero.

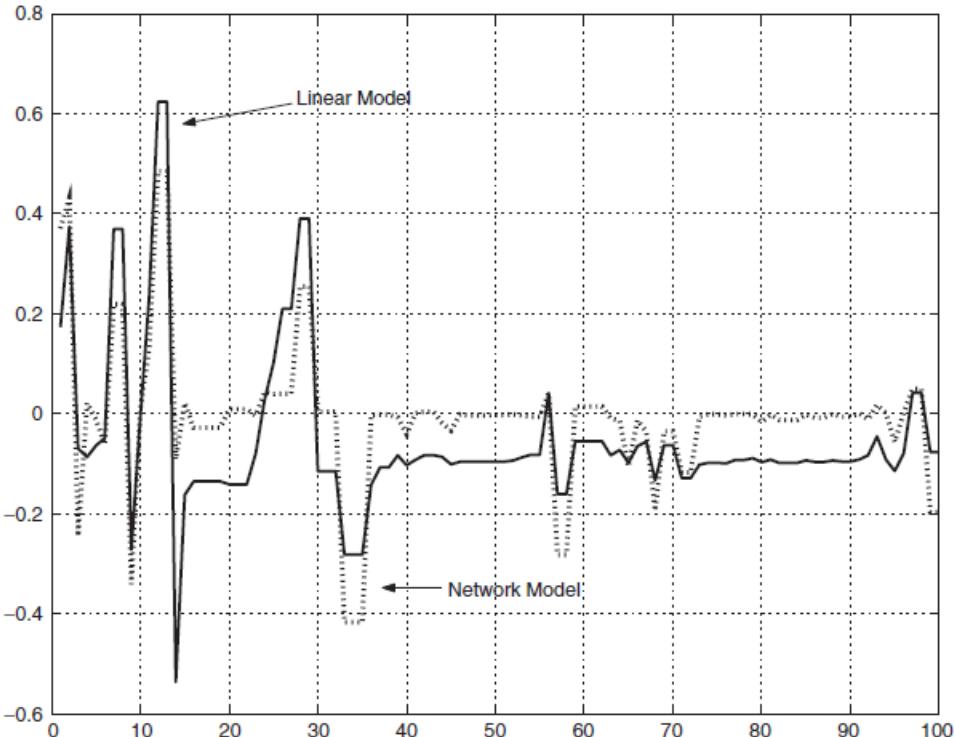
Table summarizes the out-of-sample statistics. These are the root mean squared error statistics (RMSQ), the Diebold-Mariano statistics for lags zero through four (DM-0 to DM-4), the success ratio for percentage of correct sign predictions (SR), and the bootstrap ratio (B-Ratio), which is the ratio of the network bootstrap error statistic to the linear bootstrap error measure. A value less than one, of course, represents a gain for network estimation.

The results show that the root mean squared error statistic of the network model is almost 20% lower than that of the linear model. Not surprisingly, the Diebold-Mariano tests with lags zero through four are all significant.

The success ratio for both models is perfect, since all of the returns in the stochastic chaos model are positive. The final statistic is the bootstrap ratio, the ratio of the network bootstrap error relative to the linear bootstrap error. We see that the network reduces the bootstrap error by almost 13%.

Clearly, if underlying data were generated by a stochastic process networks are to be preferred over linear models.

Stochastic Chaos – Out of Sample



Diagnostic	Linear	Neural Net
RMSQ	.147	.117
DM-0*	—	.000
DM-1*	—	.004e-5
DM-2*	—	.032e-5
DM-3*	—	.115e-5
DM-4*	—	.209e-5
SR	1	1
B-Ratio	—	.872

* marginal significance levels

Stochastic Volatility

$$\frac{dS}{S} = (\mu - \lambda \bar{k}) \cdot dt + \sqrt{V} \cdot dZ + k \cdot dq$$

$$dV = (\alpha - \beta V) \cdot dt + \sigma_v \sqrt{V} \cdot dZ_v$$

$$\text{Corr}(dZ, dZ_v) = \rho$$

$$\text{prob}(dq = 1) = \lambda \cdot dt$$

$$\ln(1 + k) \sim \phi(\ln[1 + \bar{k}] - .5\kappa, \kappa^2)$$

Diagnostic	Linear Model (Network Model) Estimate
R ²	.42 (.45)
HQIF	935 (920)
L-B*	.783
M-L*	.025
E-N*	.0008
J-B*	0
L-W-G	11
B-D-S*	.0000

* marginal significance levels

Mean return	μ	.21
Mean volatility	α	.0003
Mean reversion of volatility	β	.7024
Time interval (daily)	dt	1/250
Expected jump	\bar{k}	.3
Standard deviation of percentage jump	κ	.0281
Annual frequency of jumps	λ	2
Correlation of Weiner processes	ρ	.6

Out of Sample

Diagnostic	Linear	Neural Net
RMSQ	.157	.167
DM-0*	—	.81
DM-1*	—	.74
DM-2*	—	.73
DM-3*	—	.71
DM-4*	—	.71
SR	.646	.656
B-Ratio	—	.968

* marginal significance levels

Markov Switching Model GDP

$$x_t = c_c + \sum_{i=1}^p \phi_{1,i} x_{t-i} + \varepsilon_{1,i}, \quad \varepsilon_1 \sim \phi(0, \sigma_1^2), \text{ if } S = S^1$$

$$= c_2 + \sum_{i=1}^p \phi_{2,i} x_{t-i} + \varepsilon_{2,i}, \quad \varepsilon_2 \sim \phi(0, \sigma_2^2) \text{ if } S = S^2$$

$$\mathbf{P} = \begin{bmatrix} (S_t^1 | S_{t-1}^1) & (S_t^1 | S_{t-1}^2) \\ (S_t^2 | S_{t-1}^1) & (S_t^2 | S_{t-1}^2) \end{bmatrix} = \begin{bmatrix} (1 - w_2) & w_2 \\ w_1 & (1 - w_1) \end{bmatrix}$$

Diagnostic	Linear Model (Network Model) Estimate
R ²	.35 (.38)
HQIF	3291 (3268)
L-B*	.91
M-L*	.0009
E-N*	.0176
J-B*	.36
L-W-G	13
B-D-S*	.0002

* marginal significance levels

Parameter	State 1	State 2
c_i	.909	-.420
$\phi_{i,1}$.265	.216
$\phi_{i,2}$.029	.628
$\phi_{i,3}$	-.126	-.073
$\phi_{i,4}$	-.110	-.097
σ_i	.816	1.01
w_i	.118	.286

Out of Sample

Diagnostic	Linear	Neural Net
RMSQ	1.122	1.224
DM-0*	—	.27
DM-1*	—	.25
DM-2*	—	.15
DM-3*	—	.22
DM-4*	—	.24
SR	.77	.72
B-Ratio	—	.982

* marginal significance levels

Neural Networks On Artificial Data

The performance of alternative neural network models relative to the standard linear model for forecasting relatively complex artificially generated time series. We see that relatively simple feedforward neural nets outperform the linear models in some cases, or do not do worse than the linear models. In many cases we would be surprised if the neural networks did much better than the linear model, since the underlying data generating processes were almost linear.

The results of our investigation of these diverse stochastic experiments suggest that the real payoff from **neural networks will come from volatility forecasting rather than pure return forecasting in financial markets**, as we see in the high payoff from the implied volatility forecasting exercise with the Black-Sholes option pricing model. Since the neural networks never do appreciably worse than linear models, the only cost for using these methods is the higher computational time..

Machine Learning In Finance

5. Neural Networks Applications

German Credit Card Data

Variable	Definition	Type/Explanation	Max	Min	Median
1	Checking account	Categorical, 0 to 3	3	0	1
2	Term	Continuous	72	4	18
3	Credit history	Categorical, 0 to 4, from no history to delays	4	0	2
4	Purpose	Categorical, 0 to 9, based on type of purchase	10	0	2
5	Credit amount	Continuous	18424	250	2319.5
6	Savings account	Categorical, 0 to 4, lower to higher to unknown	4	0	1
7	Yrs in present employment	Categorical, 0 to 4, 1 unemployment, to longer years	4	0	2
8	Installment rate	Continuous	4	1	3
9	Personal status and gender	Categorical, 0 to 5, 1 male, divorced, 5 female, single	3	0	2
10	Other parties	Categorical, 0 to 2, none, 2 co-applicant, 3 guarantor	2	0	0
11	Yrs in present residence	Continuous	4	1	3
12	Property type	Categorical, 0 to 3, 0 real estate, 3 no property or unknown	3	0	2
13	Age	Continuous	75	19	33
14	Other installment plans	Categorical, 0 to 2, 0 bank, 1 stores, 2 none	2	0	0
15	Housing status	Categorical, 0 to 2, 0 rent, 1 own, 2 for free	2	0	2
16	Number of existing credits	Continuous	4	1	1
17	Job status	Categorical, 0 to 3, unemployed, 3 management	3	0	2
18	Number of dependents	Continuous	2	1	1
19	Telephone	Categorical, 0 to 1, 0 none, 1 yes, under customer name	1	0	0
20	Foreign worker	Categorical, 0 to 1, 0 yes, 1 no	1	0	0

German Credit Card Data

Method	False Positives	False Negatives	Weighted Average
Discriminant analysis	0.000	0.763	0.382
Neural network	0.095	0.196	0.146
Logit	0.095	0.196	0.146
Probit	0.702	0.003	0.352
Weibull	0.708	0.000	0.354

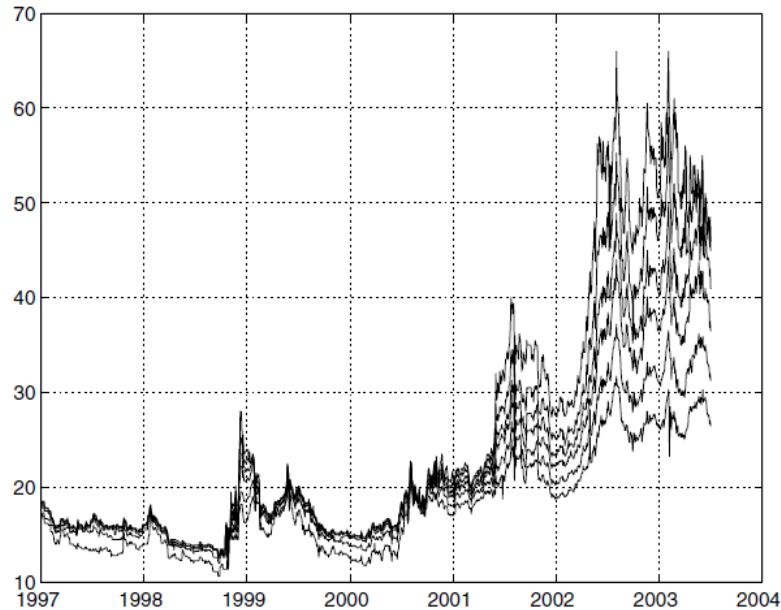
German Credit Card Data

Variable Definition	Partial Derivatives*				Prob Values**			
	Network	Logit	Probit	Weibull	Network	Logit	Probit	Weibull
1 Checking account	0.074	0.074	0.076	0.083	0.000	0.000	0.000	0.000
2 Term	0.004	0.004	0.004	0.004	0.000	0.000	0.000	0.000
3 Credit history	-0.078	-0.078	-0.077	-0.076	0.000	0.000	0.000	0.000
4 Propose	-0.007	-0.007	-0.007	-0.007	0.000	0.000	0.000	0.000
5 Credit amount	0.000	0.000	0.000	0.000	0.150	0.150	0.152	0.000
6 Savings account	-0.008	-0.008	-0.009	-0.010	0.020	0.020	0.020	0.050
7 Yrs in present employment	-0.032	-0.032	-0.031	-0.030	0.000	0.000	0.000	0.000
8 Installment rate	0.053	0.053	0.053	0.049	0.000	0.000	0.000	0.000
9 Personal status and gender	-0.052	-0.052	-0.051	-0.047	0.000	0.000	0.000	0.000
10 Other parties	-0.029	-0.029	-0.026	-0.020	0.010	0.010	0.020	0.040
11 Yrs in present residence	0.008	0.008	0.008	0.004	0.050	0.050	0.040	0.060
12 Property type	-0.002	-0.002	-0.000	0.003	0.260	0.260	0.263	0.300
13 Age	-0.003	-0.003	-0.003	-0.002	0.000	0.000	0.000	0.010
14 Other installment plans	0.057	0.057	0.062	0.073	0.000	0.000	0.000	0.000
15 Housing status	-0.047	-0.047	-0.050	-0.051	0.000	0.000	0.000	0.000
16 Number of existing credits	0.057	0.057	0.055	0.053	0.000	0.000	0.000	0.000
17 Job status	0.003	0.003	0.006	0.012	0.920	0.920	0.232	0.210
18 Number of dependents	0.032	0.032	0.030	0.022	0.710	0.710	0.717	0.030
19 Telephone	-0.064	-0.064	-0.065	-0.067	0.000	0.000	0.000	0.000
20 Foreign worker	-0.165	-0.165	-0.153	-0.135	0.000	0.000	0.000	0.000

*: Derivatives calculated as finite differences

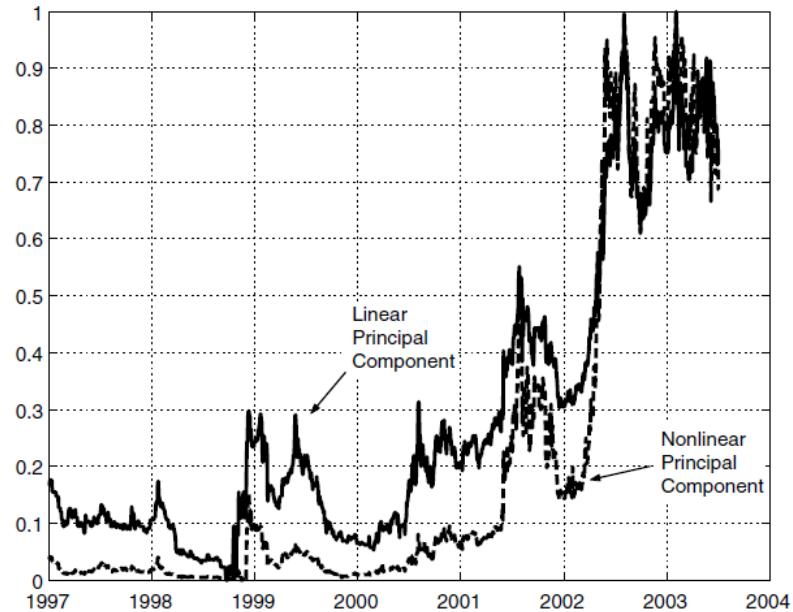
**: Prob values calculated from bootstrap distributions

Implied Volatilities US



Statistic	Maturity in Years					
	2	3	4	5	7	10
Mean	24.746	23.864	22.799	21.866	20.360	18.891
Median	17.870	18.500	18.900	19.000	18.500	17.600
Std. Dev.	14.621	11.925	9.758	8.137	6.106	4.506
Coeff. Var	0.591	0.500	0.428	0.372	0.300	0.239
Skewness	1.122	1.214	1.223	1.191	1.092	0.952
Kurtosis	2.867	3.114	3.186	3.156	3.023	2.831
Max	66.000	59.000	50.000	44.300	37.200	31.700
Min	10.600	12.000	12.500	12.875	12.750	12.600

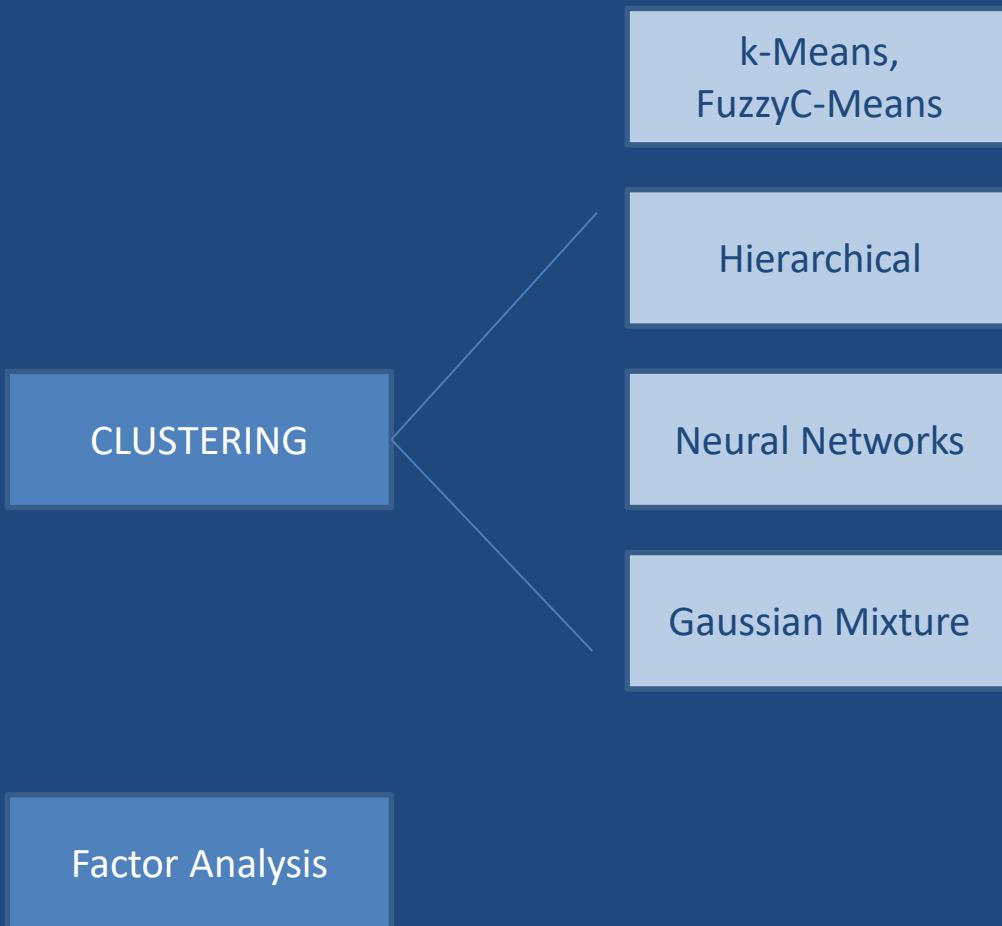
	Maturity in Years					
	2	3	4	5	7	10
Linear	0.983	0.995	0.997	0.998	0.994	0.978
Nonlinear	0.995	0.989	0.984	0.982	0.977	0.969



Machine Learning In Finance

5. Unsupervised Learning

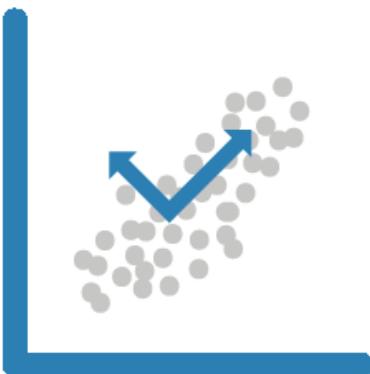
UNSUPERVISED LEARNING : CLUSTERING OVERVIEW



UNSUPERVISED LEARNING : CLUSTERING OVERVIEW

PCA

Principal component analysis (PCA)—performs a linear transformation on the data so that most of the variance or information in your high-dimensional dataset is captured by the first few principal components. The first principal component will capture the most variance, followed by the second principal component, and so on.



k-Means

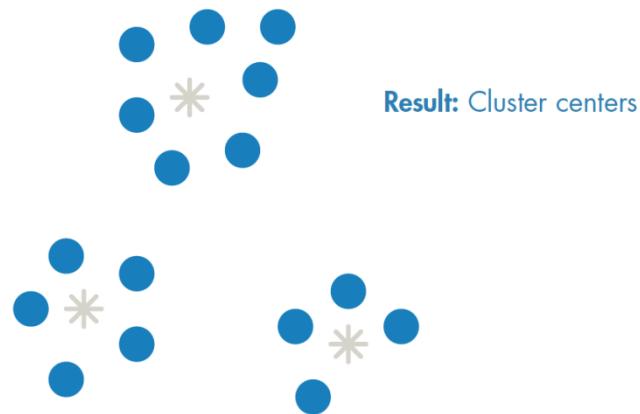
How it Works

Partitions data into k number of mutually exclusive clusters.

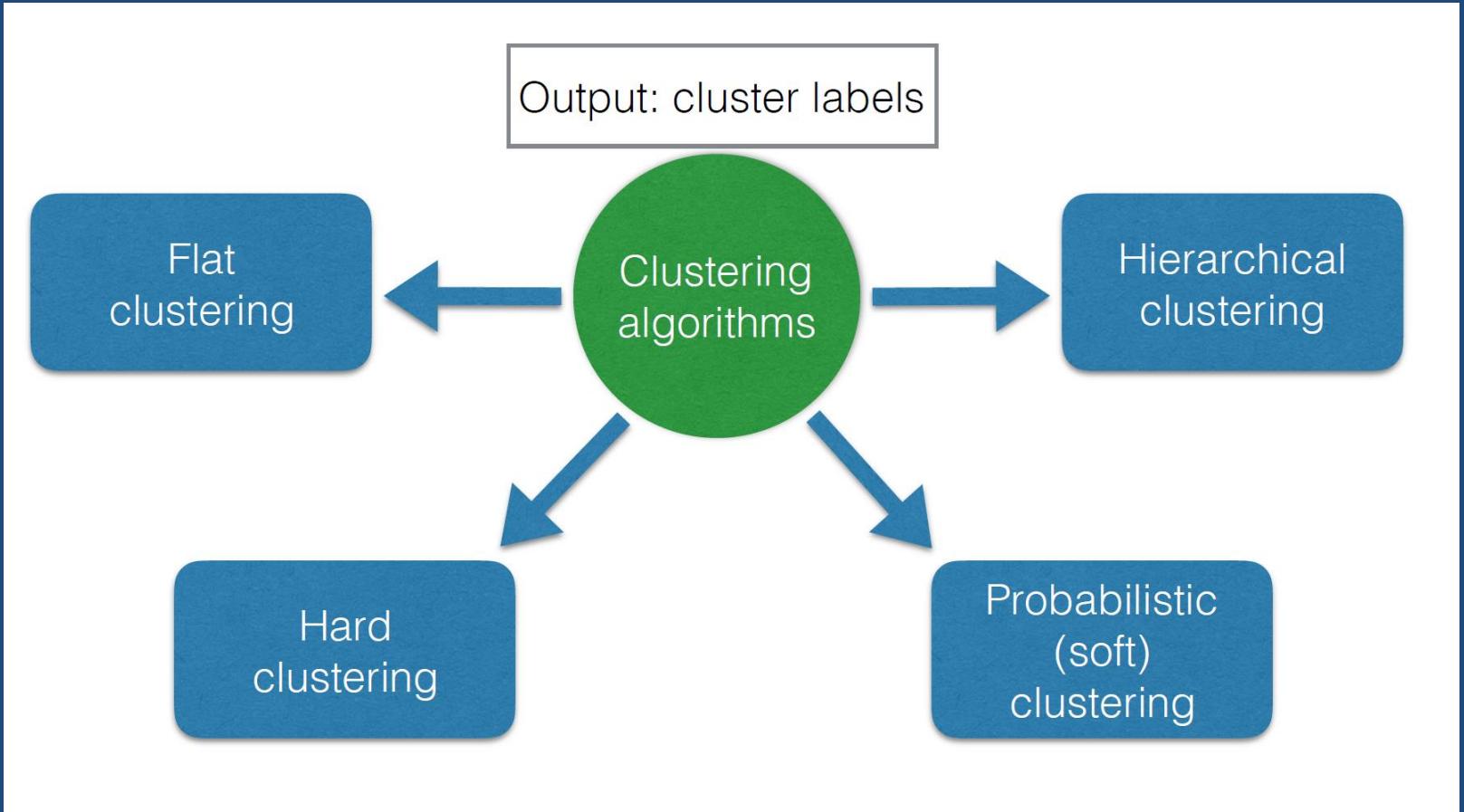
How well a point fits into a cluster is determined by the distance from that point to the cluster's center.

Best Used...

- When the number of clusters is known
- For fast clustering of large data sets



UNSUPERVISED LEARNING : CLUSTERING OVERVIEW



Unsupervised Learning in Finance

Unsupervised learning algorithms examine the dataset and identify relationships between variables and their common drivers. In unsupervised learning the machine is simply given the entire set of returns of assets and it does not have a notion of what are independent and what are the dependent variables. Methods of unsupervised learning are often categorized as either [Clustering](#) or [Factor analyses](#).

[Clustering](#) involves splitting a dataset into smaller groups based on some notion of similarity. In finance that may involve identifying historical regimes such as high/low volatility regime, rising/falling rates regime, rising/falling inflation regime, etc. Correctly identifying the regime can in turn be of high importance for allocation between different assets and risk premia.

[Factor analysis'](#) aim to identify the main drivers of the data or identify best representation of the data. For instance, yield curve movements may be described by parallel shift of yields, steepening of the curve, and convexity of the curve. In a multi-asset portfolio, factor analysis will identify the main drivers such as momentum, value, carry, volatility, liquidity, etc. A very well-known method of factor analysis is Principal Component Analysis (PCA). The PCA method carries over from the field of statistics to unsupervised Machine Learning without any changes.

Machine Learning In Finance

Unsupervised Learning

5.1 Clustering

Unsupervised Learning in Finance - Algorithms

- K-means – Simplest clustering algorithm that starts by initially marking random points as exemplars. It iteratively does a two-step calculation: in the first step, it maps points to closest exemplar; in the second step it redefines the exemplar as the mean of the points mapped to it. It locates a fixed number of clusters .
- Ward – A hierarchical clustering technique that is similar to K-means, except that it uses a decision tree to cluster points.
- Birch – A hierarchical clustering technique designed for very large databases. It can incrementally cluster streaming data; hence in many cases, it can cluster with a single pass over the data set.
- Affinity Propagation – Algorithm involves passing of ‘soft’ information that treats every point as a possible exemplar and allows for the possibility of every point being included in a cluster around it. The algorithm is known to be good for finding a large number of small clusters. The number of clusters chosen can be indirectly influenced via a ‘preference’ parameter.

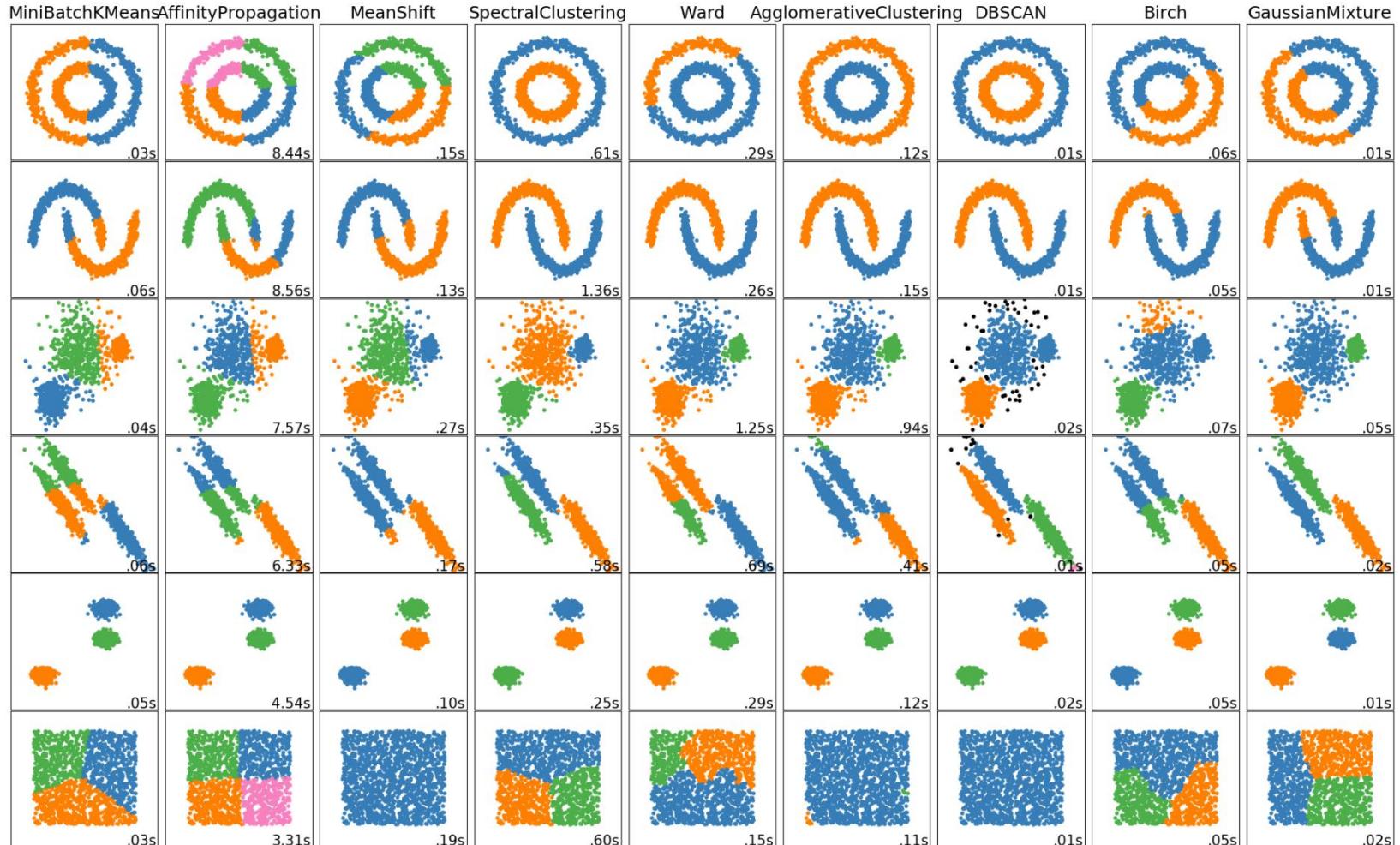
Unsupervised Learning in Finance - Algorithms

- **Spectral Clustering** – Like affinity propagation, it passes messages between points, but does not identify the exemplar of each cluster. It needs to be told the number of clusters to locate (we have specified two). This is understandable, since the algorithm computes an affinity matrix between samples, embeds into a low-dimensional space and then runs K-means to locate the clusters. This is known to work well for a small number of clusters.
- **Mini-Batch** – To reduce processing time, mini-batches of data are processed with a k-means algorithm. However unlike kmeans the cluster centroids are only updated with each new batch, rather than each new point. Mini-Batch typically converges faster than K-Means, but the quality can be slightly lower.
- **HDBSCAN** – Hierarchical Density-Based Spatial Clustering of Applications with Noise is a ‘soft’ clustering algorithm where points are assigned to clusters with a probability and outliers are excluded. HDBSCAN runs multiple DBScans over the data seeking the highest cluster stability over various parameters.

Unsupervised Learning in Finance - Algorithms

- ICA – Independent Component Analysis is similar to Principal Component Analysis (PCA) but is better at distinguishing non-Gaussian signals. While these techniques are typically used in factor extraction and dimension reduction, we have used the exposures to these common (hidden) factors to form clusters to explore the potential of these components to identify similarity between members' returns.
- Agglomerative Clustering – This is another hierarchical clustering technique that starts with each point as exemplar and then merges points to form clusters. Unlike Ward which looks at the sum of squared distance within each cluster, this looks at the average distance between all observations of pairs of clusters.
- HDBScan – This forms clusters with a roughly similar density of points around them. Points in low-density regions are treated as outliers. It can potentially scan the data set multiple times before converging on the clusters.

Unsupervised Learning in Finance - Clustering



Unsupervised Learning in Finance - Clustering

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
<i>K-Means</i>	number of clusters	Very large n_samples, medium n_clusters with <i>MiniBatch code</i>	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
<i>Affinity propagation</i>	damping, sample preference	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
<i>Mean-shift</i>	bandwidth	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Distances between points
<i>Spectral clustering</i>	number of clusters	Medium n_samples, small n_clusters	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
<i>Ward hierarchical clustering</i>	number of clusters	Large n_samples and n_clusters	Many clusters, possibly connectivity constraints	Distances between points
<i>Agglomerative clustering</i>	number of clusters, linkage type, distance	Large n_samples and n_clusters	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
<i>DBSCAN</i>	neighborhood size	Very large n_samples, medium n_clusters	Non-flat geometry, uneven cluster sizes	Distances between nearest points
<i>Gaussian mixtures</i>	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers
<i>Birch</i>	branching factor, threshold, optional global clusterer.	Large n_clusters and n_samples	Large dataset, outlier removal, data reduction.	Euclidean distance between points

Clustering – Application JPMorgan

Metric	Name	Notes: Most scores range -1 or 0 to + 1
ARI	Adjusted Rand Index	See detailed breakout box below
AS	Accuracy Score	Subsets must exactly match
PS	Precision Score	Ratio of True Positives to TP + False Positives
F1	F1 Score (AKA F-measure)	Precision (TP/TP+FP) to Recall (TP/TP+FN) $F1=2P*R/P+R$
HS	Homogeneity Score	Each cluster contains only members of a single class
CS	Completeness Score	All members of a given class are assigned the same cluster
HCV	Homogeneity Completeness V-Measure	Average of HS and CS
HL^	Hamming Loss	Fraction of labels that are incorrectly predicted, 0 is best to 1
JS	Jaccard Similarity Score	Size of the intersection divided by the size of the union
MI*	Mutual Information	Not standardized, max value unbounded
A_MI	Adjusted Mutual Information Score	Adjusted for chance, perfect = 1, chance = 0, can be negative
Z_MI	Normalized Mutual Information Score	Normalization of MI to be between 0 and 1
Avg^	Average Rank (Ascending)	All metrics are ranked, then averaged.

Clustering – Application JPMorgan

Method \ Score	ARI	F1	Z_MI	A_MI	HS	CS	HCV	AS	PS	JS	MI*	HL^	Avg^
Birch (2D PCA)	0.142	0.008	0.656	0.277	0.712	0.604	0.654	0.008	0.028	0.008	3.054	0.992	1.8
Ward	0.120	0.003	0.648	0.260	0.704	0.596	0.645	0.005	0.004	0.005	3.018	0.995	3.7
Affinity	0.124	0.003	0.619	0.275	0.653	0.586	0.618	0.005	0.004	0.005	2.800	0.995	4.0
Spectral	0.031	0.011	0.584	0.217	0.599	0.569	0.584	0.011	0.044	0.011	2.569	0.989	4.4
MiniBatch	0.123	0.003	0.616	0.265	0.651	0.583	0.615	0.005	0.003	0.005	2.790	0.995	4.9
Agglomerative	0.129	0.001	0.645	0.259	0.700	0.594	0.643	0.002	0.000	0.002	3.002	0.998	5.0
ICA	0.050	0.004	0.539	0.134	0.569	0.510	0.538	0.003	0.057	0.003	2.441	0.997	5.9
HDBSCAN	0.013	0.004	0.508	0.043	0.503	0.513	0.508	0.002	0.012	0.002	2.157	0.998	6.9
HRP	0.003	0.000	0.314	0.017	0.175	0.562	0.267	0.002	0.000	0.002	0.749	0.998	8.4

Source: MSCI, FactSet, J.P. Morgan. * indicates score unbounded, all other scores/indices are bounded between -1 or 0 and +1. Hamming Loss is a loss function, so smaller values are preferred

We found the Birch, Ward and Affinity Propagation techniques return clusters that most closely match MSCI Country GICS Sectors, while HRP, HDBScan and ICA are the most different (note that we are not suggesting that these scores should be closer to 1; the goal of the clustering algorithms was not to match GICS country sectors but we are using GICS as a comparison).

Machine Learning In Finance

5.2 PCA Examples

Eigenvalues, Eigenvectors and Eigenportfolios

Principal component analysis is a vector space transform used to reduce multidimensional data sets to lower dimensions. New variables are created as a linear combination of the original data. Each of the new variables (principal components) are constructed to be uncorrelated (orthogonal) with all others. Each successive principal components explains a decreasing amount of variation in the dataset.

The coefficients of the principal components are calculated so that the first principal component contains the maximum variance ('the market portfolio'). The second principal component is calculated to have the second most variance but it is uncorrelated with the first principal component ('the market neutral portfolio with max variance'). The further portfolios are all 'market-neutral' but with decreasing variance and high noise-to-signal ratio.

PCA is realized with a eigendecomposition on a covariance matrix or a correlation matrix of assets values (standardized). We will get eigenvectors (factors loadings) and eigenvalues (factor variances). The eigenvectors represents asset weights towards each principal component portfolio.

The total number of principal portfolios equals to the number of principal components.

The variance of each principal portfolio is its corresponding eigenvalue. The eigenvectors have values ranging in [-1,+1] but in order to create a long-short portolio we need to rescale factor loadings.

PCA Decomposition – Covariance Matrix

First of all we estimate the 9×9 covariance matrix

$$\text{Cov}\{\mathbf{X}\} = \mathbf{E}\boldsymbol{\Lambda}\mathbf{E}'$$

In this expression $\boldsymbol{\Lambda}$ is the diagonal matrix of the $N \equiv 9$ eigenvalues of the covariance sorted in decreasing order:

$$\boldsymbol{\Lambda} \equiv \text{diag}(\lambda_1, \dots, \lambda_9)$$

and the matrix \mathbf{E} is the juxtaposition of the respective eigenvectors and represents a rotation:

$$\mathbf{E} \equiv (e^{(1)}, \dots, e^{(9)})$$

Principal Portfolios

Real-life portfolios are usually characterized by non-zero covariances σ_{ij} for at least some i and j . The idea of a principal portfolio is to find a change of base of the original asset space that generates a new set of synthetic assets with zero covariances $s_{\mu\nu}$ for all combinations of μ and ν . We know from basic linear algebra that any real-valued symmetric $N \times N$ matrix is orthogonally diagonalizable and possesses a complete set of orthogonal eigenvectors. If the matrix is positive semi-definite, all eigenvalues will be non-negative.

The eigenvectors equations for the covariance matrix Ω are given by

$$\Omega e^\mu = \lambda_\mu e^\mu \text{ for } \mu = 1, \dots, N$$

where e denotes the N -dimensional eigenvector and λ the scalar eigenvalue.

We can now define a N -dimensional square matrix $E \equiv \{e^1, \dots, e^N\}$ whose columns are the N eigenvectors e^μ . With $\Lambda \equiv \text{diag}\{\lambda_1, \dots, \lambda_N\}$ and \circ symbolizing the Hadamard product, the following relations hold:

Principal Portfolios

$$\mathbf{E}^{-1}\boldsymbol{\Omega}\mathbf{E} = \boldsymbol{\Lambda},$$

$$\begin{bmatrix} \sigma_1^2 \\ \sigma_2^2 \\ \vdots \\ \sigma_N^2 \end{bmatrix} = (\mathbf{E} \circ (\mathbf{E}^{-1})') \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \end{bmatrix}$$

Eigenvectors are only defined up to a non-zero scalar multiple. It is common practice to standardize the eigenvectors to unit length so that $(\mathbf{e}^\mu)' \mathbf{e}^\nu = \delta_{\mu\nu}$ where $\delta_{\mu\nu}$ is the Kronecker delta. With this standardization, the matrix of eigenvectors \mathbf{E} is said to be orthonormal. In comparison to an orthogonal matrix, \mathbf{E} then has the additional characteristics that $\mathbf{E}^{-1} = \mathbf{E}'$ and the sum of squares of any row and column is equal to unity (Jackson, 2005) With an orthonormal base, it follows from the equation that each asset's variance equals a weighted sum of the eigenvalues:

$$\sigma_i^2 = \sum_{\mu=1}^N \lambda_\mu (e_i^\mu)^2, \quad \sum_{\mu=1}^N (e_i^\mu)^2 = 1 \text{ for all } i.$$

Principal Portfolios

We now define a new N-dimensional column vector \mathbf{p} as

$$\mathbf{p} \equiv \mathbf{E}^{-1}\mathbf{w}$$

Substituting \mathbf{w} by \mathbf{Ep} in equation

$$\sigma(\mathbf{w}) = \sqrt{\mathbf{w}'\Omega\mathbf{w}} = \sqrt{\sum_i w_i^2 \sigma_i^2 + \sum_i \sum_{j \neq i} w_i w_j \rho_{ij} \sigma_i \sigma_j}.$$

We get

$$\sigma^2 = (\mathbf{Ep})'\Omega\mathbf{Ep} = \mathbf{p}'\mathbf{E}'\Omega\mathbf{E}\mathbf{p} = \mathbf{p}'\Lambda\mathbf{p}.$$

Principal Portfolios

Portfolio variance can either be described by the quadratic form $\mathbf{w}'\Omega\mathbf{w}$ or the quadratic form $\mathbf{p}'\Lambda\mathbf{p}$. We can use the transformations shown in previous equations to translate the original system characterized by a weights vector w and a covariance matrix Ω into a principal system characterized by a principal weights vector p and a diagonal covariance matrix of the principal portfolios Λ .

This change of base simplifies the covariance structure considerably.

On the other hand, much of the complexity of Ω is now shifted into the principal weights vector p whose elements can be negative and will in general not sum to one.

	Assets	Principals
Weights and indices	w_i, w_j	p_μ, p_ν
Portfolio variance	$\sigma^2 = \sum_i \sum_j w_i w_j \rho_{i,j} \sigma_i \sigma_j$	$s^2 = \sum_\mu p_\mu^2 s_\mu^2$
Marginal contribution to risk	$\frac{w_i \sigma_i^2 + \sum_{j \neq i} w_j \rho_{ij} \sigma_i \sigma_j}{\sigma}$	$\frac{p_\mu s_\mu^2}{s}$
Total contribution to risk	$\frac{w_i^2 \sigma_i^2 + \sum_{j \neq i} w_i w_j \rho_{ij} \sigma_i \sigma_j}{\sigma}$	$\frac{p_\mu^2 s_\mu^2}{s}$
Percentage contribution to risk	$\frac{w_i^2 \sigma_i^2 + \sum_{j \neq i} w_i w_j \rho_{ij} \sigma_i \sigma_j}{\sigma^2}$	$\frac{p_\mu^2 s_\mu^2}{s^2}$

Principal Portfolios Multi Asset Results – C Kind

Short	Future name	Class	Exchange	Return	Risk	Sharpe
US	US LONG BOND	Bond	CBT	2.37%	10.28%	0.23
TY	US 10YR NOTE	Bond	CBT	1.54%	6.67%	0.23
SP	S&P 500	Equity	CME	7.72%	15.03%	0.51
Z	FTSE 100	Equity	LIF	5.36%	14.81%	0.36
C	CORN	Commodity	CBT	8.99%	28.22%	0.32
QS	GAS OIL	Commodity	ICE	13.16%	33.06%	0.40
JY	JPYUSD	Currency	CME	2.60%	11.21%	0.23
CD	CANUSD	Currency	CME	1.08%	7.71%	0.14

	US	TY	SP	Z	C	QS	JY	CD
US	1.00	0.95	-0.08	-0.09	0.01	-0.15	0.16	-0.10
TY	0.95	1.00	-0.08	-0.10	0.02	-0.12	0.21	-0.08
SP	-0.08	-0.08	1.00	0.79	0.25	0.04	0.00	0.51
Z	-0.09	-0.10	0.79	1.00	0.19	0.04	-0.04	0.39
C	0.01	0.02	0.25	0.19	1.00	0.02	0.04	0.20
QS	-0.15	-0.12	0.04	0.04	0.02	1.00	0.05	0.29
JY	0.16	0.21	0.00	-0.04	0.04	0.05	1.00	0.01
CD	-0.10	-0.08	0.51	0.39	0.20	0.29	0.01	1.00

Principal Portfolios Multi Asset Results – C Kind

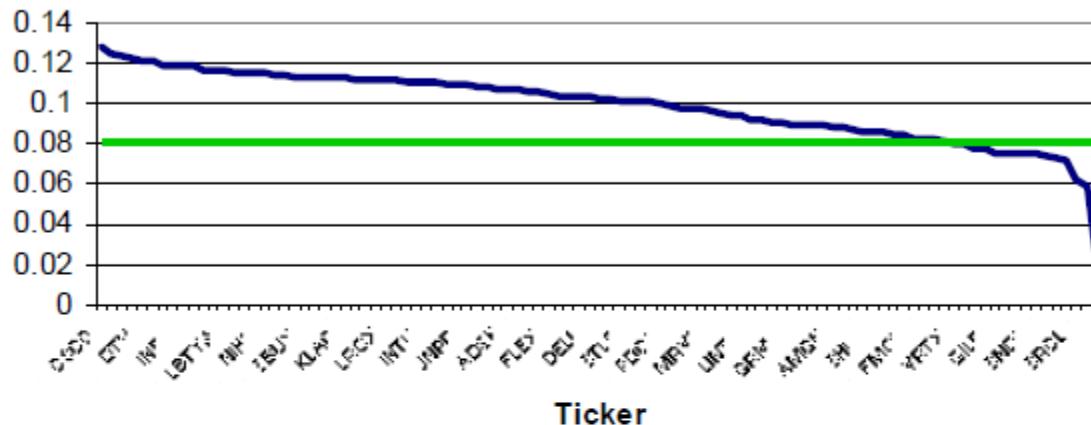
	Assets		Principals				
	EW	RP	PRP	ME	PRPLS	MELS	PRB
Return (%)	5.75	3.61	3.75	3.97	2.37	0.17	6.97
Risk (%)	8.44	6.07	6.49	6.98	5.70	7.47	10.84
Sharpe	0.68	0.59	0.58	0.57	0.42	0.02	0.64
MDD (%)	28.3	17.5	19.5	19.7	11.4	21.4	22.8
VaR (95%)	-3.5	-2.7	-2.7	-3.0	-2.6	-3.6	-4.1
CVaR (95%)	-5.3	-3.8	-4.2	-4.4	-3.6	-5.4	-6.4
Turnover (%)	0.40	1.51	9.49	10.47	10.83	35.72	9.20

Results of PCA for Nasdaq 100

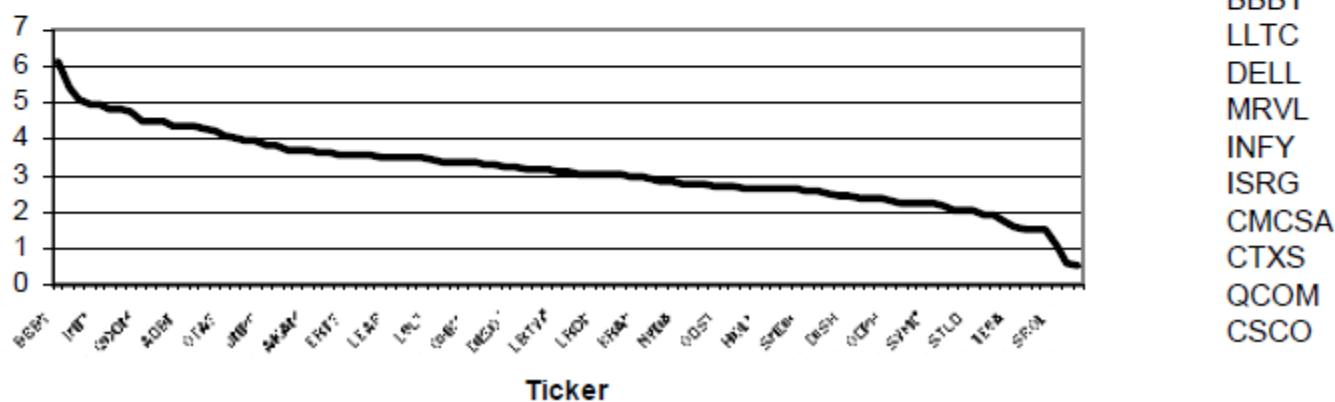
- 4 significant eigenvectors/eigenvalues
- -- first Eigen-state explains about 44% of the correlation
- -- total explained variance= 51%
- -- Now we need to identify the eigenportfolios in terms of real market factors (industry, size, etc, etc).

First Eigenvector: Market

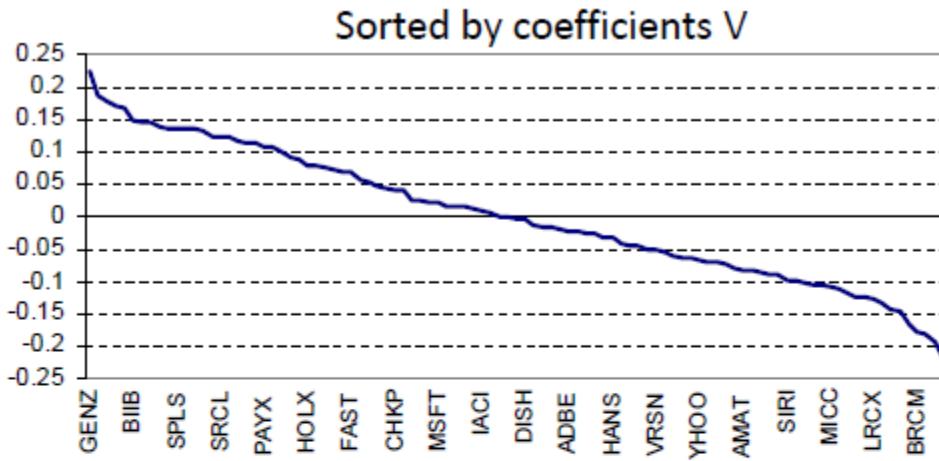
Sorted Eigenvector



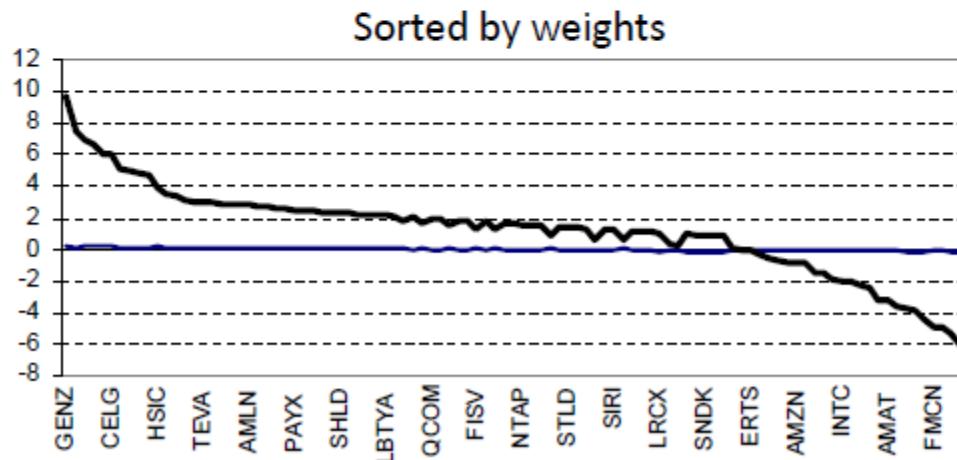
Sorted Weights



Second Eigenvector: Biotech vs Chips

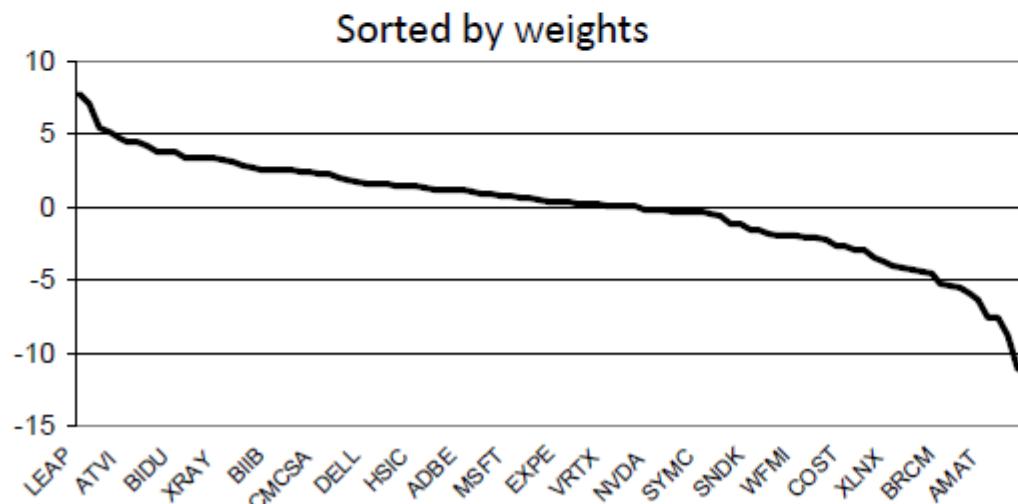
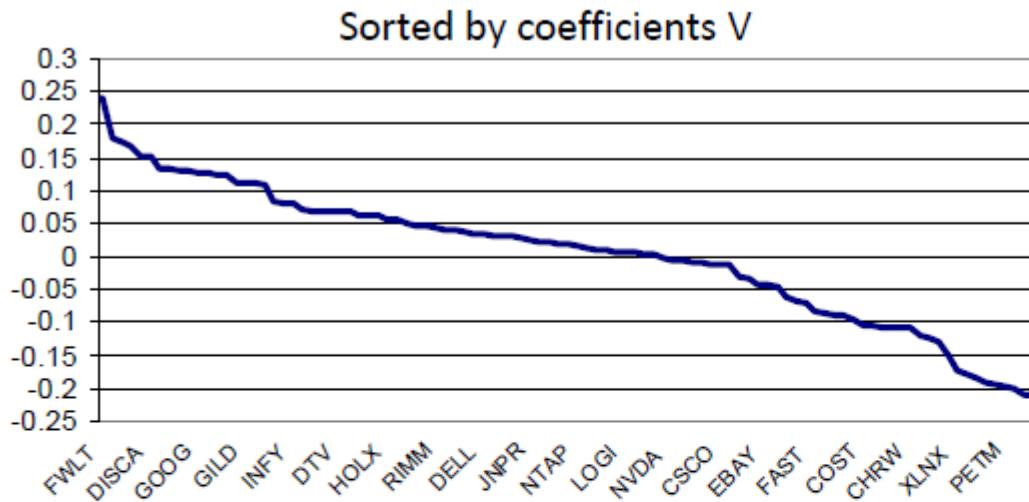


Top 10	Bottom 10
GENZ	MRVL
CEPH	NVDA
HSIC	FWLT
CELG	BRCM
GILD	SNDK
BIIB	JOYG
XRAY	RIMM
AMLN	BIDU
CTAS	LRCX
ESRX	ALTR



Top 10	Bottom 10
GENZ	BRCM
BBBY	FWLT
BIIB	DELL
GILD	FMCN
CEPH	AKAM
CELG	BIDU
ESRX	ALTR
CTAS	FLEX
AMGN	AMAT

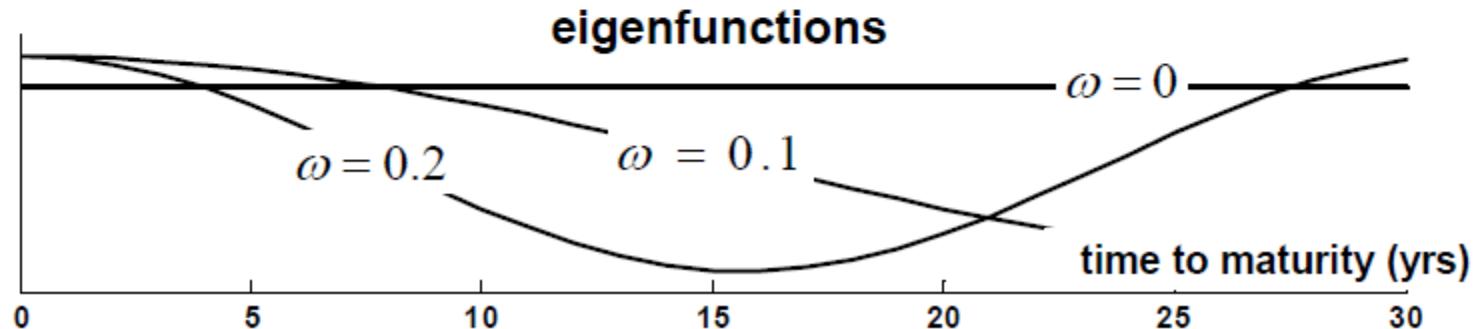
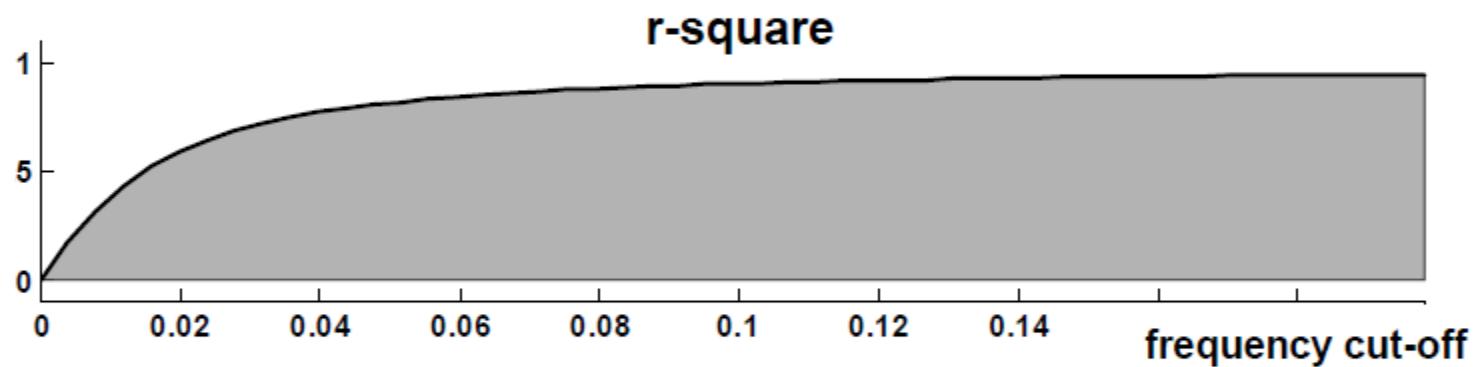
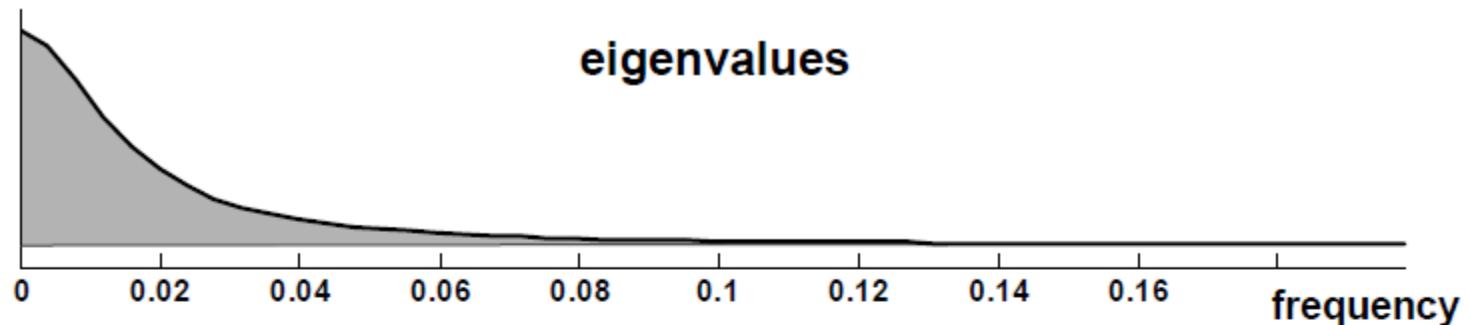
Third Eigenvector: Manufacturing vs Chips



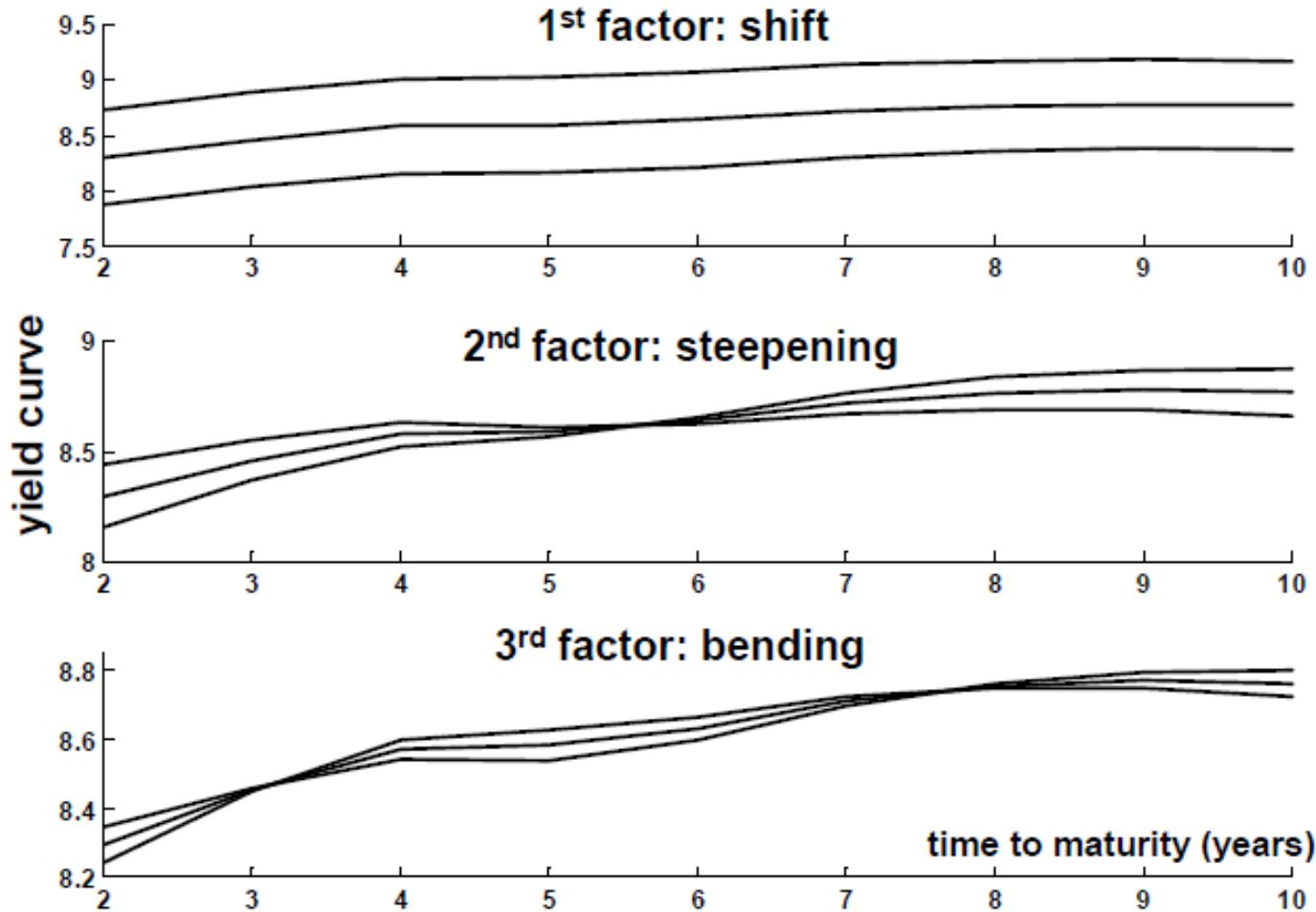
Top 10	Bottom 10
FWLT	KLAC
LEAP	ALTR
JOYG	BBBY
STLD	PETM
TEVA	LRCX
DISCA	AMAT
DISH	LLTC
CEPH	SHLD
ATVI	XLNX
LBTYA	BRCM

Top 10	Bottom 10
LEAP	BBBY
FWLT	LLTC
DISCA	SHLD
FMCN	AMAT
NIHD	ALTR
ATVI	PETM
GILD	MRVL
CEPH	LRCX
GOOG	BRCM
JOYG	KLAC

PCA Swap Rates

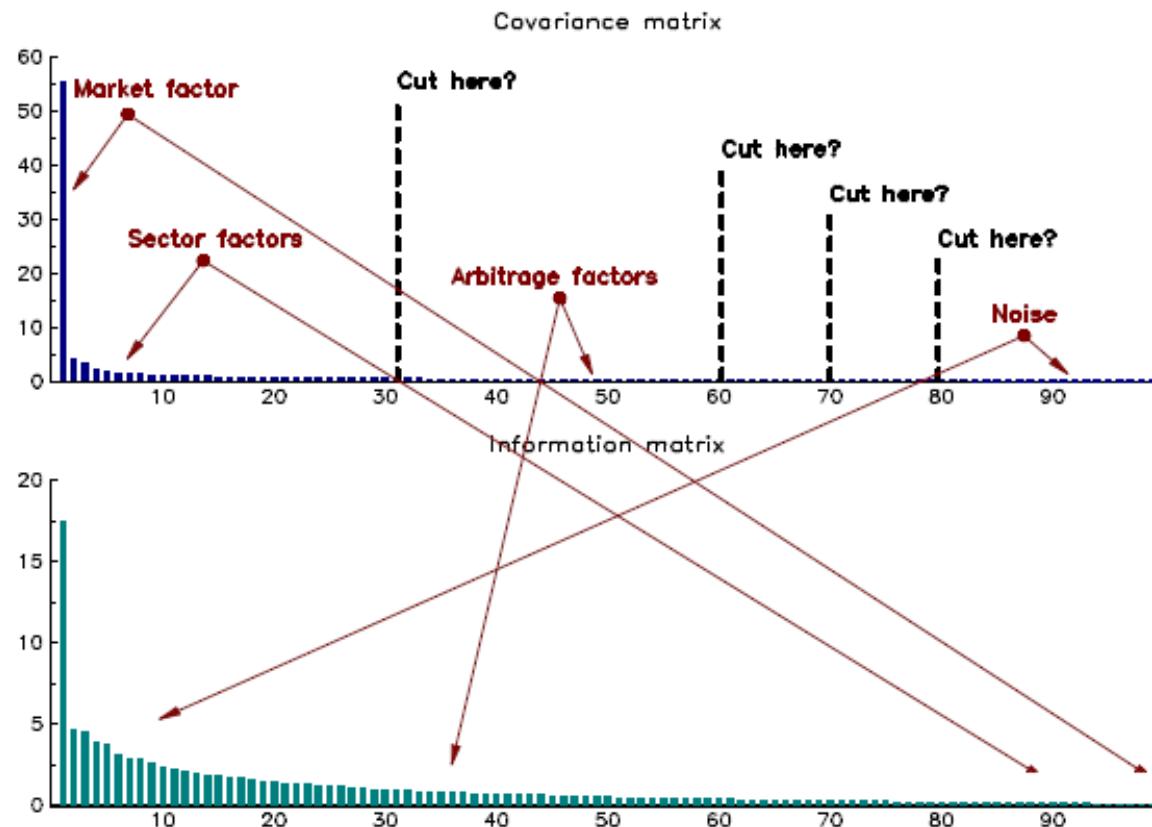


PCA Swap Rates



PCA – Filtering Factors

Figure: Eigendecomposition of the FTSE 100 covariance matrix



Machine Learning In Finance

6. Deep Learning

Neural Networks

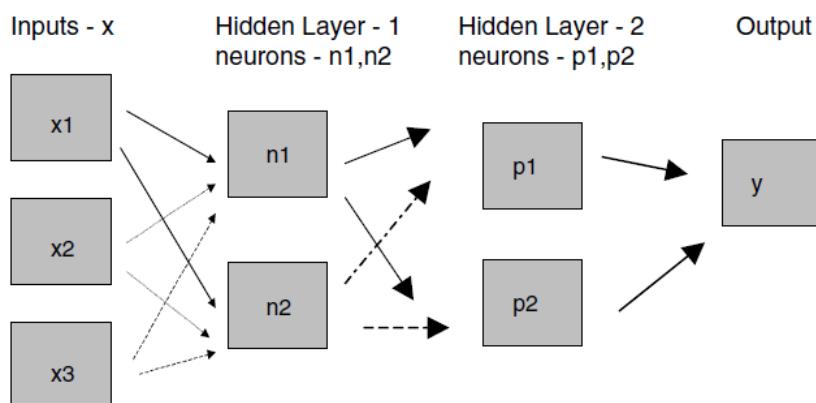
Neural Networks

How it Works

Inspired by the human brain, a neural network consists of highly connected networks of neurons that relate the inputs to the desired outputs. The network is trained by iteratively modifying the strengths of the connections so that given inputs map to the correct response.

Best Used...

- For modeling highly nonlinear systems
- When data is available incrementally and you wish to constantly update the model
- When there could be unexpected changes in your input data
- When model interpretability is not a key concern



$$n_{k,t} = w_{k,0} + \sum_{i=1}^{i^*} w_{k,i} x_{i,t}$$

$$N_{k,t} = \frac{1}{1 + e^{-n_{k,t}}}$$

$$p_{l,t} = \rho_{l,0} + \sum_{k=1}^{k^*} w_{l,k} N_{k,t}$$

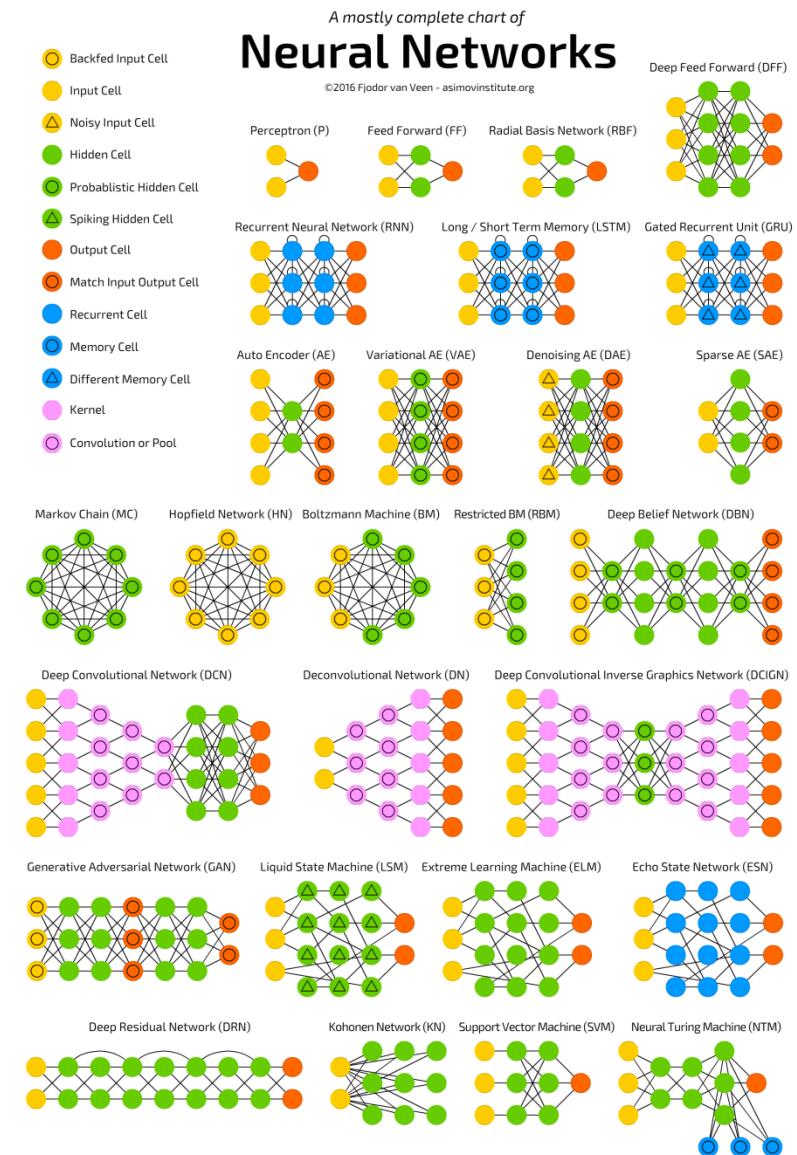
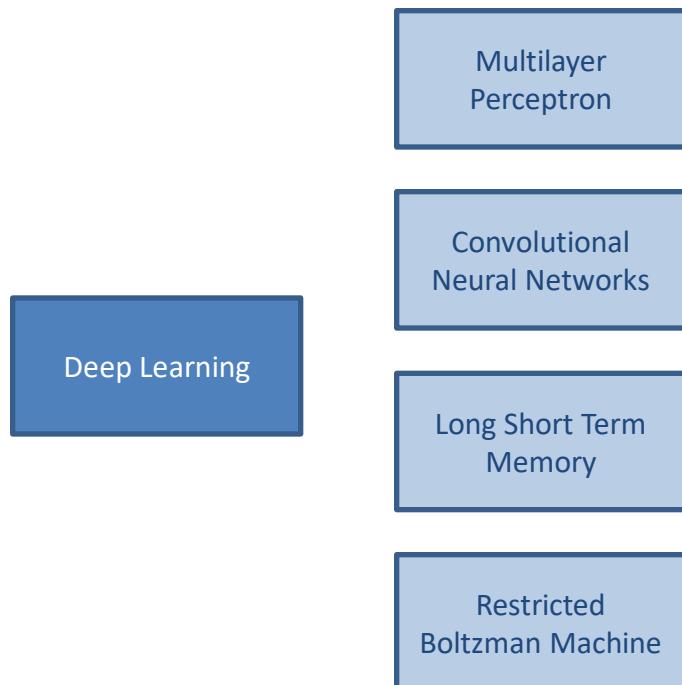
$$P_{l,t} = \frac{1}{1 + e^{-p_{l,t}}}$$

$$y_t = \gamma_0 + \sum_{l=1}^{l^*} \gamma_l P_{l,t}$$

Deep Learning Flow



Deep Learning



Machine Learning In Finance

6.1 Mathematics of Deep Learning

Mathematics of Deep Learning

Deep networks are parametric models that perform sequential operations on their input data. Each such operation, colloquially called a “layer”, consists of a linear transformation, say, a convolution of its input, followed by a pointwise nonlinear “activation function”, e.g., a sigmoid. Deep networks have recently led to dramatic improvements in classification performance in various applications in speech and natural language processing, and computer vision. The crucial property of deep networks that is believed to be the of their performance is that they have a large number of layers as compared to classical neural networks; but there are other architectural modifications such as rectified linear activations (ReLUs) and residual “shortcut” connections. Other major factors in their success is the availability of massive datasets, say, millions of images in datasets like ImageNet, and efficient GPU computing hardware for solving the resultant high-dimensional optimization problem which may have up to 100 million parameters.

The empirical success of deep learning, especially convolutional neural networks (CNNs) for image-based tasks, presents numerous puzzles to theoreticians. In particular, there are three key factors in deep learning, namely the architectures, regularization techniques and optimization algorithms, which are critical to train well-performing deep networks and understanding their necessity and interplay is essential if we are to unravel the secrets of their success.

Mathematics of Deep Learning

The empirical success of deep learning, especially convolutional neural networks (CNNs) for image-based tasks, presents numerous puzzles to theoreticians. In particular, there are three key factors in deep learning, namely the architectures, regularization techniques and optimization algorithms, which are critical to train well-performing deep networks and understanding their necessity and interplay is essential if we are to unravel the secrets of their success.

One possible explanation is that deeper architectures are able to better capture **invariant properties** of the data compared to their shallow counterparts.

In computer vision, for example, the category of an object is invariant to changes in viewpoint, illumination, etc. While a mathematical analysis of why deep networks are able to capture such invariances remains elusive, recent progress has shed some light on this issue for certain sub-classes of deep networks. In particular, scattering networks are a class of deep networks whose convolutional filter banks are given by complex, multi-resolution wavelet families.

As a result of this extra structure, they are provably stable and locally invariant signal representations, and reveal the fundamental role of geometry and stability that underpins the generalization performance of modern deep convolutional network architectures.

Mathematics of Deep Learning

Generalization Properties

Another critical property of a neural network architecture is its ability to generalize from a small number of training examples. Traditional results from statistical learning theory show that the number of training examples needed to achieve good generalization grows polynomially with the size of the network.

In practice, however, deep networks are trained with much fewer data than the number of parameters ($N \ll D$ regime) and yet they can be prevented from overfitting using very simple (and seemingly counter-productive) regularization techniques like Dropout , which simply freezes a random subset of the parameters at each iteration.

One possible explanation for this conundrum is that deeper architectures produce an embedding of the input data that approximately preserves the distance between data points in the same class, while increasing the separation between classes.

Mathematics of Deep Learning

Optimization properties

Surprisingly, these techniques from statistical physics are intimately connected to the regularization properties of partial differential equations (PDEs).

For instance, local entropy, the loss that Entropy-SGD minimizes, is the solution of the HamiltonJacobi-Bellman PDE and can therefore be written as a stochastic optimal control problem, which penalizes greedy gradient descent. This direction further leads to connections between variants of SGD with good empirical performance and standard methods in convex optimization such as infconvolutions and proximal methods. Researchers are only now beginning to unravel the loss functions of deep networks in terms of their topology, which dictates the complexity of optimization, and their geometry, which seems to be related to generalization properties of the classifiers

Mathematics of Deep Learning

Optimization properties

The classical approach to training neural networks is to minimize a (regularized) loss using backpropagation , a gradient descent method specialized to neural networks. Modern versions of backpropagation rely on stochastic gradient descent (SGD) to efficiently approximate the gradient for massive datasets. While SGD has been rigorously analyzed only for convex loss functions, in deep learning the loss is a non-convex function of the network parameters, hence there are no guarantees that SGD finds the global minimizer. In practice, there is overwhelming evidence that SGD routinely yields good solutions for deep networks.

Recent work has also revealed that local minima discovered by SGD that lead to good generalization error belong to very flat regions of the parameter space. This motivates algorithms like Entropy-SGD that are specialized to find such regions and draw from similar results in the analysis of binary perceptrons in statistical physics; they have been shown to perform well on deep networks .

Mathematics of Deep Learning

Optimization properties

It is remarkable that empirical practice has managed to converge to the use of the cross-entropy loss with dropout, that happens to be what would have been prescribed by first principles, since for certain choices of distribution, training is equivalent to minimizing the information bottleneck Lagrangian, that yields an approximation of a minimal sufficient invariant statistic, which define an optimal representation.

It is only recently that developments in theory have made this association possible, and developments in optimization and hardware have made deep neural networks a sensational success.

Mathematics of Deep Learning

Stochastic gradient descent (often shortened to SGD), also known as incremental gradient descent, is an iterative method for optimizing a differentiable objective function, a stochastic approximation of gradient descent optimization.

A recent article implicitly credits Herbert Robbins and Sutton Monro for developing SGD in their 1951 article titled "A Stochastic Approximation Method"; see Stochastic approximation for more information.

It is called stochastic because samples are selected randomly (or shuffled) instead of as a single group (as in standard gradient descent) or in the order they appear in the training set.

In stochastic (or "on-line") gradient descent, the true gradient of $Q(w)$ is approximated by a gradient at a single example:

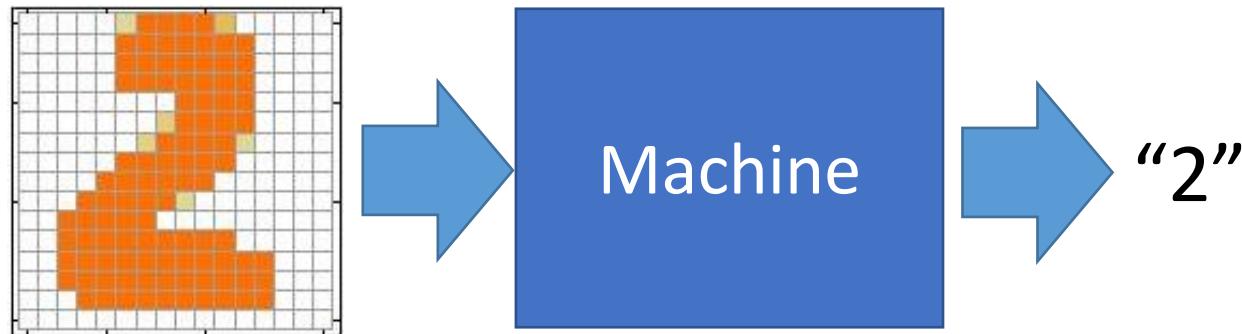
- Choose an initial vector of parameters w and learning rate η .
- Repeat until an approximate minimum is obtained:
 - Randomly shuffle examples in the training set.
 - For $i = 1, 2, \dots, n$ do $w := w - \eta \nabla Q_i(w)$

Machine Learning In Finance

6.2 Deep Learning Example

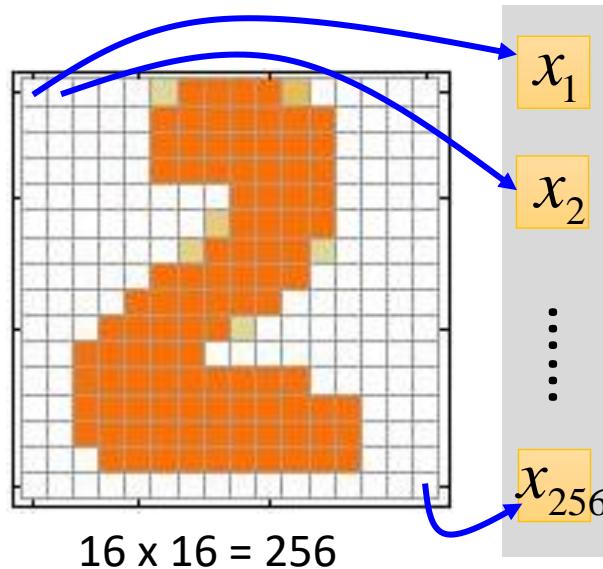
Example Application

- Handwriting Digit Recognition



Handwriting Digit Recognition

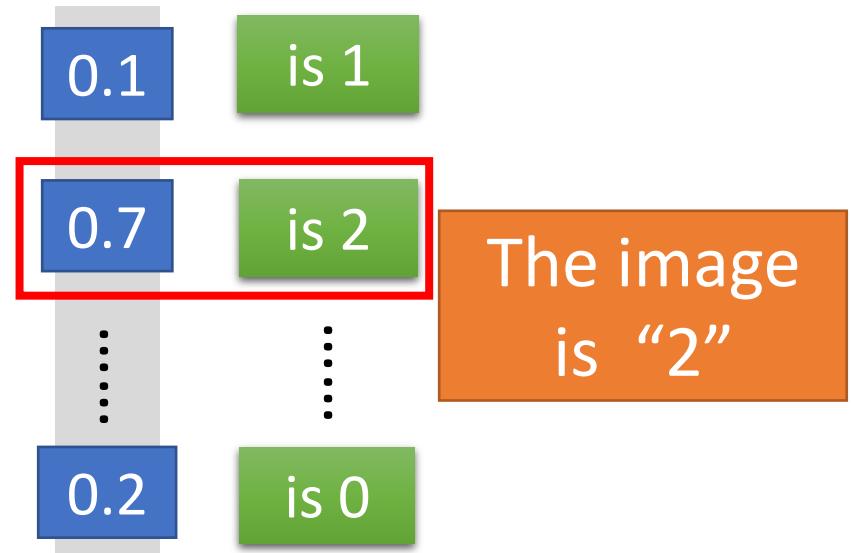
Input



Ink \rightarrow 1

No ink \rightarrow 0

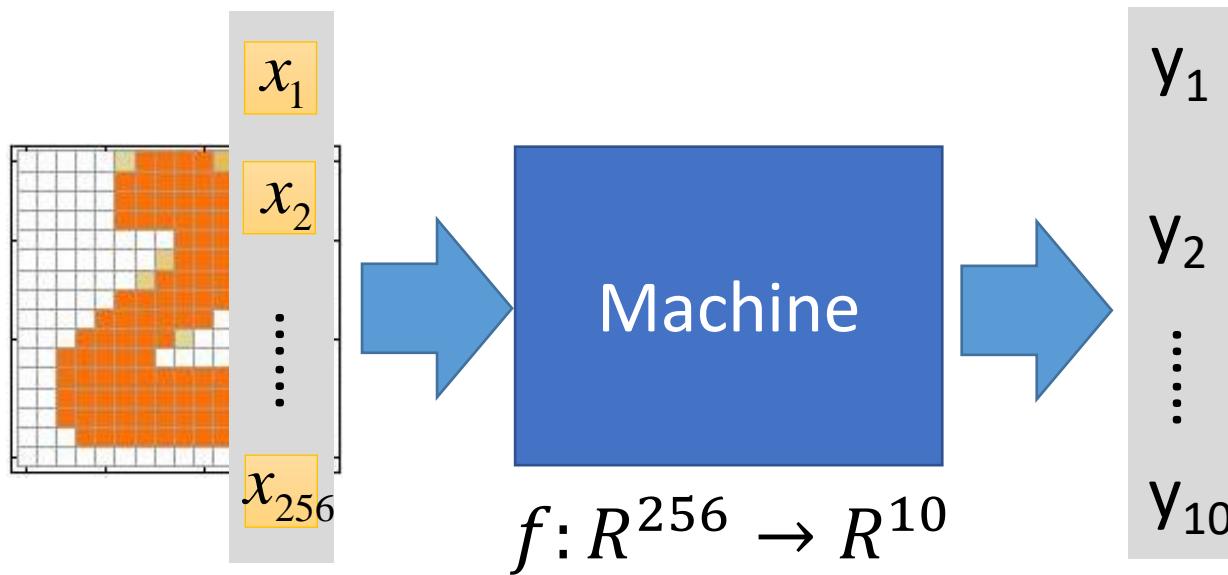
Output



Each dimension represents the confidence of a digit.

Example Application

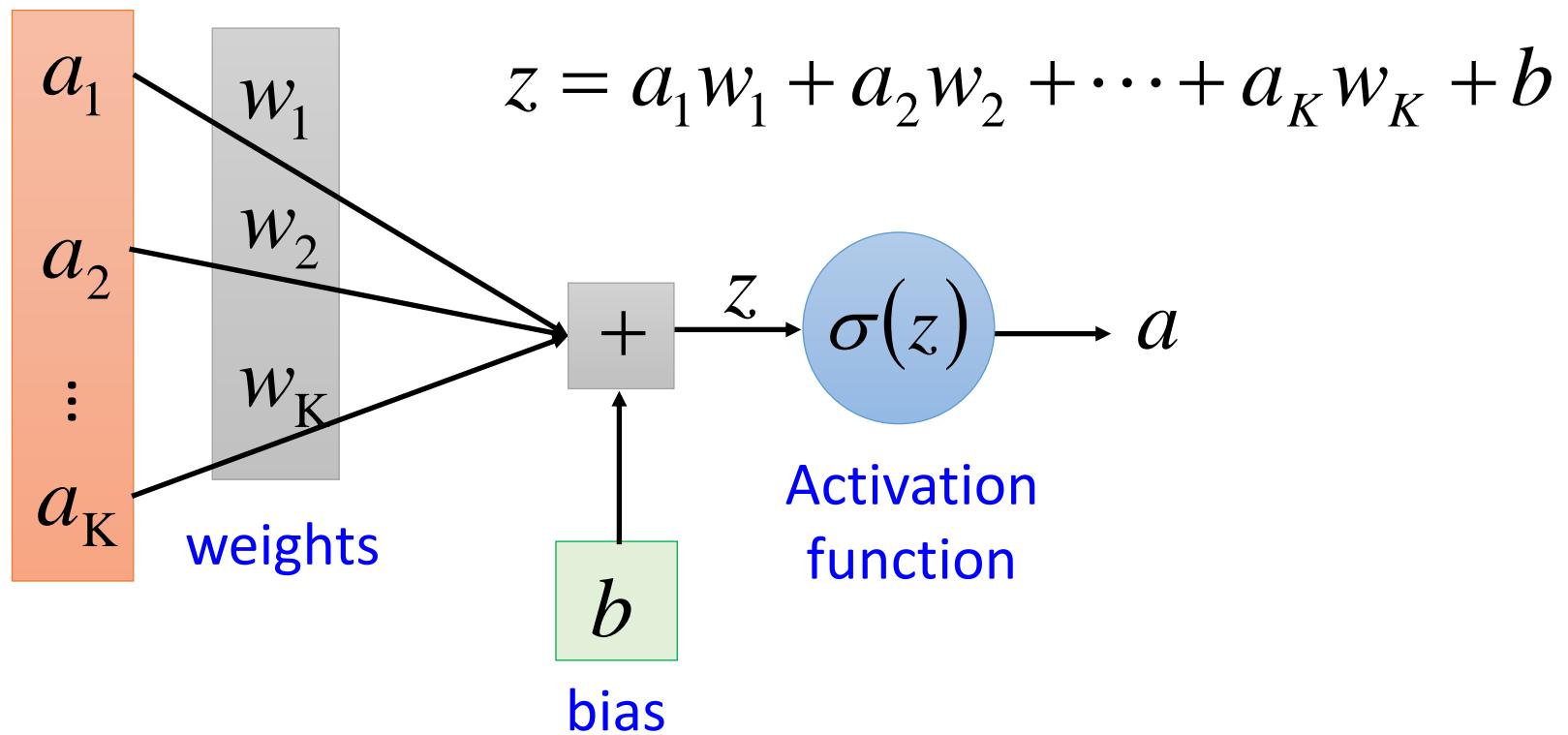
- Handwriting Digit Recognition



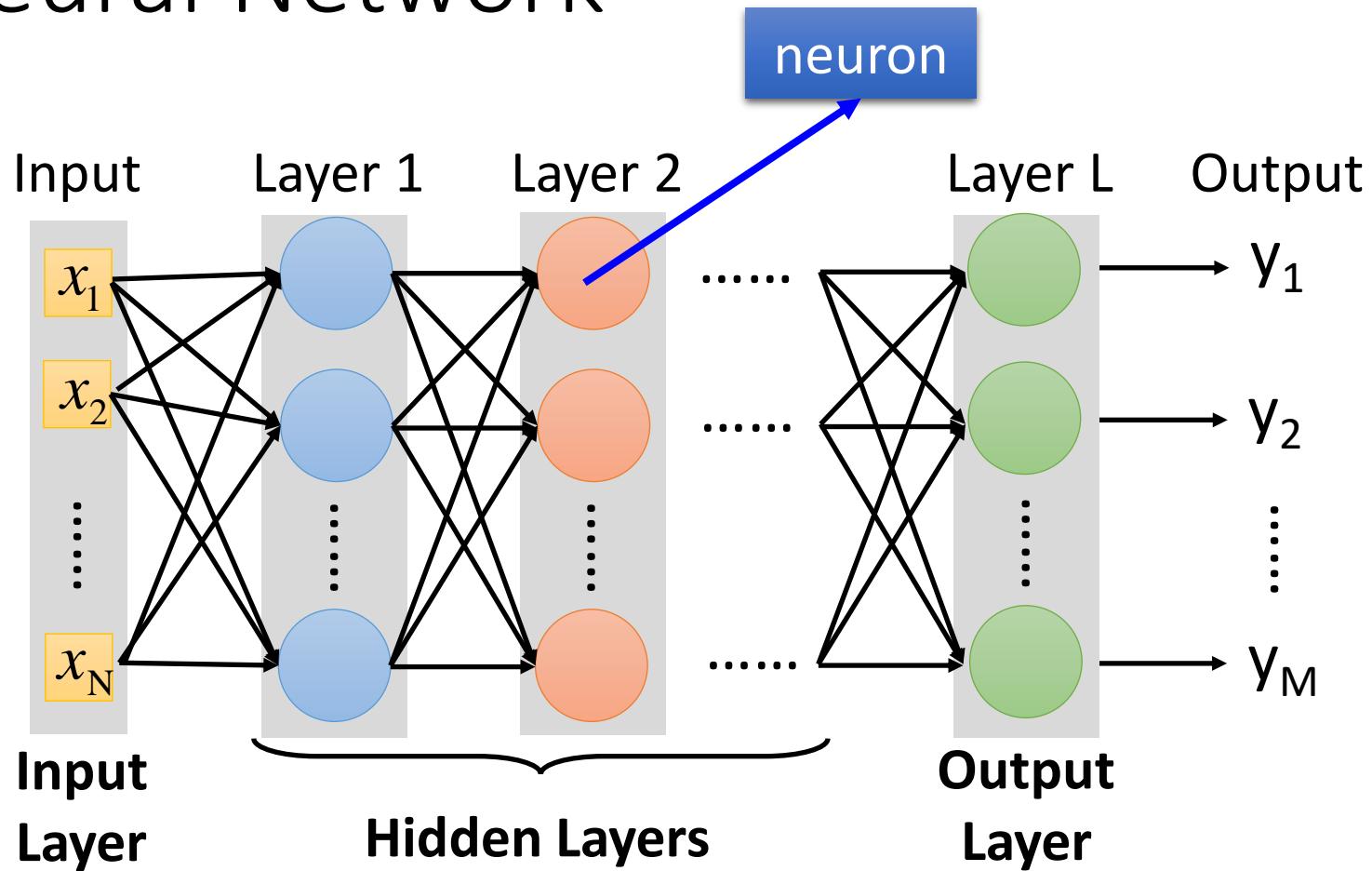
In deep learning, the function f is represented by neural network

Element of Neural Network

Neuron $f: R^K \rightarrow R$

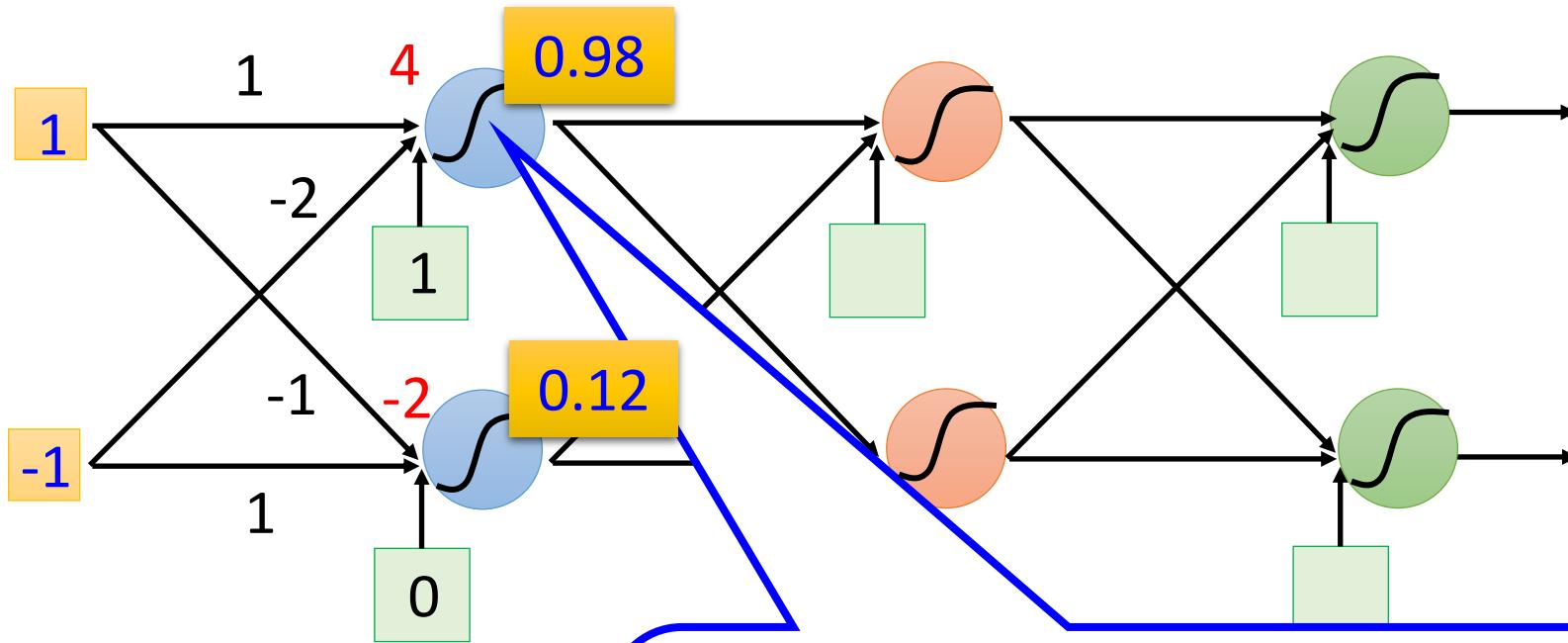


Neural Network



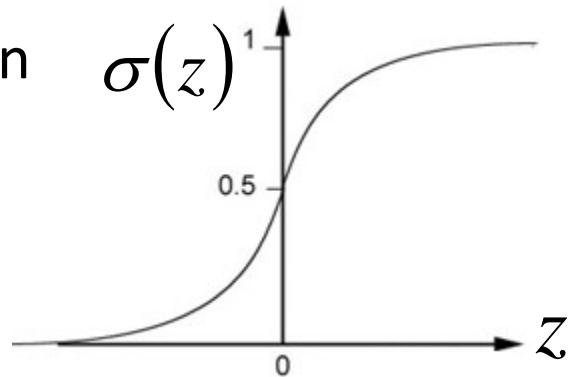
Deep means many hidden layers

Example of Neural Network

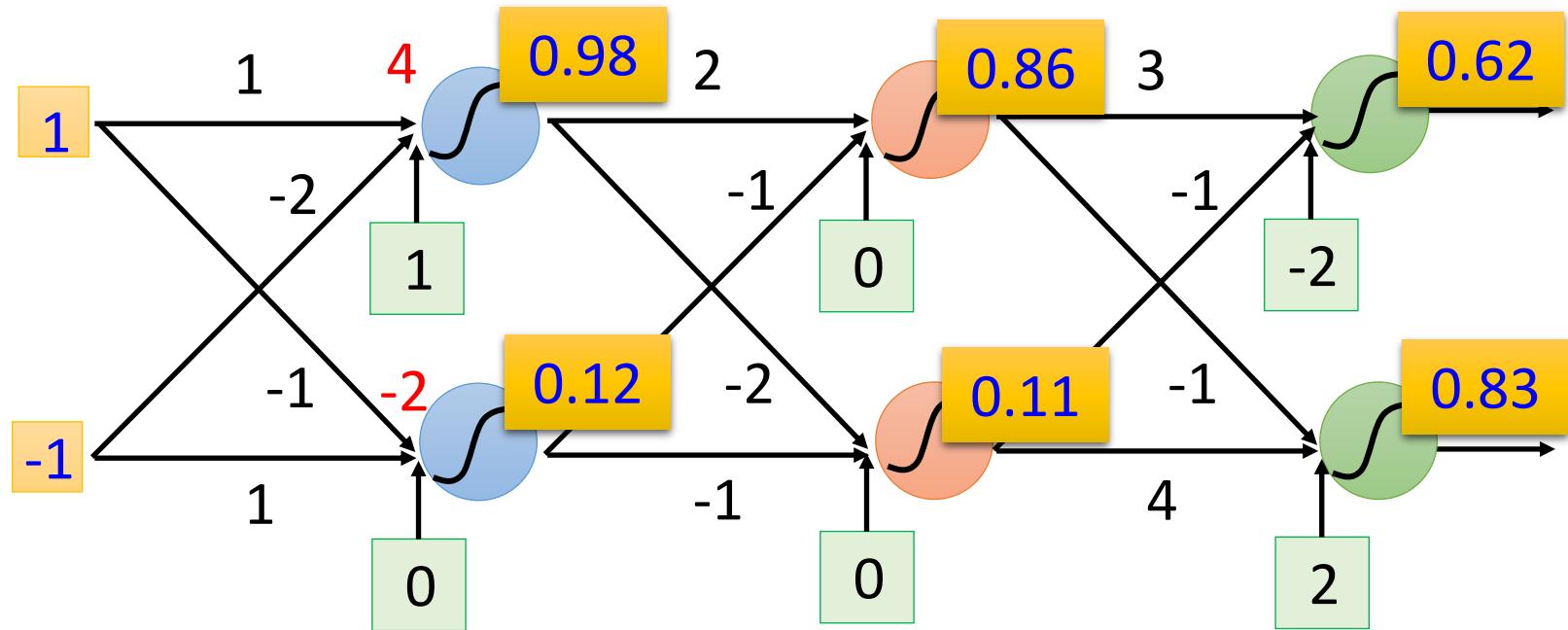


Sigmoid Function

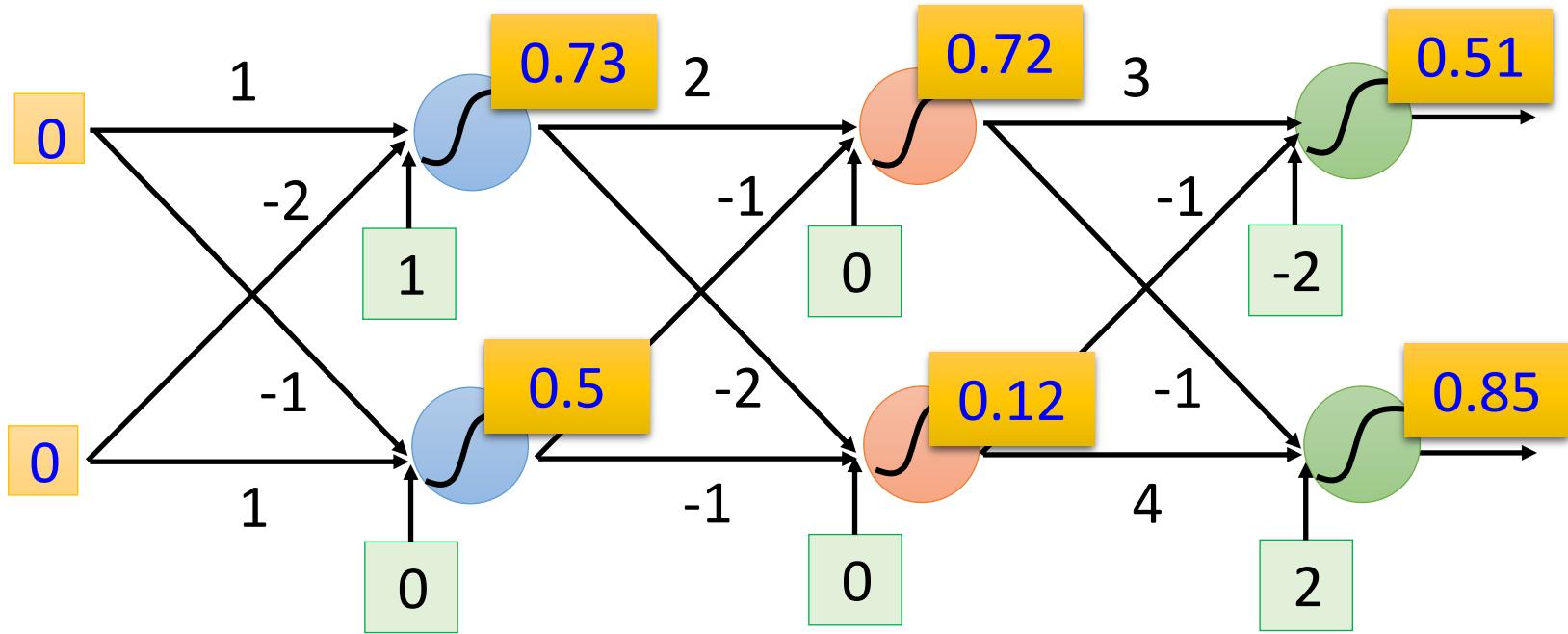
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Example of Neural Network



Example of Neural Network



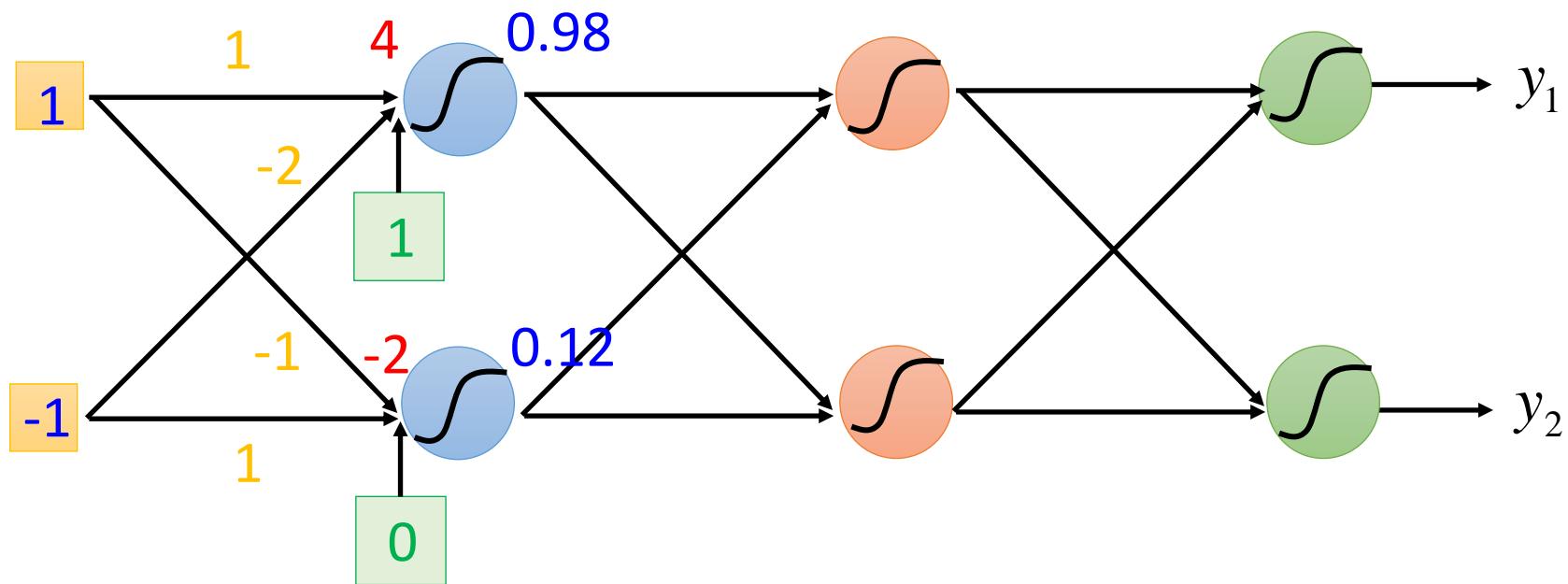
$$f: R^2 \rightarrow R^2$$

$$f \left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \right) = \begin{bmatrix} 0.62 \\ 0.83 \end{bmatrix}$$

$$f \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 0.51 \\ 0.85 \end{bmatrix}$$

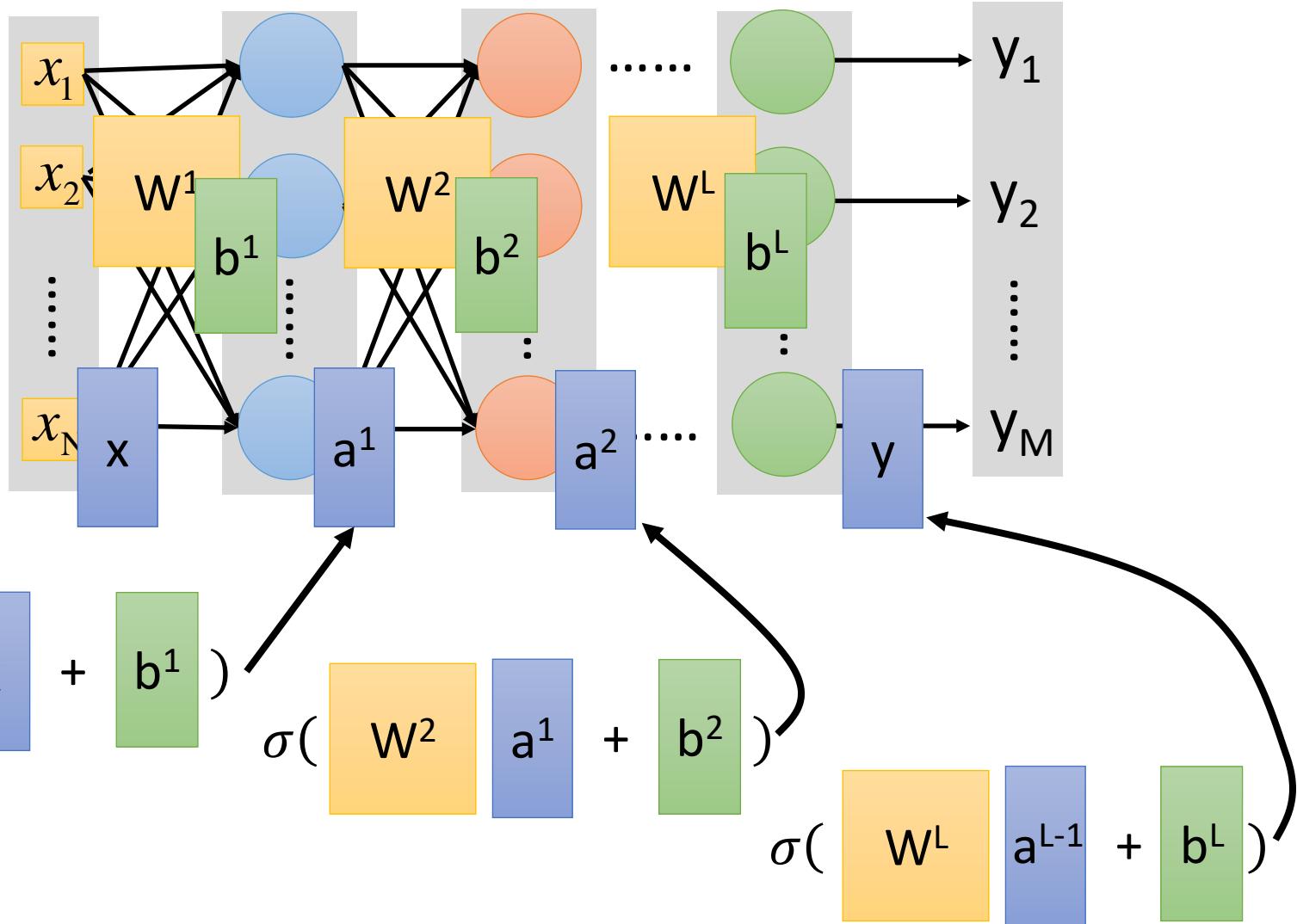
Different parameters define different function

Matrix Operation

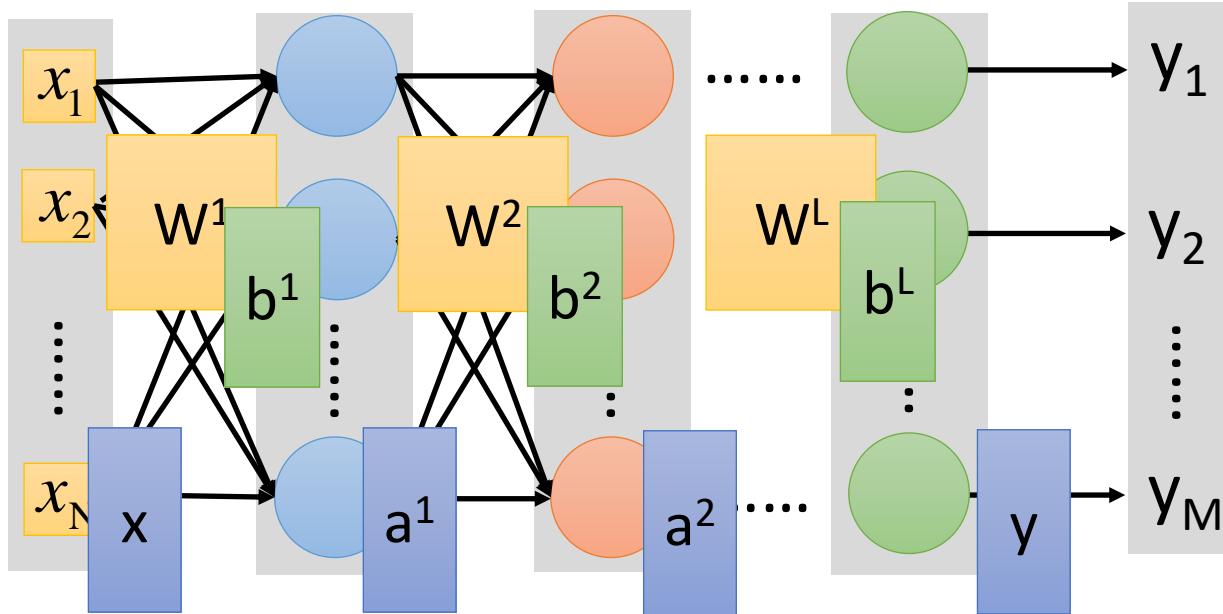


$$\sigma \left(\underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\begin{bmatrix} 4 \\ -2 \end{bmatrix}} \right) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

Neural Network



Neural Network



$$y = f(x)$$

Using parallel computing techniques
to speed up matrix operation

$$= \sigma(W^L \dots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \dots + b^L)$$

Softmax

- Softmax layer as the output layer

Ordinary Layer

$$z_1 \rightarrow \sigma \rightarrow y_1 = \sigma(z_1)$$

$$z_2 \rightarrow \sigma \rightarrow y_2 = \sigma(z_2)$$

$$z_3 \rightarrow \sigma \rightarrow y_3 = \sigma(z_3)$$

In general, the output of network can be any value.

May not be easy to interpret

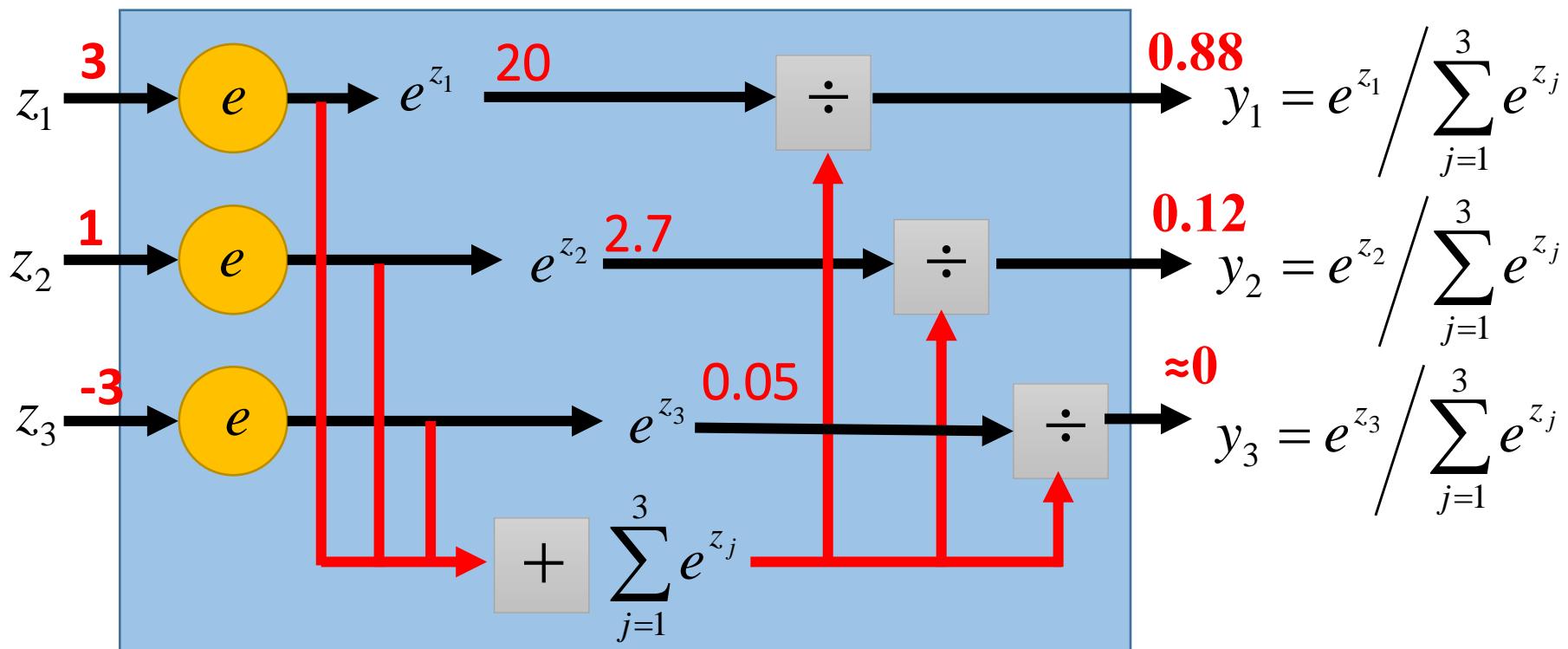
Softmax

- Softmax layer as the output layer

Probability:

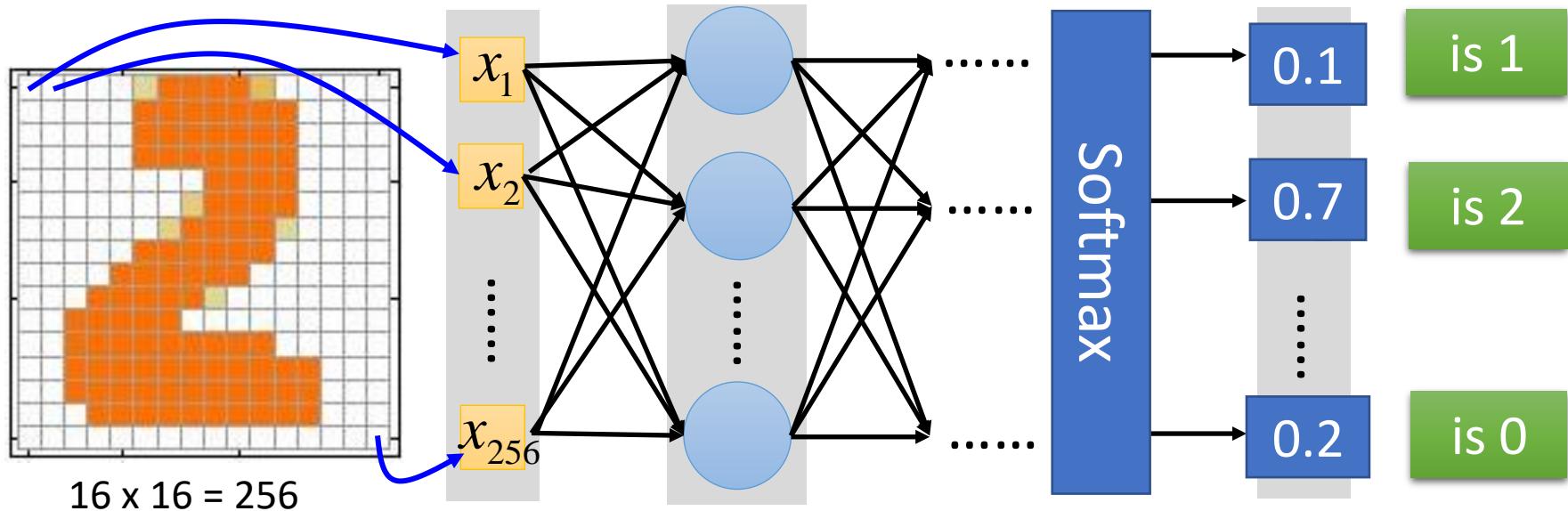
- $1 > y_i > 0$
- $\sum_i y_i = 1$

Softmax Layer



How to set network parameters

$$\theta = \{W^1, b^1, W^2, b^2, \dots, W^L, b^L\}$$



$16 \times 16 = 256$

Ink $\rightarrow 1$

No ink $\rightarrow 0$

Set the network parameters θ such that

Input: How to let the neural network achieve this

Input: y_2 has the maximum value

Training Data

- Preparing training data: images and their labels



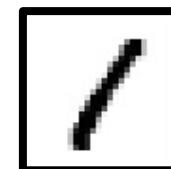
“5”



“0”



“4”



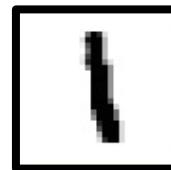
“1”



“9”



“2”



“1”

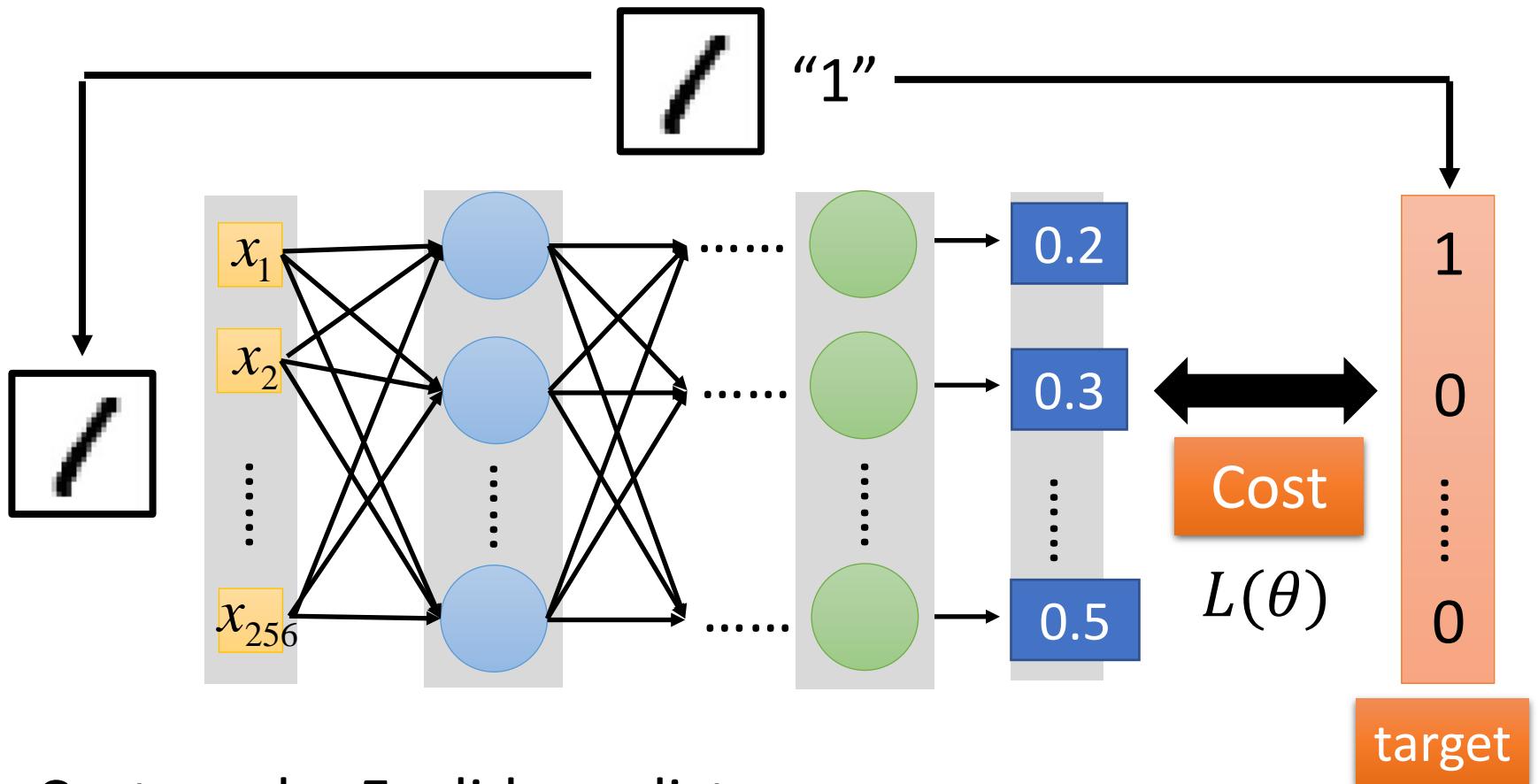


“3”

Using the training data to find
the network parameters.

Cost

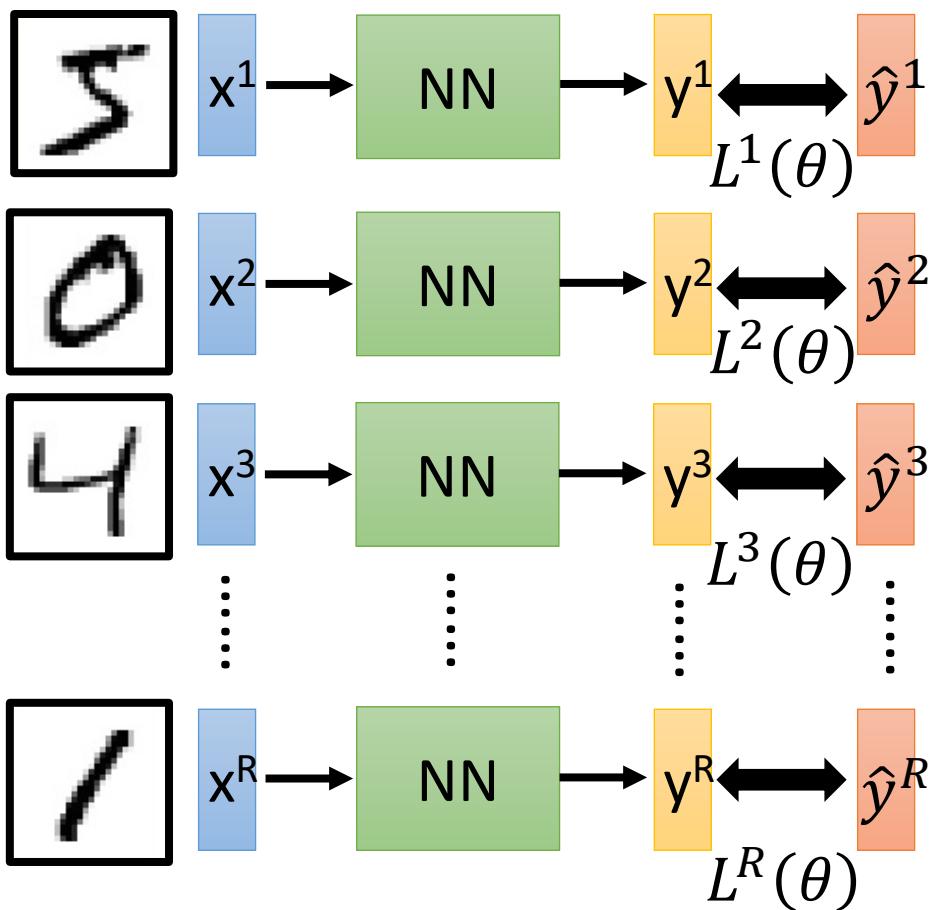
Given a set of network parameters θ , each example has a cost value.



Cost can be Euclidean distance or cross entropy of the network output and target

Total Cost

For all training data ...



Total Cost:

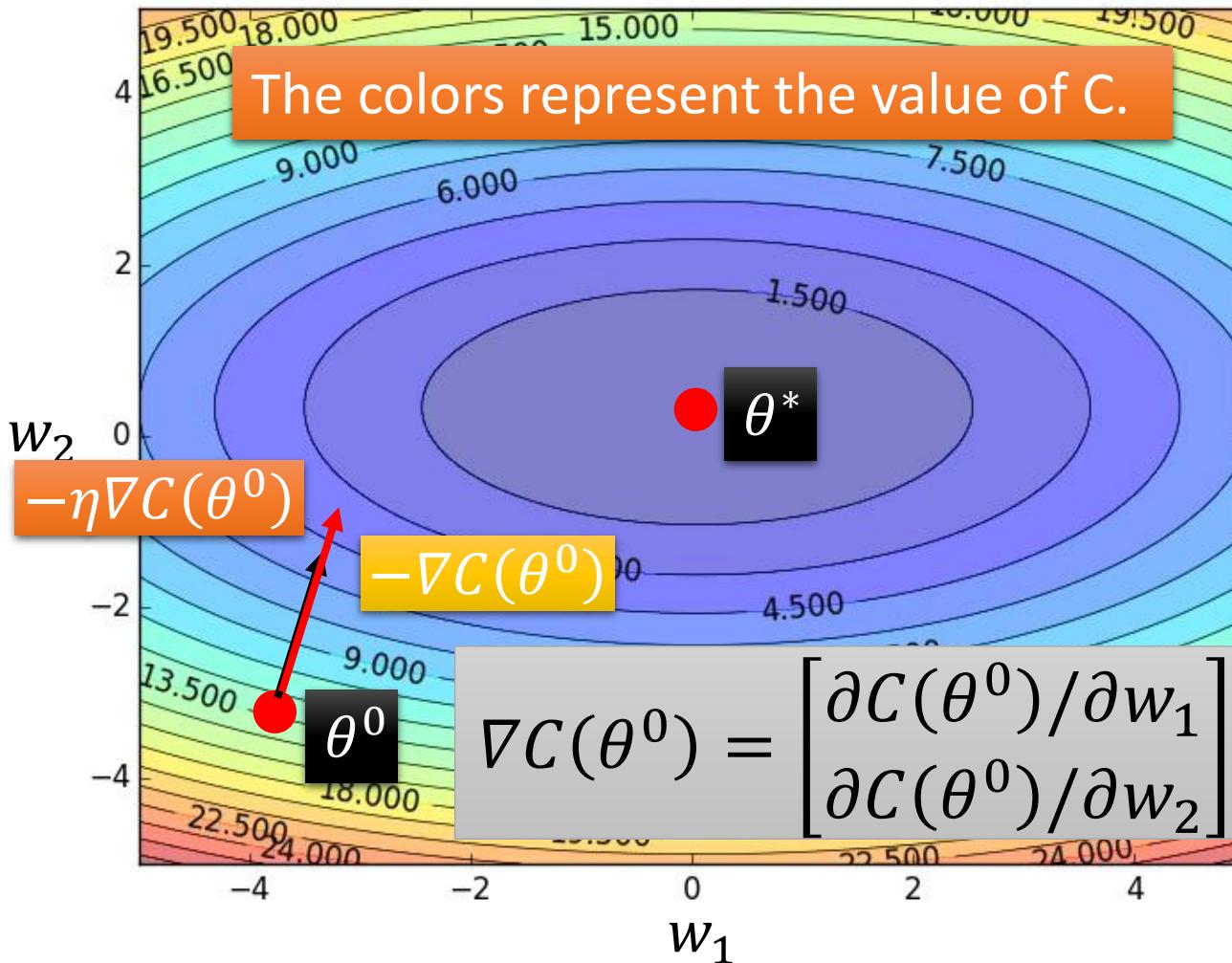
$$C(\theta) = \sum_{r=1}^R L^r(\theta)$$

How bad the network parameters θ is on this task

Find the network parameters θ^* that minimize this value

Gradient Descent

Error Surface



Assume there are only two parameters w_1 and w_2 in a network.

$$\theta = \{w_1, w_2\}$$

Randomly pick a starting point θ^0

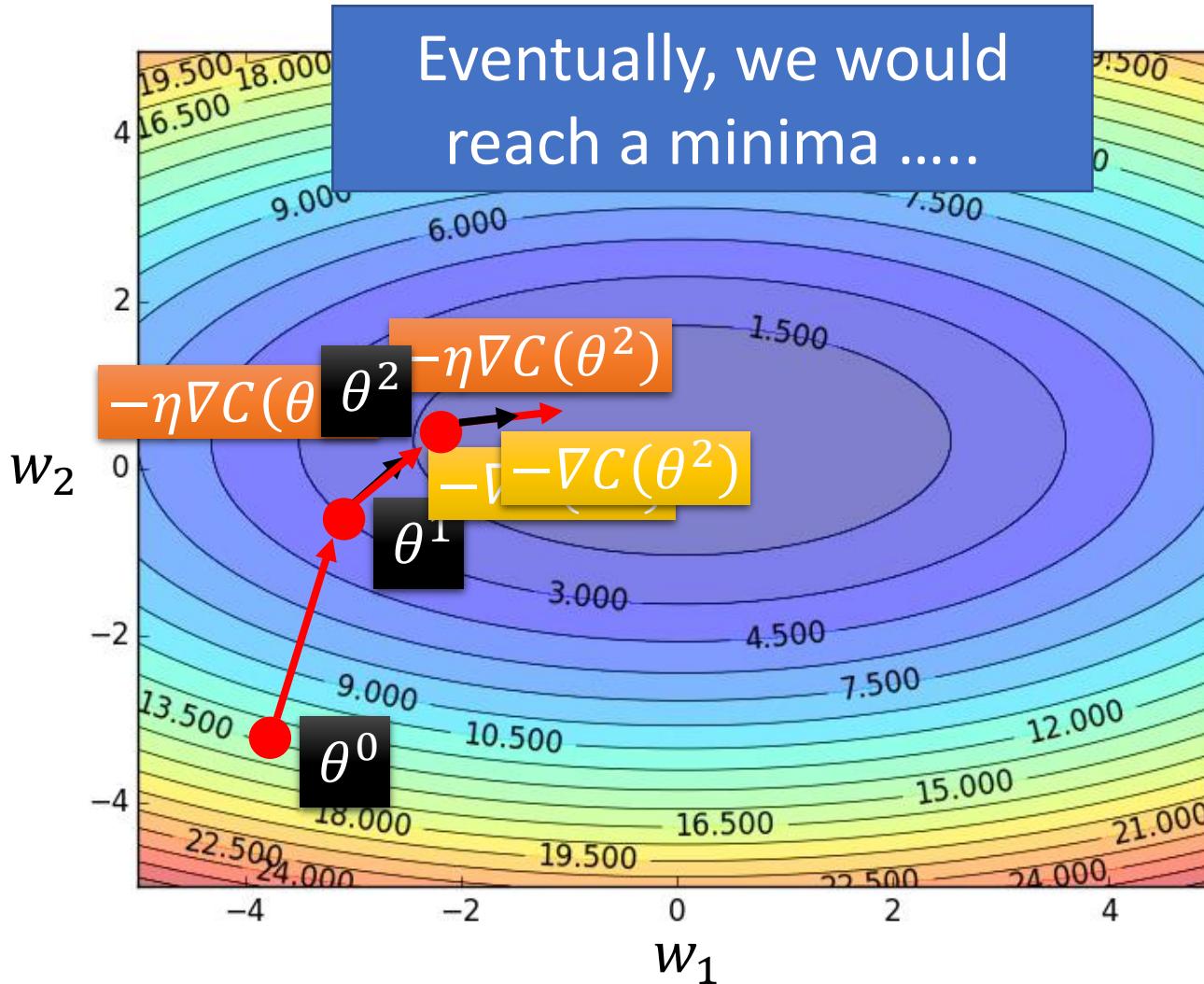
Compute the negative gradient at θ^0

$$\rightarrow -\nabla C(\theta^0)$$

Times the learning rate η

$$\rightarrow -\eta \nabla C(\theta^0)$$

Gradient Descent



Randomly pick a starting point θ^0

Compute the negative gradient at θ^0

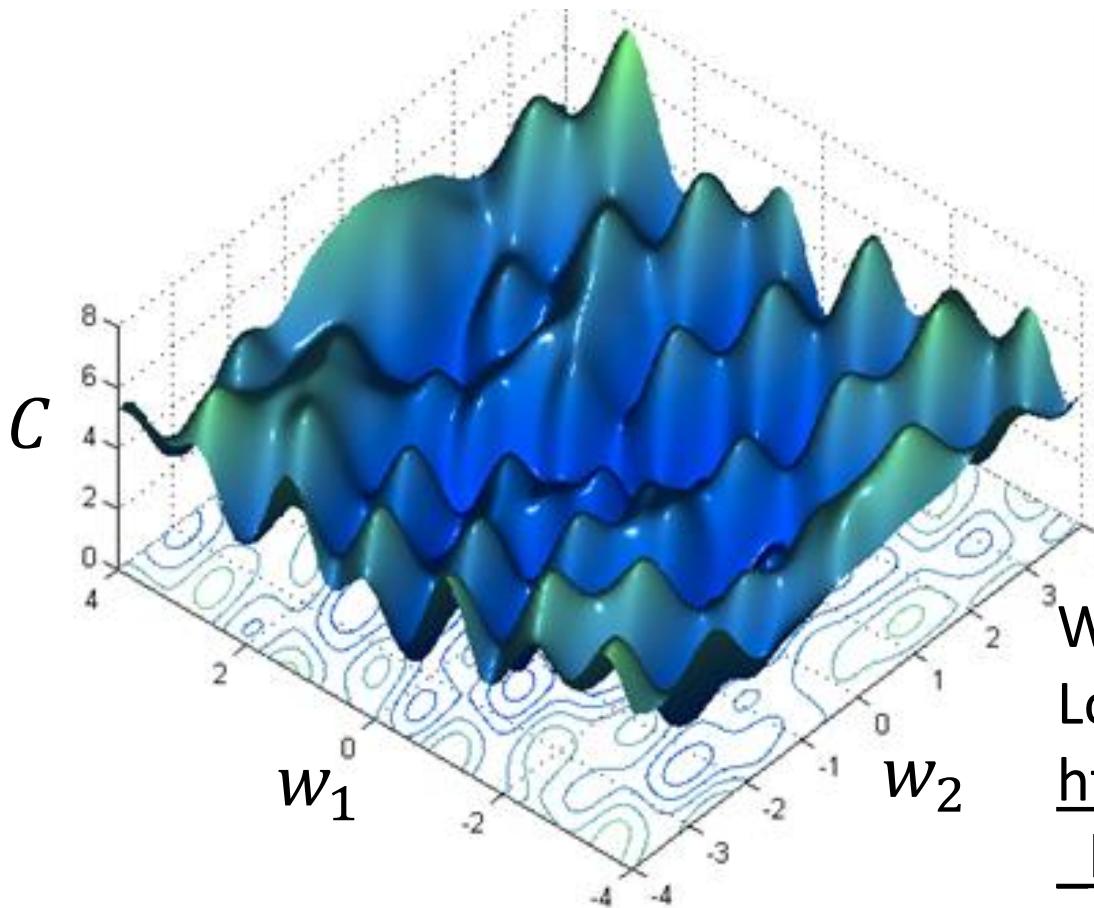
$$\rightarrow -\nabla C(\theta^0)$$

Times the learning rate η

$$\rightarrow -\eta \nabla C(\theta^0)$$

Local Minima

- Gradient descent never guarantee global minima



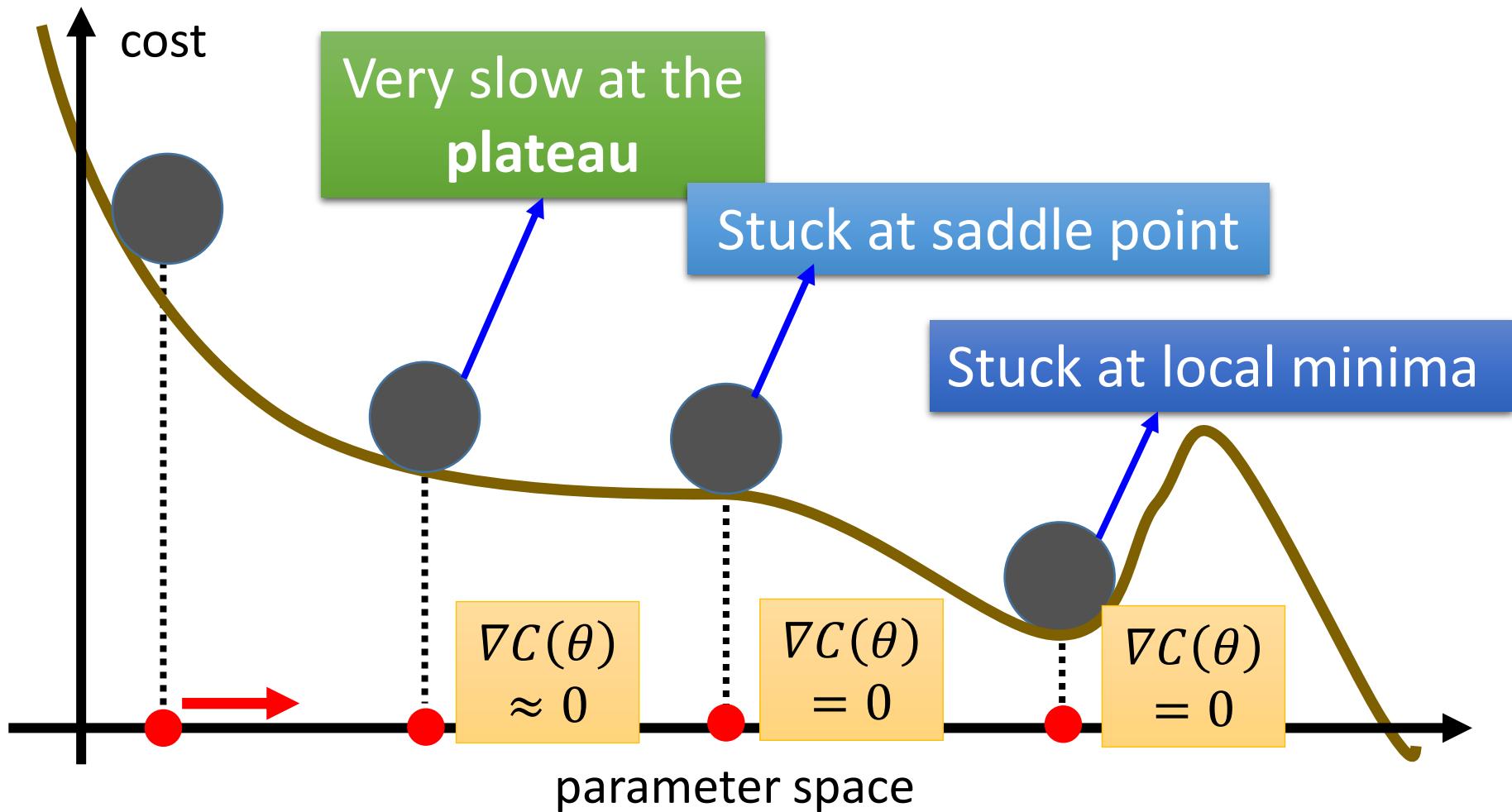
Different initial point θ^0



Reach different minima, so different results

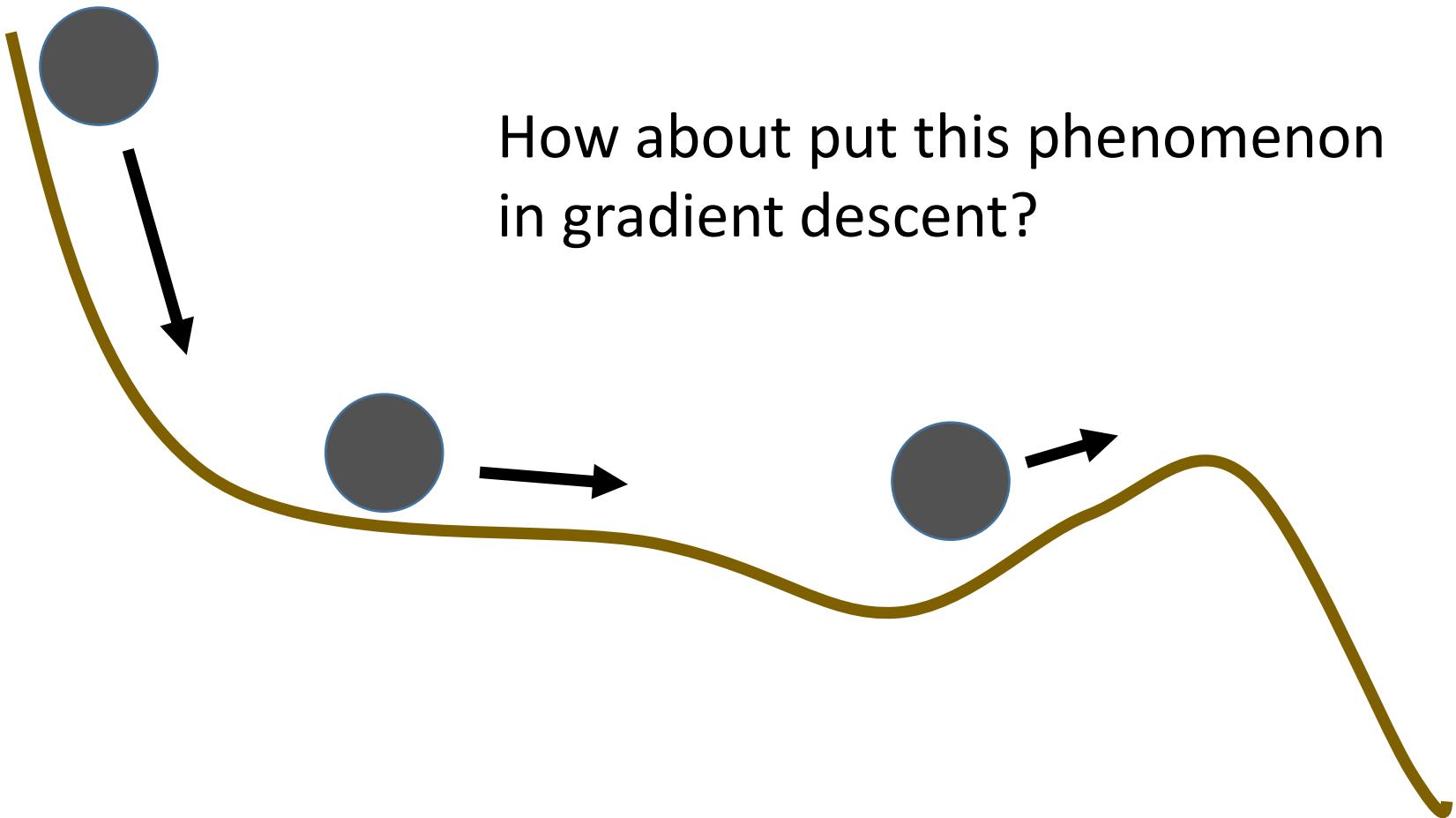
Who is Afraid of Non-Convex Loss Functions?
http://videolectures.net/eml07_lecun_wia/

Besides local minima



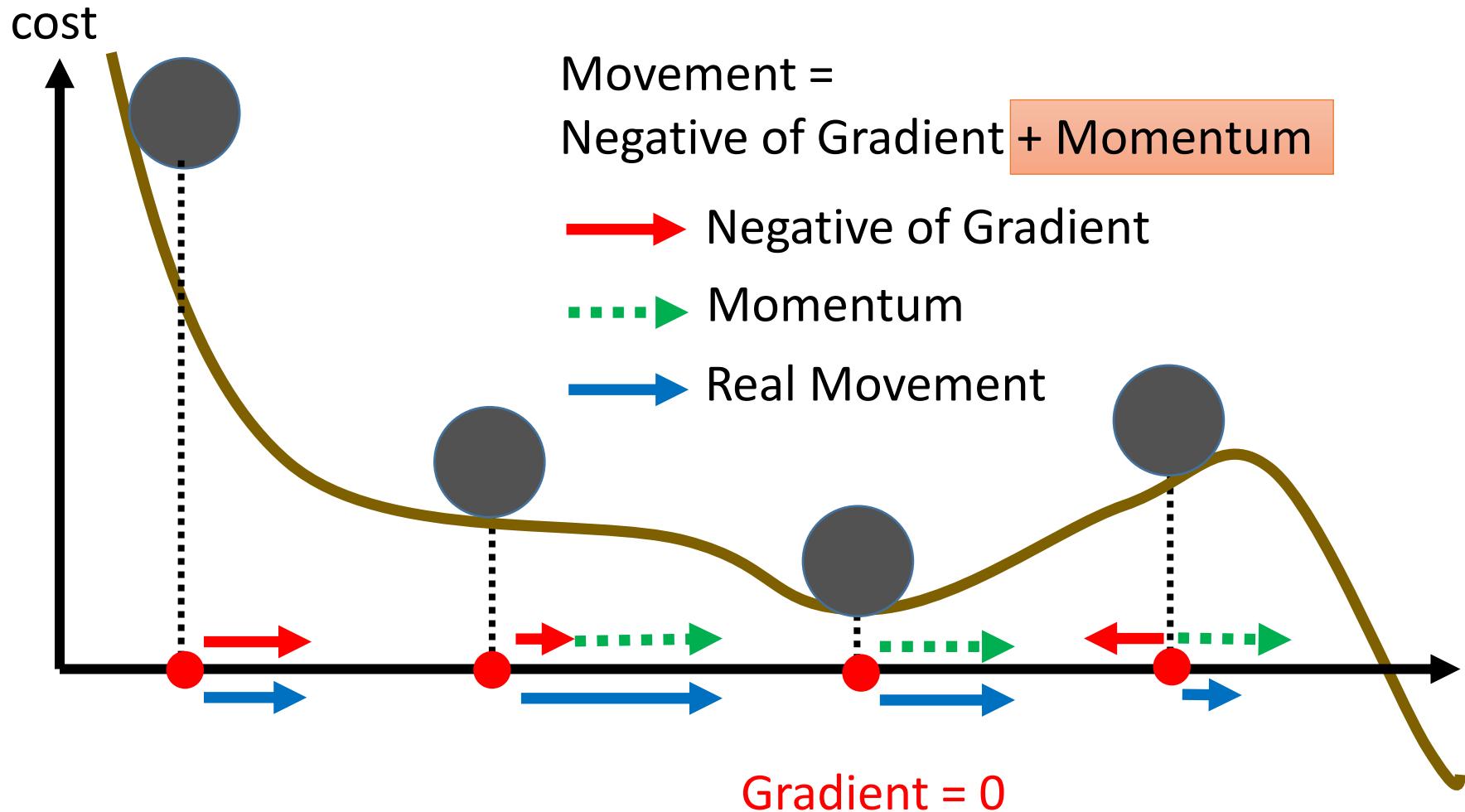
In physical world

- Momentum



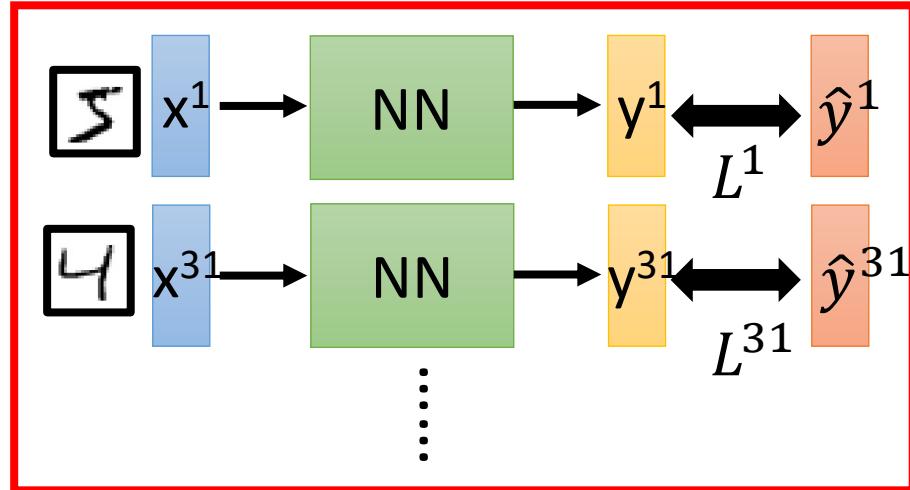
Momentum

Still not guarantee reaching global minima, but give some hope

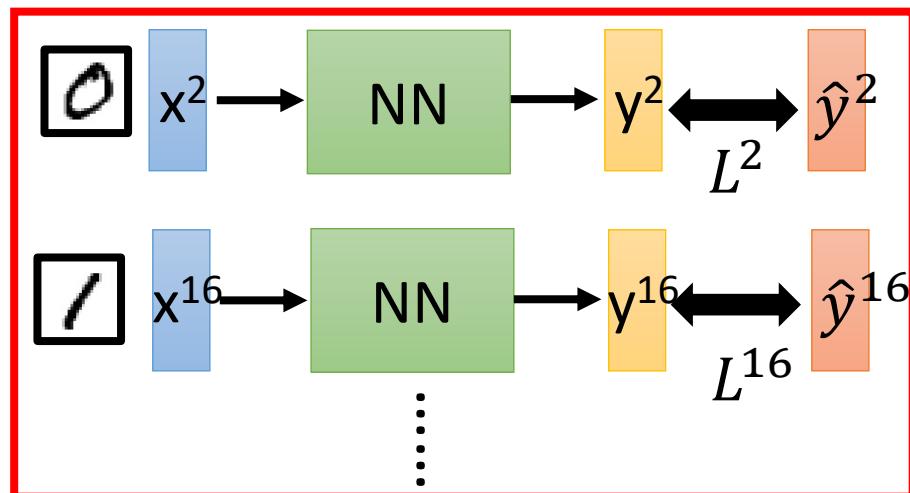


Mini-batch

Mini-batch



Mini-batch



- Randomly initialize θ^0
- Pick the 1st batch
 $C = L^1 + L^{31} + \dots$
 $\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$
- Pick the 2nd batch
 $C = L^2 + L^{16} + \dots$
 $\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$
⋮

C is different each time
when we update
parameters!

Machine Learning In Finance

6.2 Deep Learning Rationale

Deeper is Better?

Layer X Size	Word Error Rate (%)
1 X 2k	24.2
2 X 2k	20.4
3 X 2k	18.4
4 X 2k	17.8
5 X 2k	17.2
7 X 2k	17.1

Not surprised, more parameters, better performance

Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

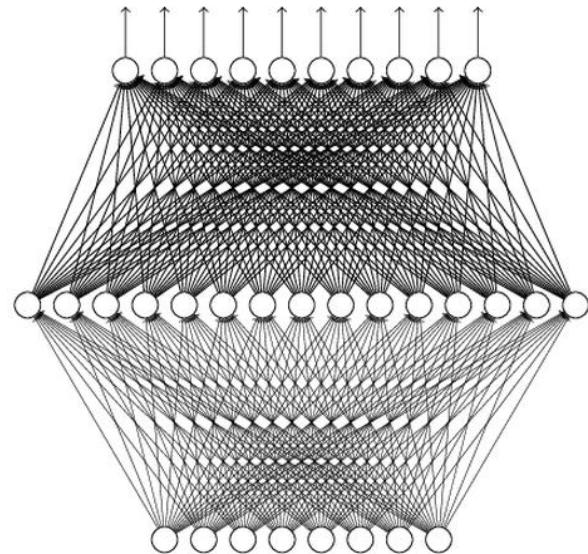
Universality Theorem

Any continuous function f

$$f : R^N \rightarrow R^M$$

Can be realized by a network
with one hidden layer

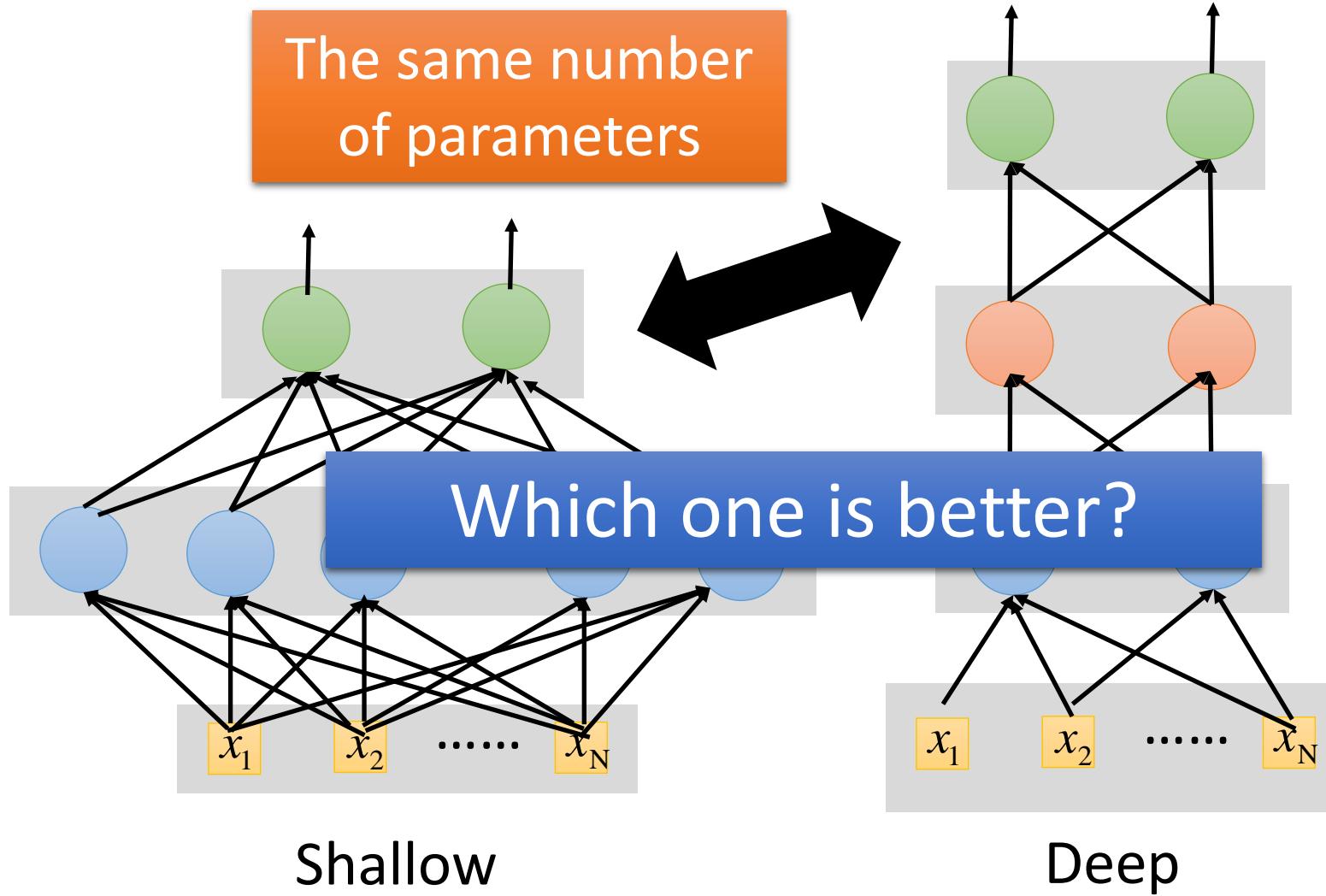
(given **enough** hidden
neurons)



Reference for the reason:
<http://neuralnetworksanddeeplearning.com/chap4.html>

Why “Deep” neural network not “Fat” neural network?

Fat + Short v.s. Thin + Tall



Fat + Short v.s. Thin + Tall

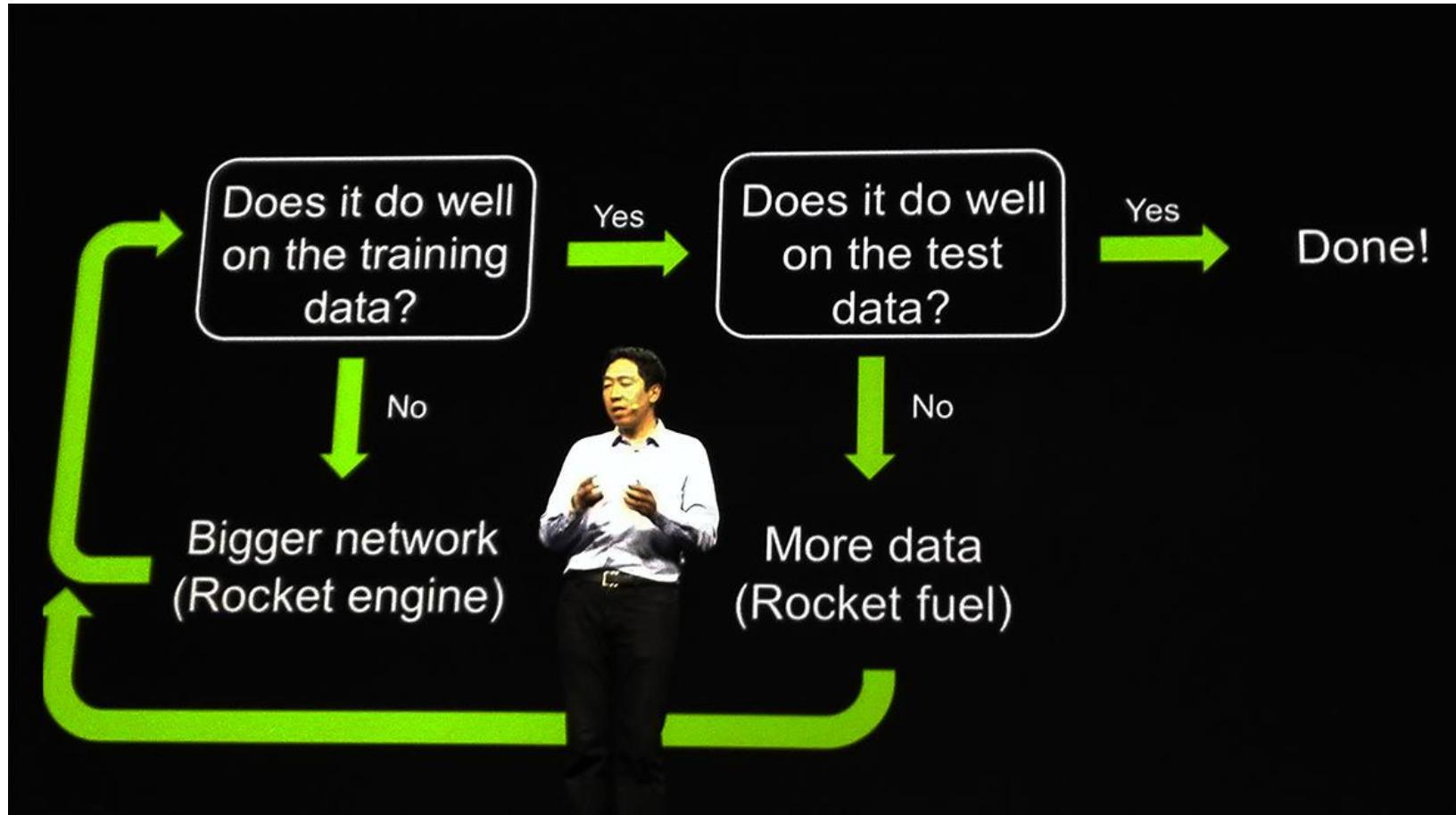
Layer X Size	Word Error Rate (%)	Layer X Size	Word Error Rate (%)
1 X 2k	24.2		
2 X 2k	20.4		
3 X 2k	18.4		
4 X 2k	17.8		
5 X 2k	17.2	1 X 3772	22.5
7 X 2k	17.1	1 X 4634	22.6
		1 X 16k	22.1

Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

Machine Learning In Finance

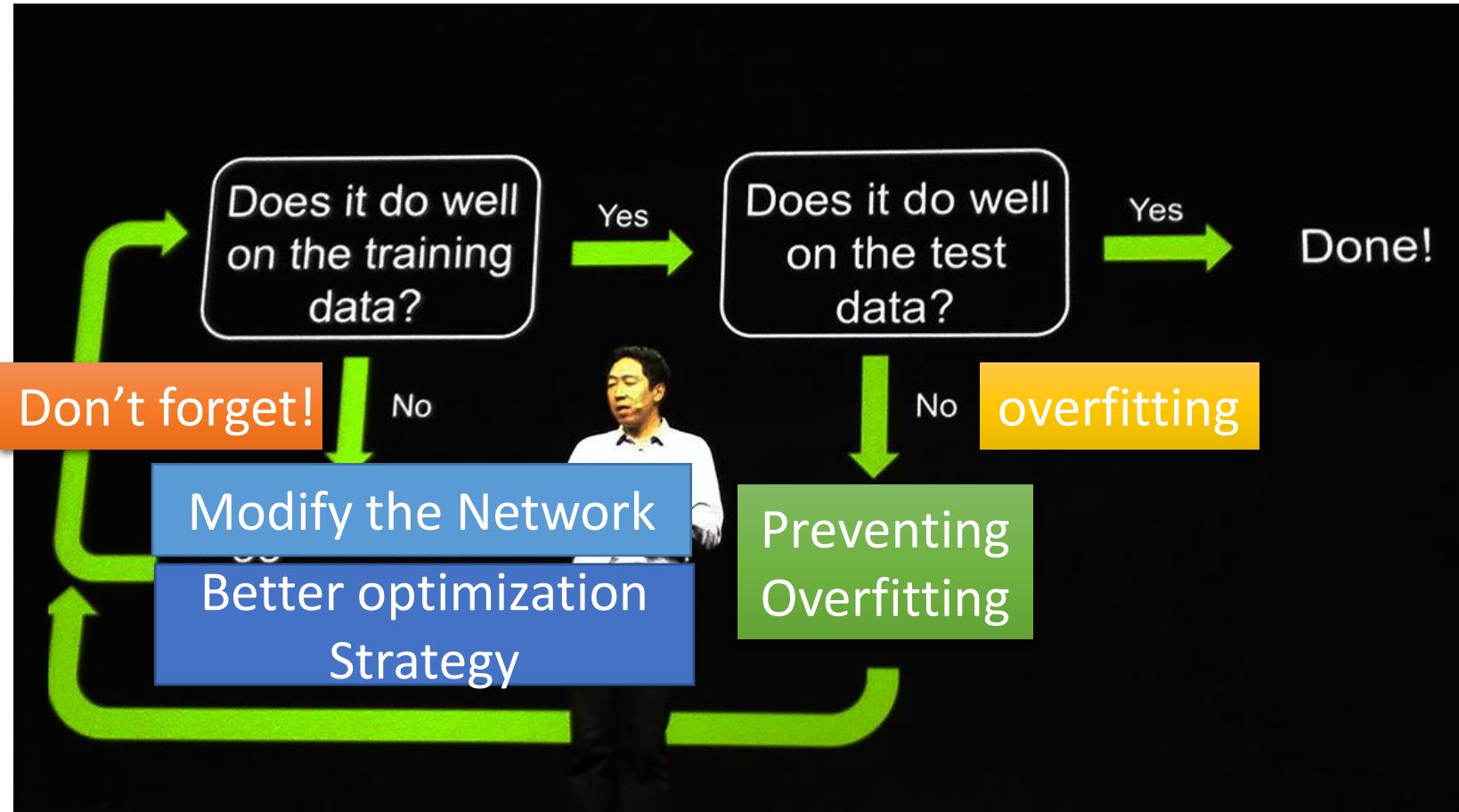
6.3 Deep Learning Training

Recipe for Learning



<http://www.gizmodo.com.au/2015/04/the-basic-recipe-for-machine-learning-explained-in-a-single-powerpoint-slide/>

Recipe for Learning



<http://www.gizmodo.com.au/2015/04/the-basic-recipe-for-machine-learning-explained-in-a-single-powerpoint-slide/>

Recipe for Learning

Modify the Network

- New activation functions, for example, ReLU or Maxout

Better optimization Strategy

- Adaptive learning rates

Prevent Overfitting

- Dropout

Only use this approach when you already obtained good results on the training data.

Machine Learning In Finance

Deep Learning Training

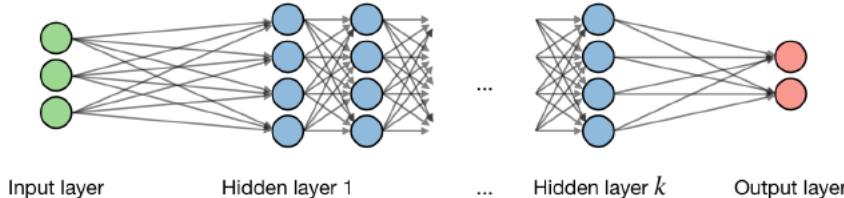
6.3.1 Activation Functions

Deep Learning

Neural Networks

Neural networks are a class of models that are built with layers. Commonly used types of neural networks include convolutional and recurrent neural networks.

□ **Architecture** – The vocabulary around neural networks architectures is described in the figure below:

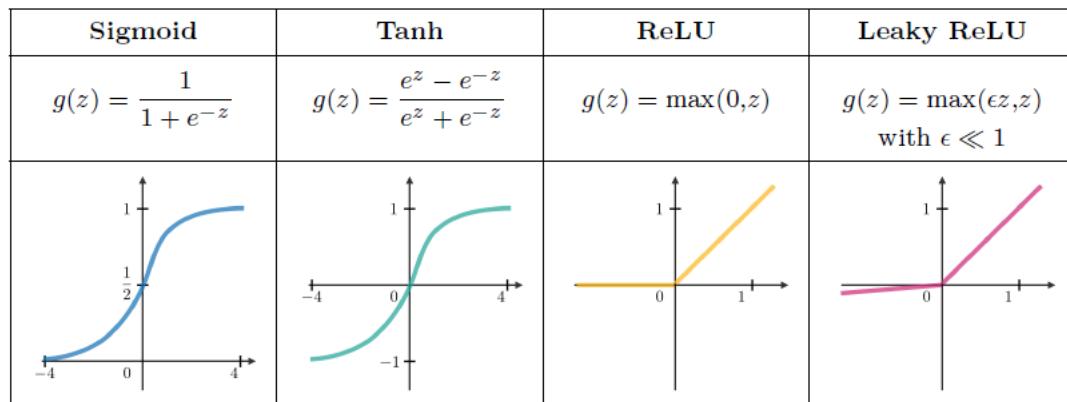


By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$z_j^{[i]} = w_j^{[i]T} x + b_j^{[i]}$$

where we note w , b , z the weight, bias and output respectively.

□ **Activation function** – Activation functions are used at the end of a hidden unit to introduce non-linear complexities to the model. Here are the most common ones:

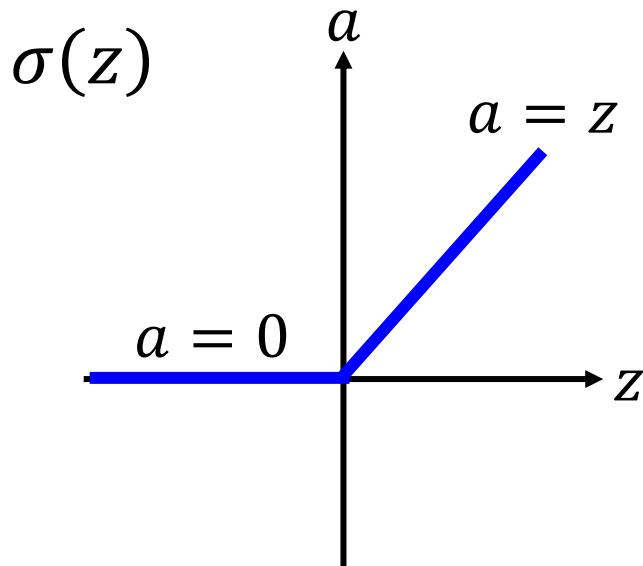


□ **Cross-entropy loss** – In the context of neural networks, the cross-entropy loss $L(z,y)$ is commonly used and is defined as follows:

$$L(z,y) = - \left[y \log(z) + (1 - y) \log(1 - z) \right]$$

ReLU

- Rectified Linear Unit (ReLU)

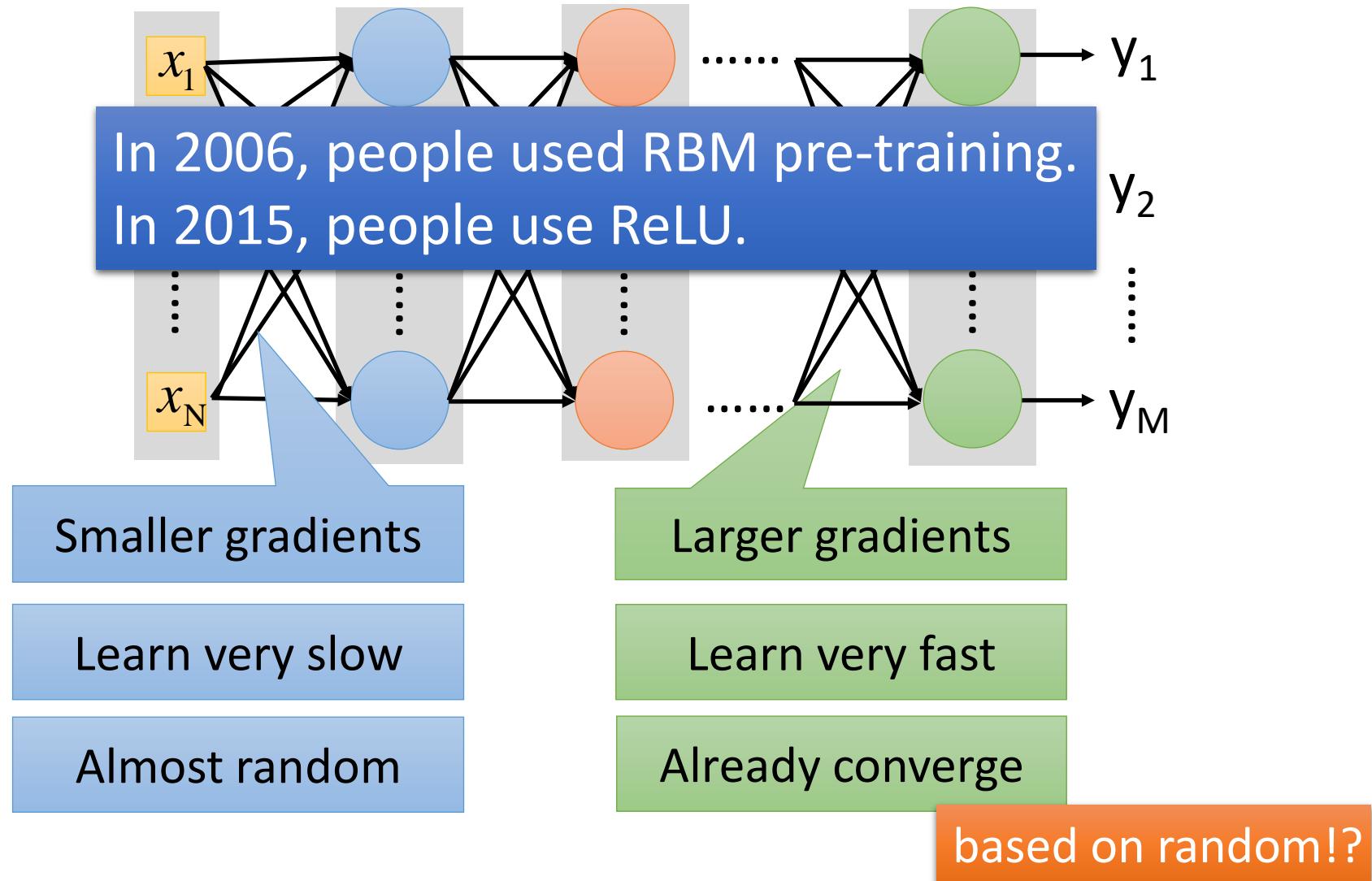


[Xavier Glorot, AISTATS'11]
[Andrew L. Maas, ICML'13]
[Kaiming He, arXiv'15]

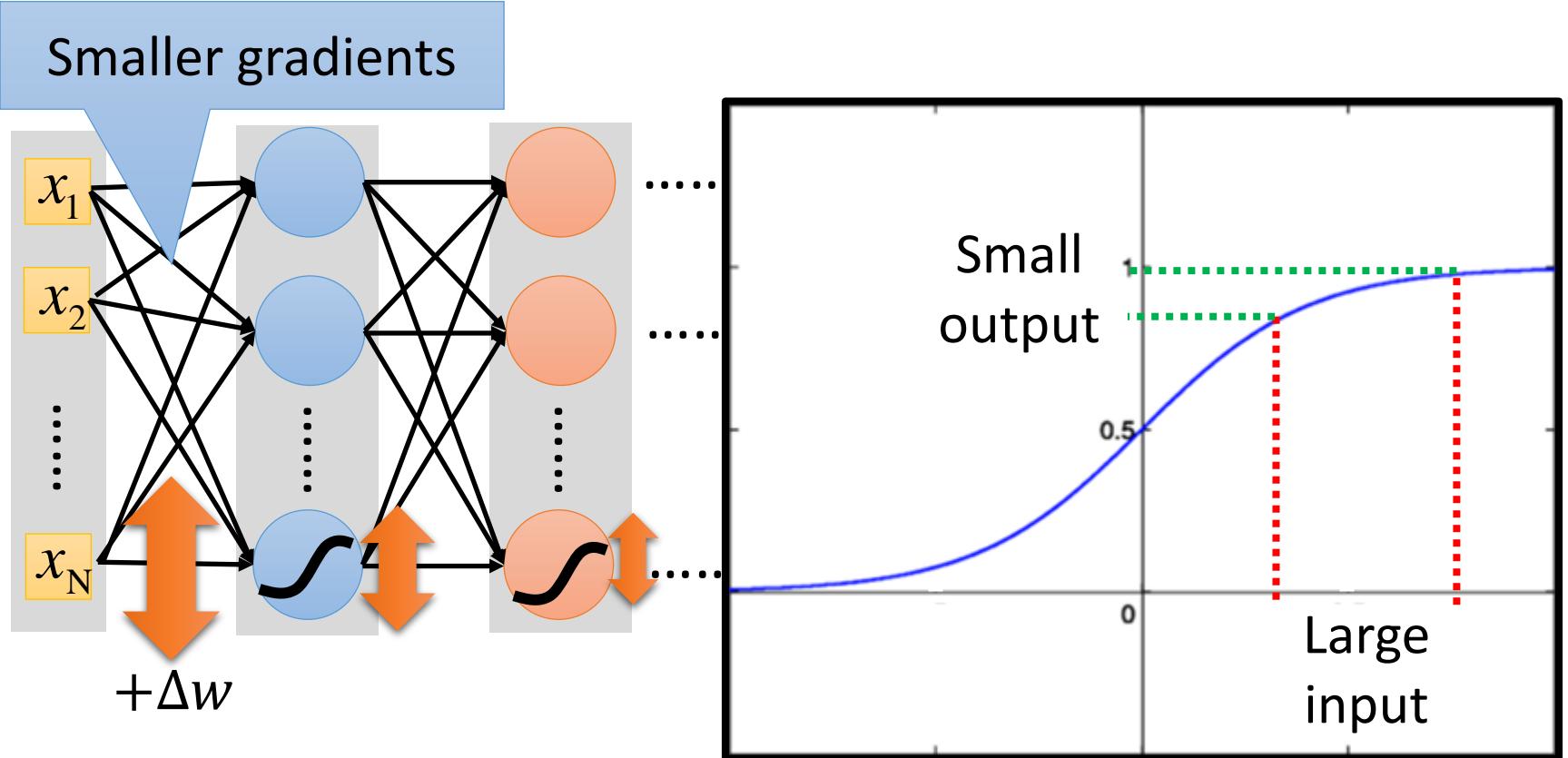
Reason:

1. Fast to compute
2. Biological reason
3. Infinite sigmoid with different biases
4. Vanishing gradient problem

Vanishing Gradient Problem



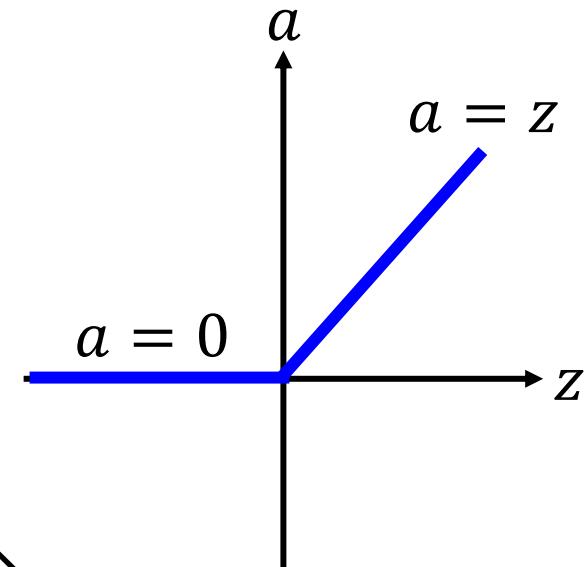
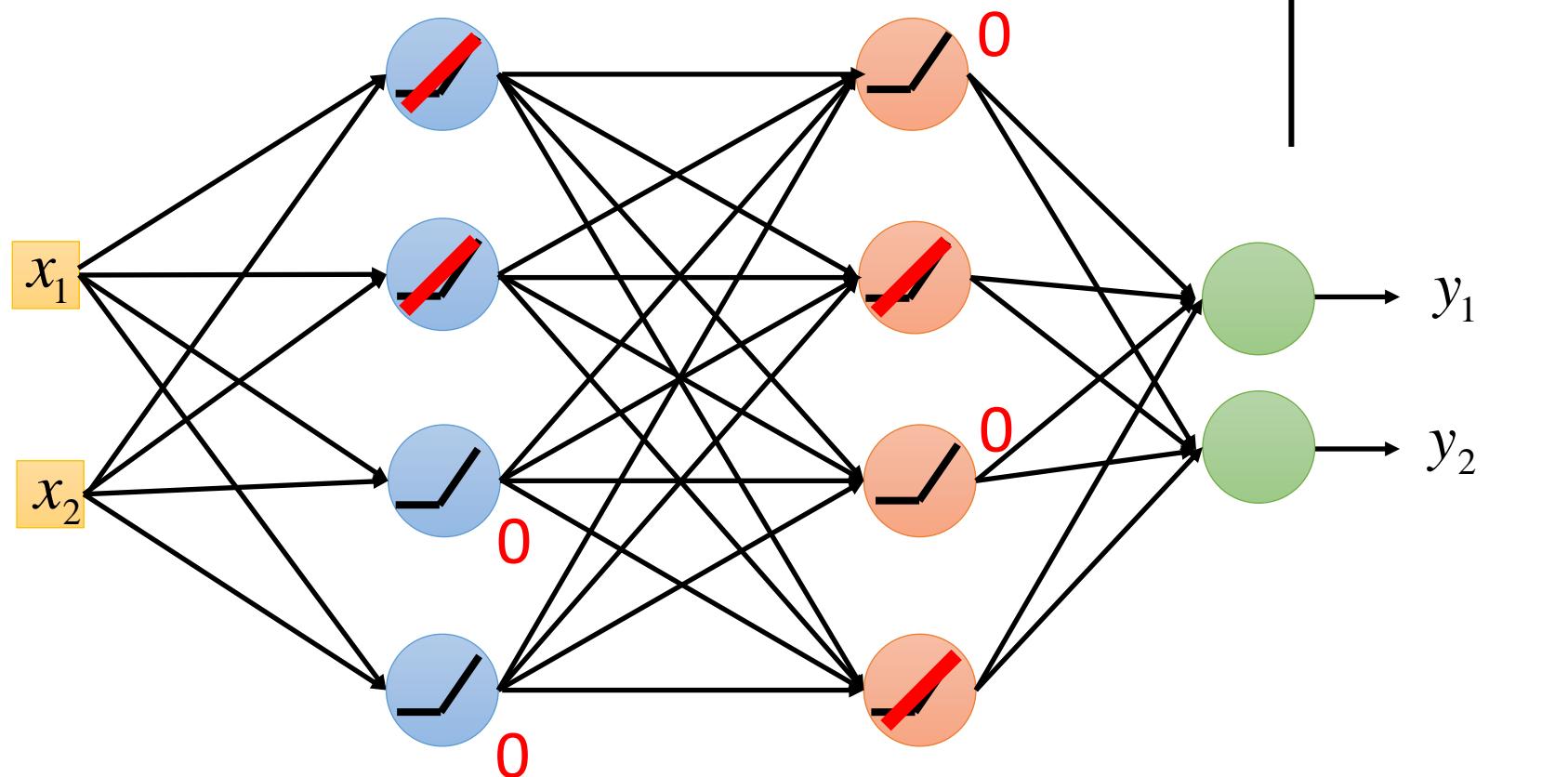
Vanishing Gradient Problem



Intuitive way to compute the gradient ...

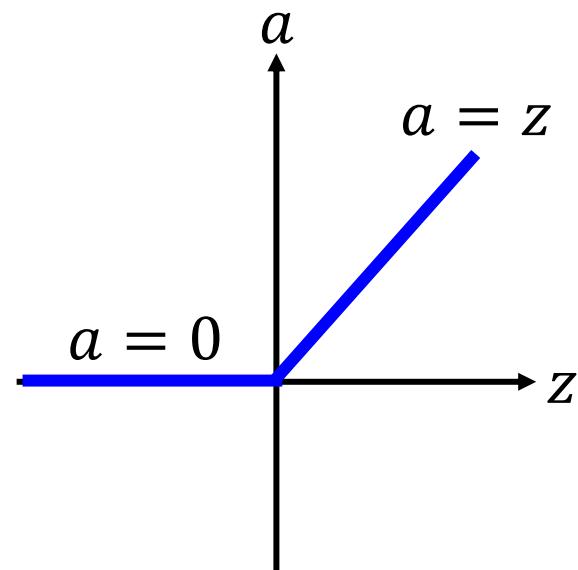
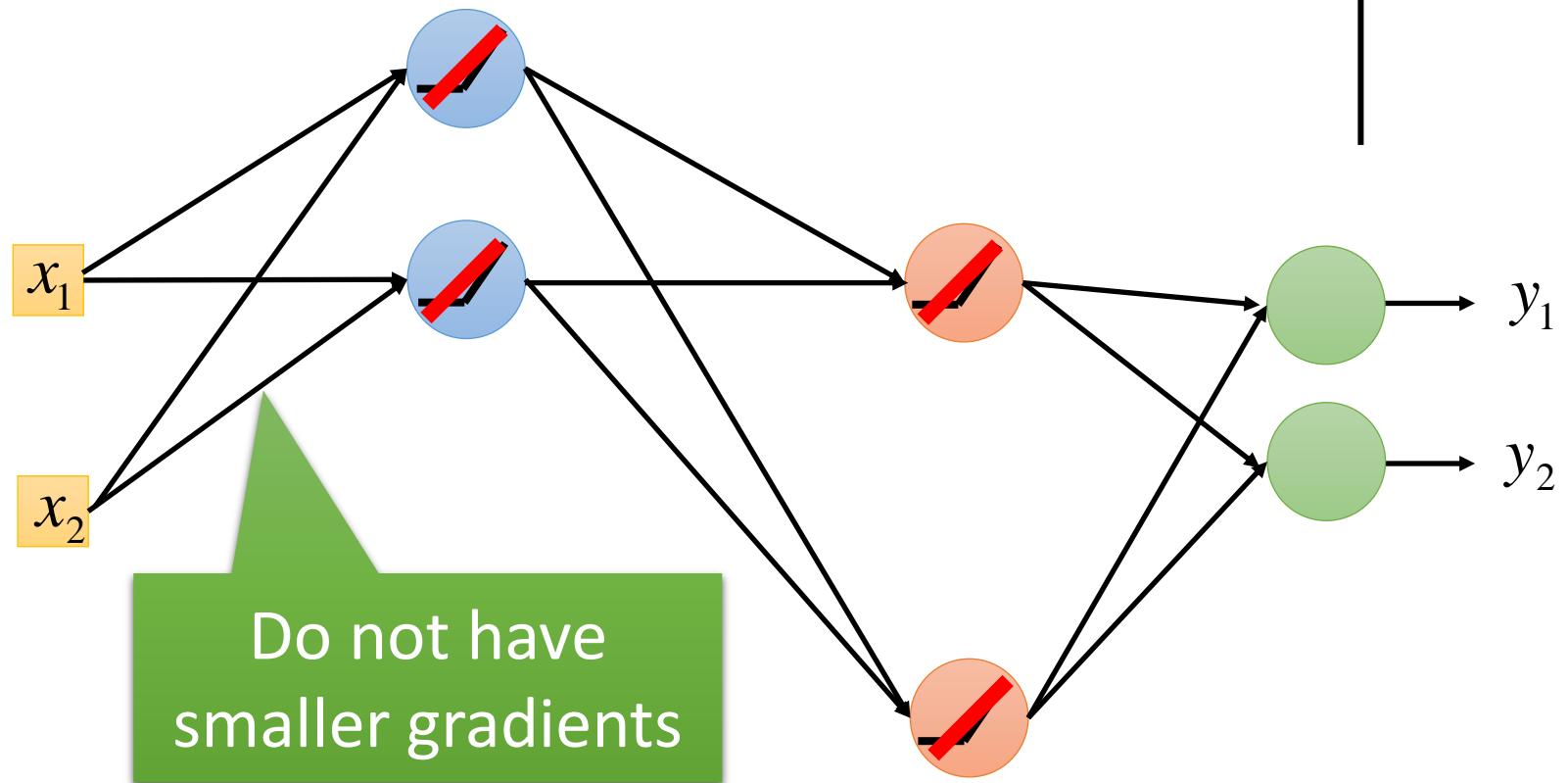
$$\frac{\partial C}{\partial w} = ? \quad \frac{\Delta C}{\Delta w}$$

ReLU



ReLU

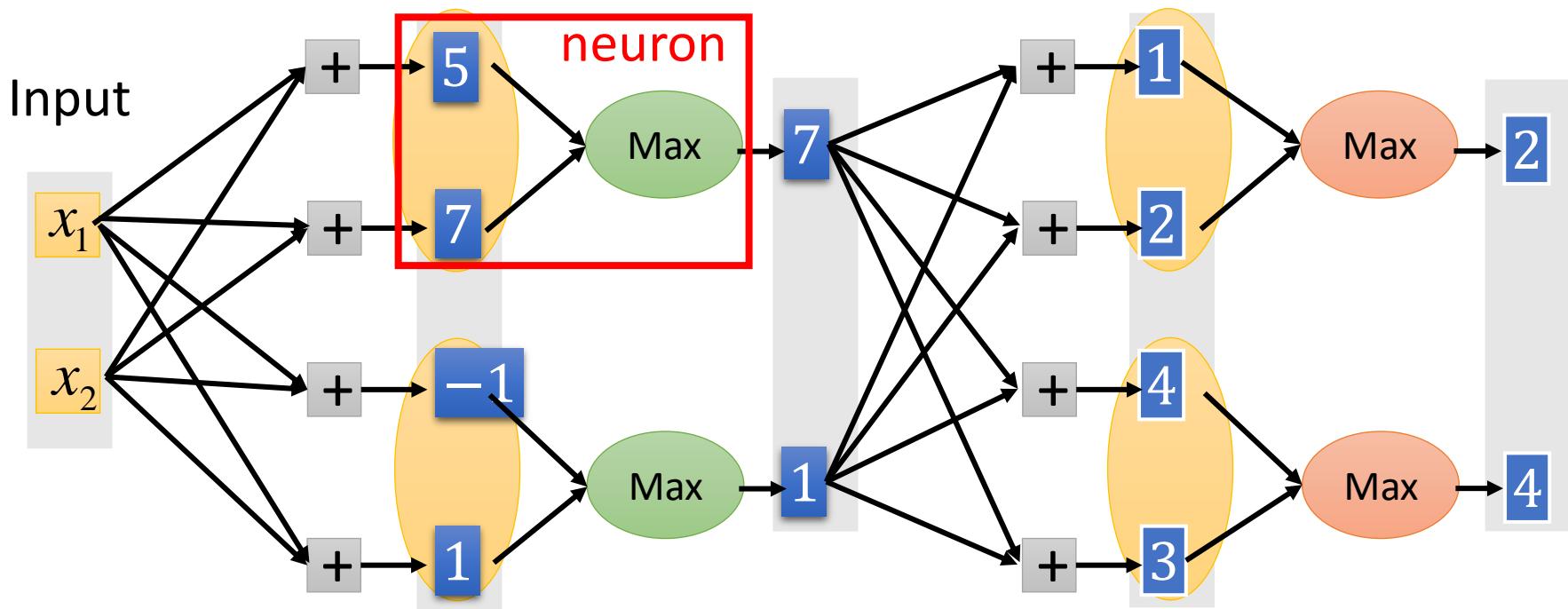
A Thinner linear network



Maxout

ReLU is a special cases of Maxout

- Learnable activation function [Ian J. Goodfellow, ICML'13]



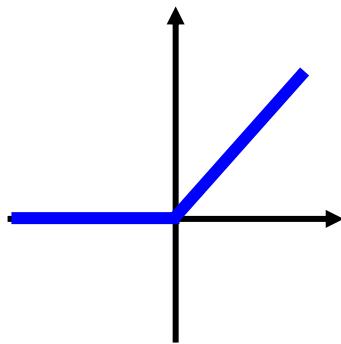
You can have more than 2 elements in a group.

Maxout

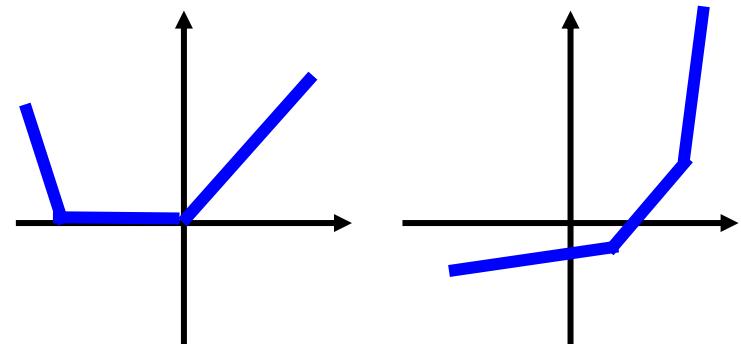
ReLU is a special cases of Maxout

- Learnable activation function [Ian J. Goodfellow, ICML'13]
 - Activation function in maxout network can be any piecewise linear convex function
 - How many pieces depending on how many elements in a group

2 elements in a group



3 elements in a group



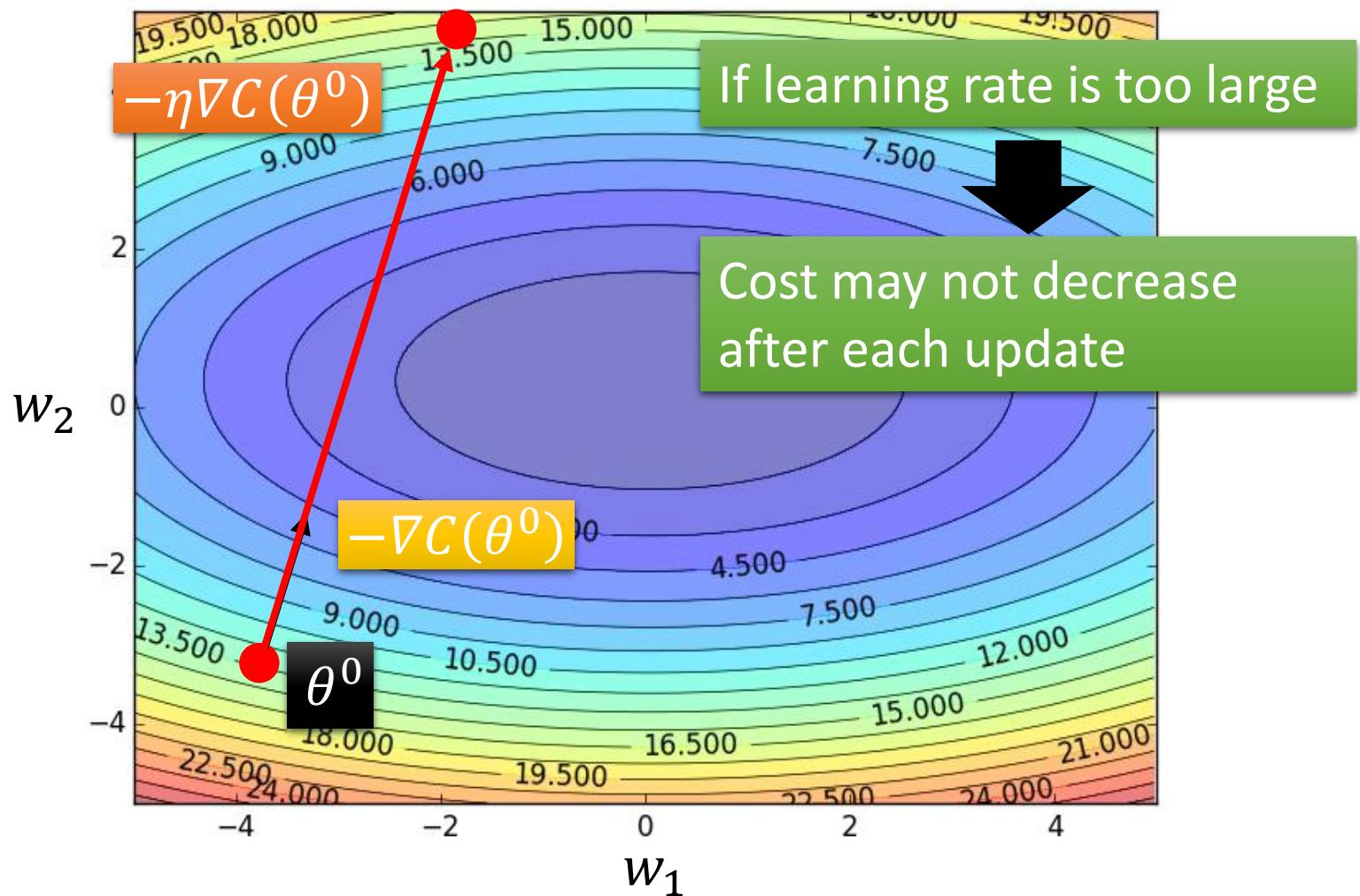
Machine Learning In Finance

Deep Learning Training

6.3.2 Adaptive Learning Rate

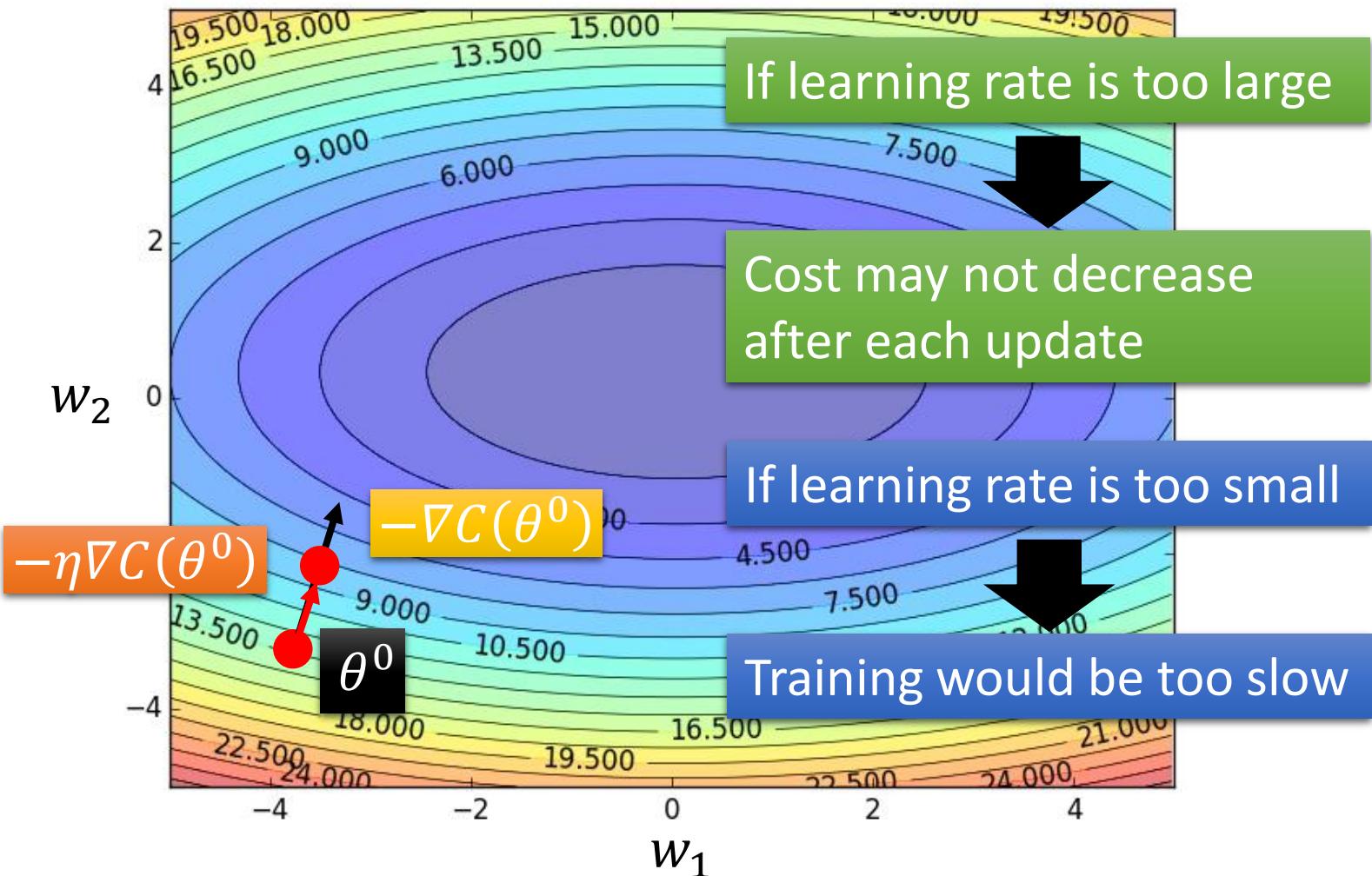
Learning Rate

Set the learning rate η carefully



Learning Rate

Can we give different parameters different learning rates?



Adagrad

Original Gradient Descent

$$\theta^t \leftarrow \theta^{t-1} - \eta \nabla C(\theta^{t-1})$$

Each parameter w are considered separately

$$w^{t+1} \leftarrow w^t - \eta_w g^t \quad g^t = \frac{\partial C(\theta^t)}{\partial w}$$

Parameter dependent learning rate

$$\eta_w = \frac{\eta}{\sqrt{\sum_{i=0}^t (g^i)^2}}$$

constant

Summation of the square of the previous derivatives

Adagrad

$$\eta_w = \frac{\eta}{\sqrt{\sum_{i=0}^t (g^i)^2}}$$

w_1	\mathbf{g}^0
	0.1

Learning rate:

$$\frac{\eta}{\sqrt{0.1^2}}$$

$$= \frac{\eta}{0.1}$$



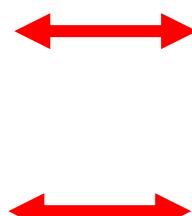
$$\frac{\eta}{\sqrt{0.1^2 + 0.2^2}} = \frac{\eta}{\sqrt{0.01 + 0.04}} = \frac{\eta}{\sqrt{0.05}} = \frac{\eta}{0.22}$$

w_2	\mathbf{g}^0
	20.0

Learning rate:

$$\frac{\eta}{\sqrt{20^2}}$$

$$= \frac{\eta}{20}$$



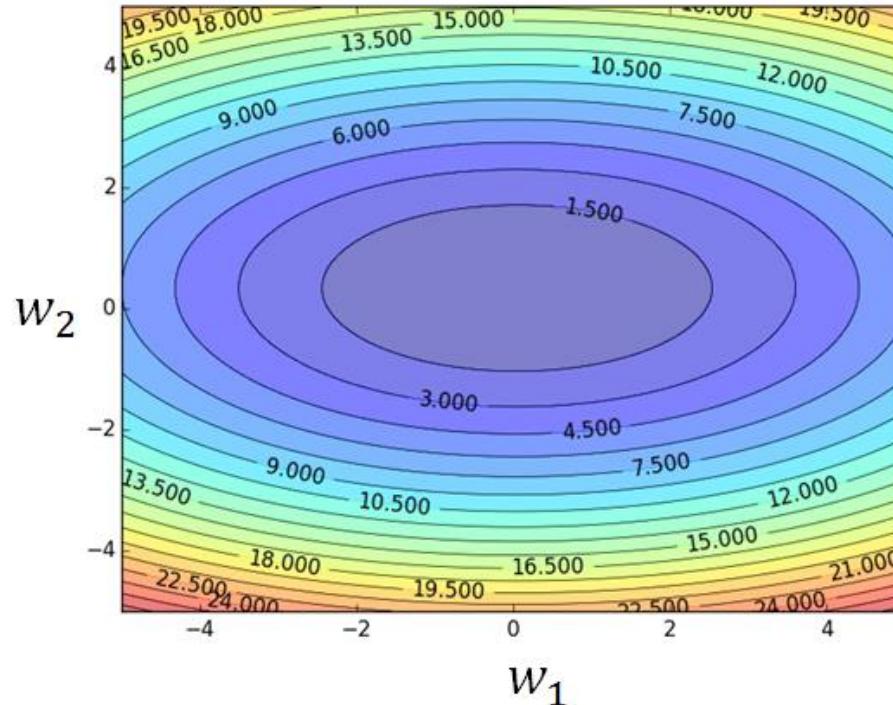
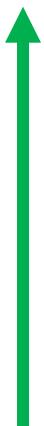
$$\frac{\eta}{\sqrt{20^2 + 10^2}} = \frac{\eta}{\sqrt{400 + 100}} = \frac{\eta}{\sqrt{500}} = \frac{\eta}{22}$$

- Observation:**
1. Learning rate is smaller and smaller for all parameters
 2. Smaller derivatives, larger learning rate, and vice versa

Why?

Larger derivatives

Smaller Learning Rate



Smaller Derivatives

Larger Learning Rate

2. Smaller derivatives, larger learning rate, and vice versa

Why?

Machine Learning In Finance

Deep Learning Training

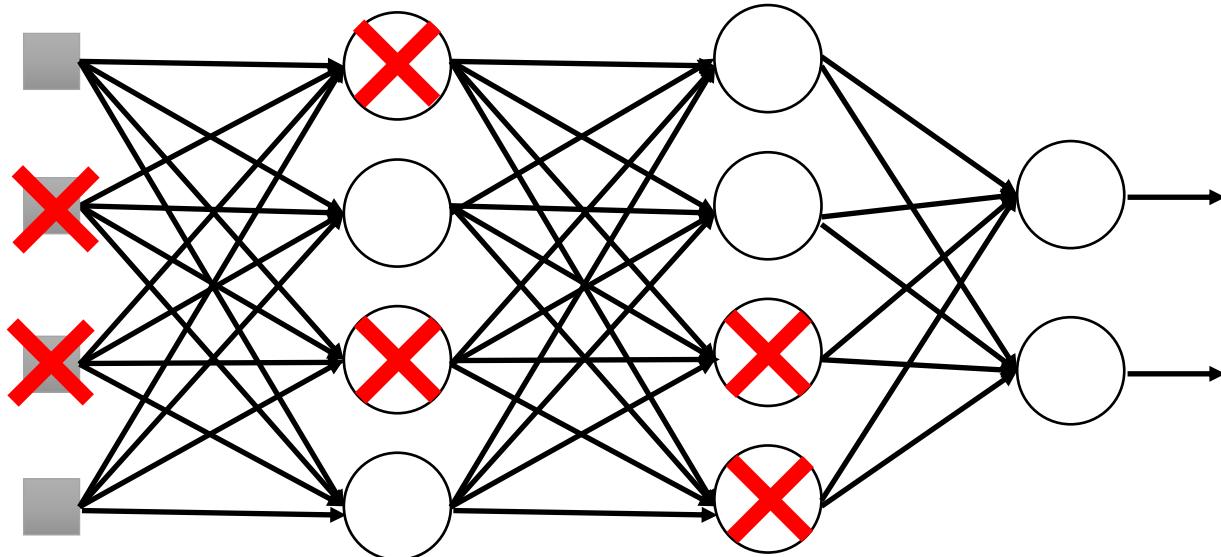
6.3.3 Drop Out

Dropout

Pick a mini-batch

$$\theta^t \leftarrow \theta^{t-1} - \eta \nabla C(\theta^{t-1})$$

Training:



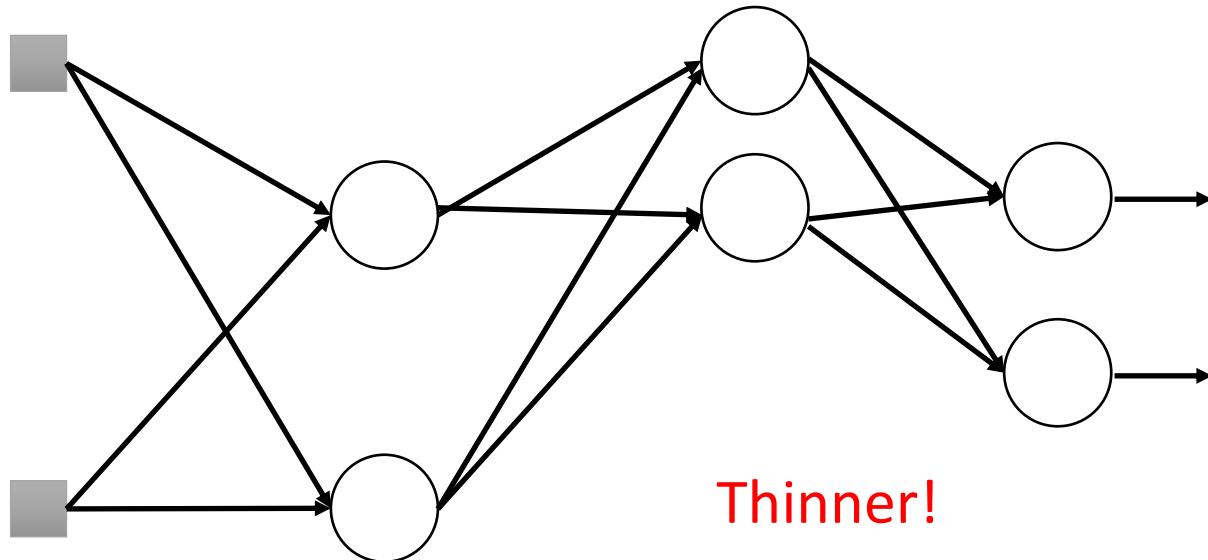
- **Each time before computing the gradients**
 - Each neuron has p% to dropout

Dropout

Pick a mini-batch

$$\theta^t \leftarrow \theta^{t-1} - \eta \nabla C(\theta^{t-1})$$

Training:

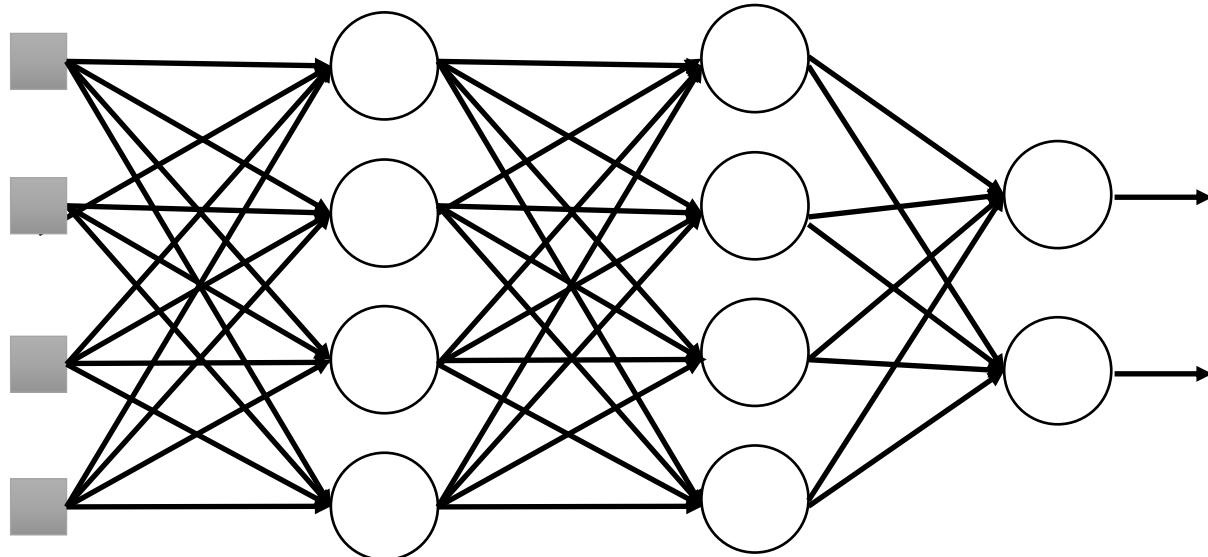


- **Each time before computing the gradients**
 - Each neuron has p% to dropout
 - ➡ **The structure of the network is changed.**
 - Using the new network for training

For each mini-batch, we resample the dropout neurons

Dropout

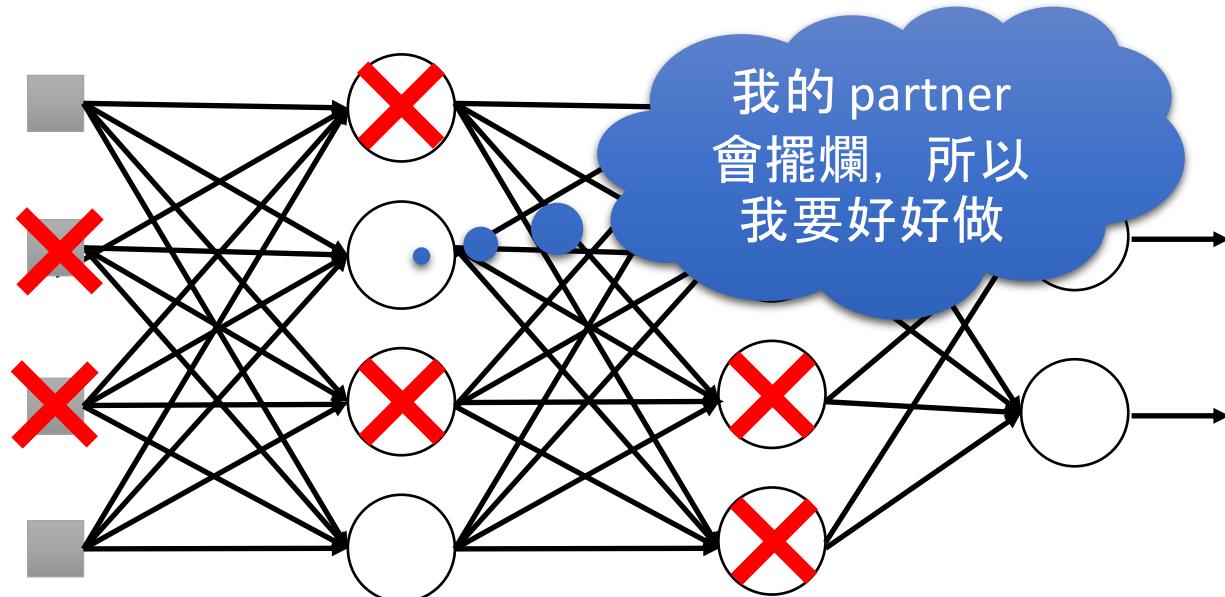
Testing:



➤ No dropout

- If the dropout rate at training is $p\%$,
all the weights times $(1-p)\%$
- Assume that the dropout rate is 50%.
If a weight $w = 1$ by training, set $w = 0.5$ for testing.

Dropout - Intuitive Reason



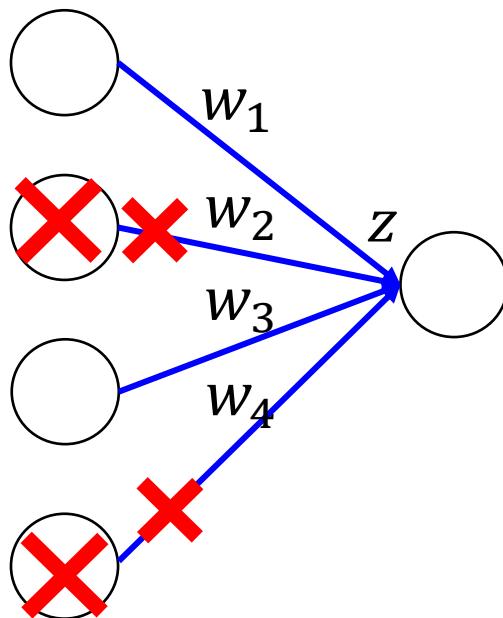
- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

Dropout - Intuitive Reason

- Why the weights should multiply $(1-p)\%$ (dropout rate) when testing?

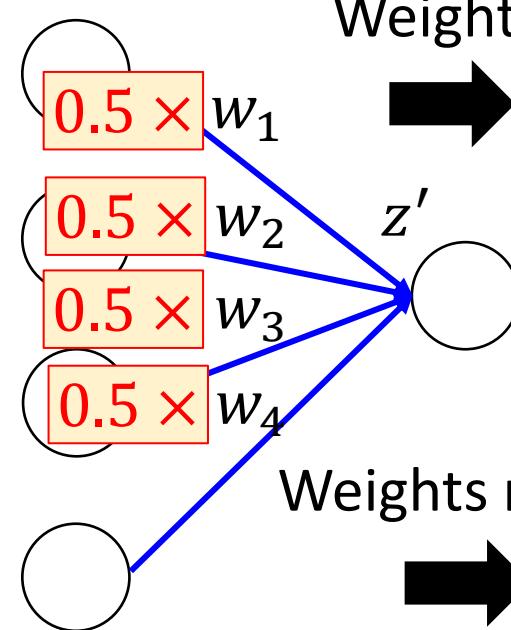
Training of Dropout

Assume dropout rate is 50%

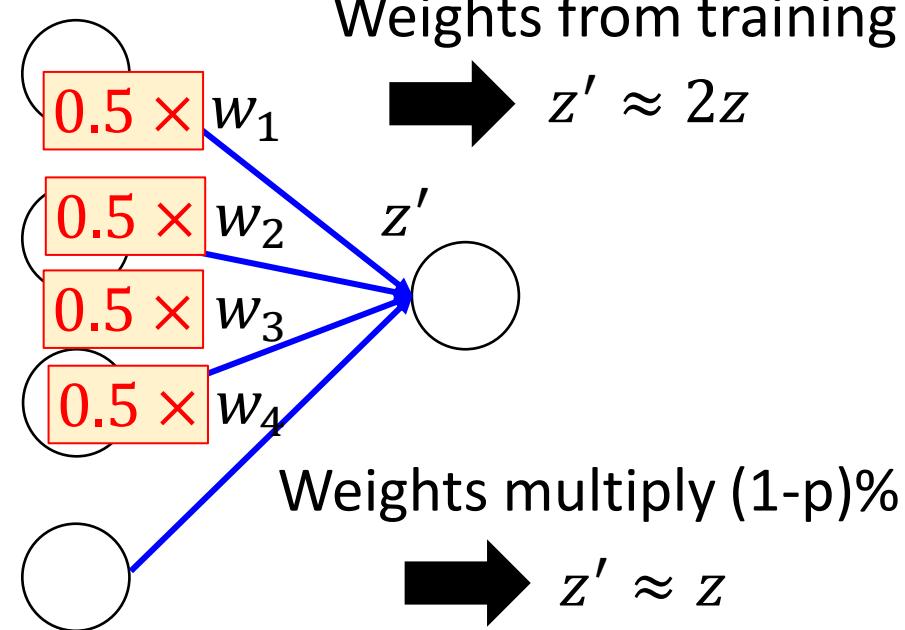


Testing of Dropout

No dropout



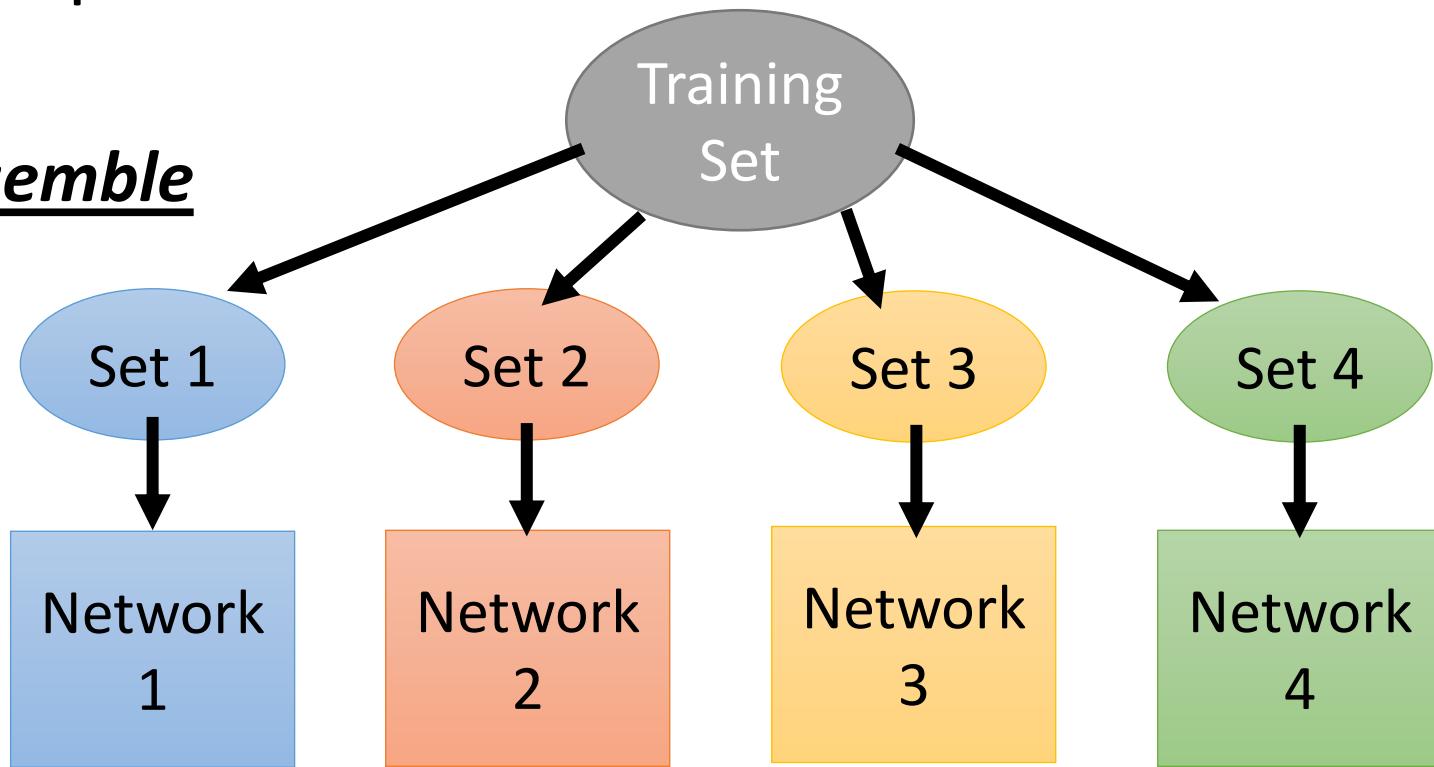
Weights from training



$\rightarrow z' \approx z$

Dropout is a kind of ensemble.

Ensemble



Train a bunch of networks with different structures

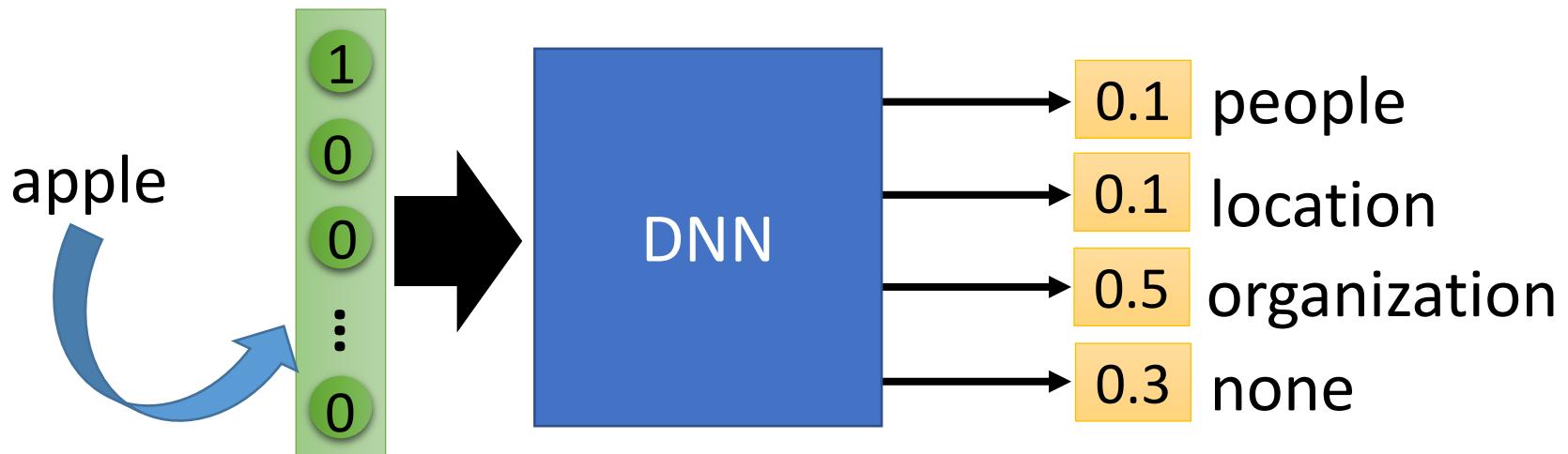
Machine Learning In Finance

Deep Learning Training

6.4 Neural Networks with Memory

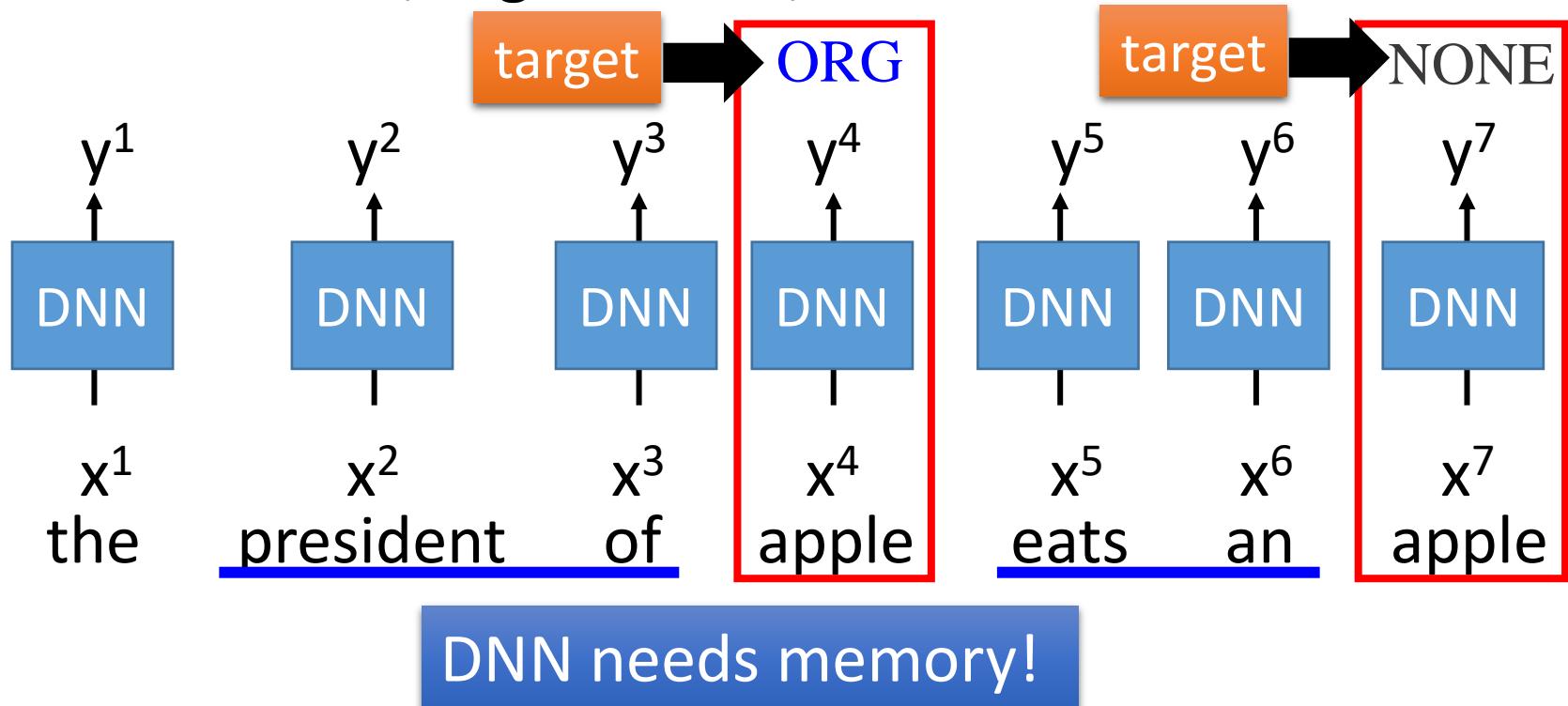
Neural Network needs Memory

- Name Entity Recognition
 - Detecting named entities like name of people, locations, organization, etc. in a sentence.



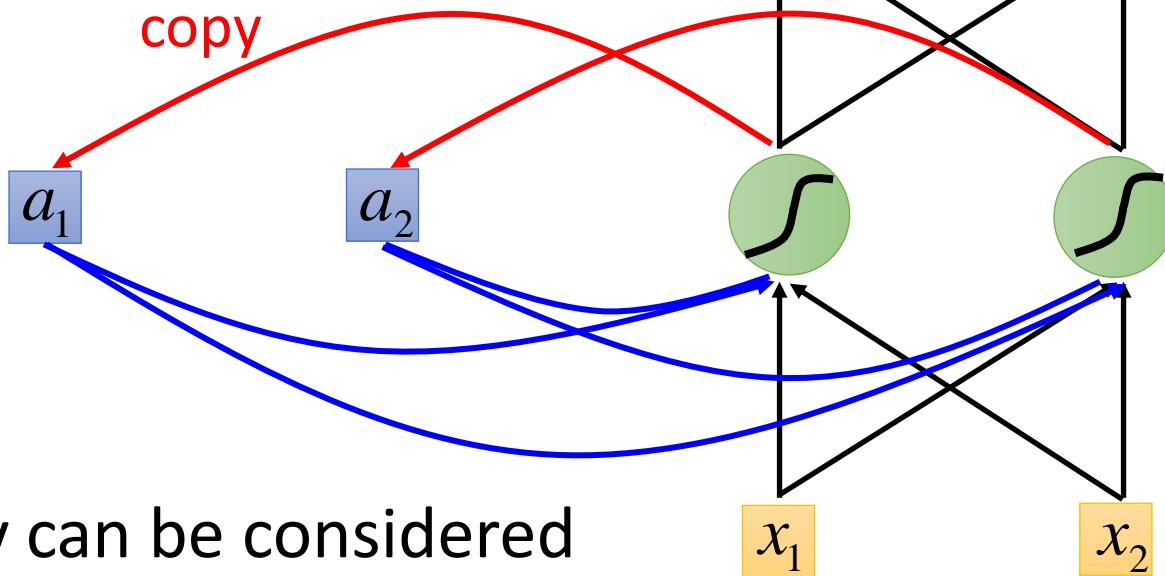
Neural Network needs Memory

- Name Entity Recognition
 - Detecting named entities like name of people, locations, organization, etc. in a sentence.



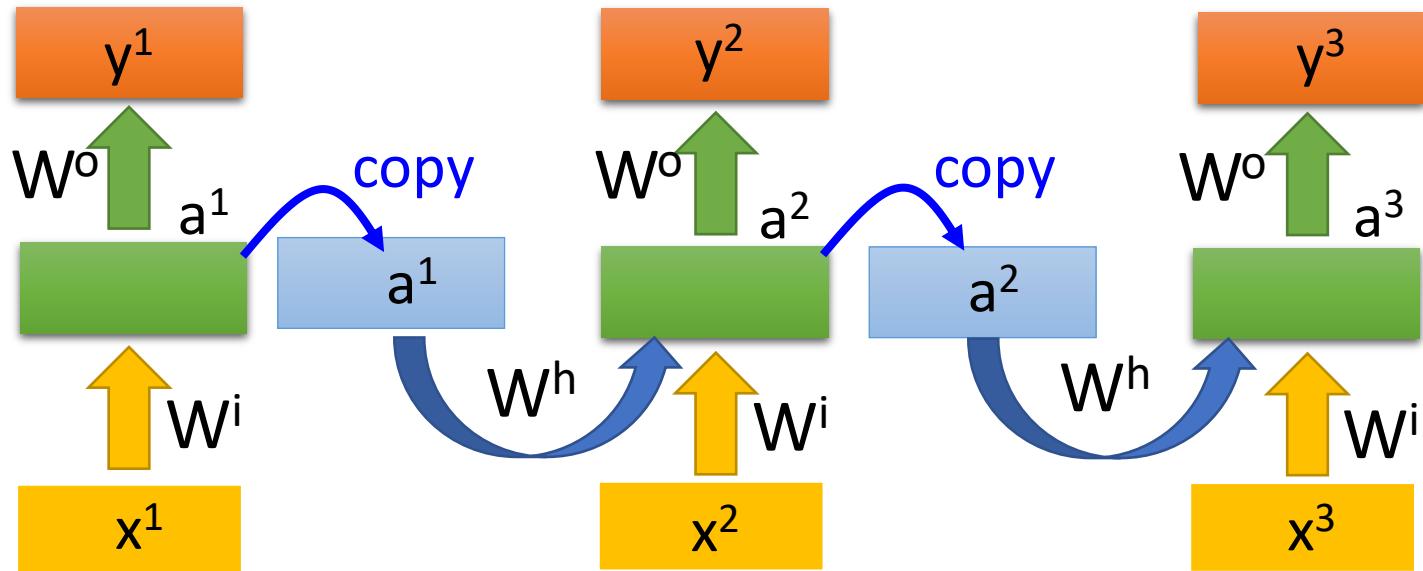
Recurrent Neural Network (RNN)

The output of hidden layer
are stored in the memory.



Memory can be considered
as another input.

RNN

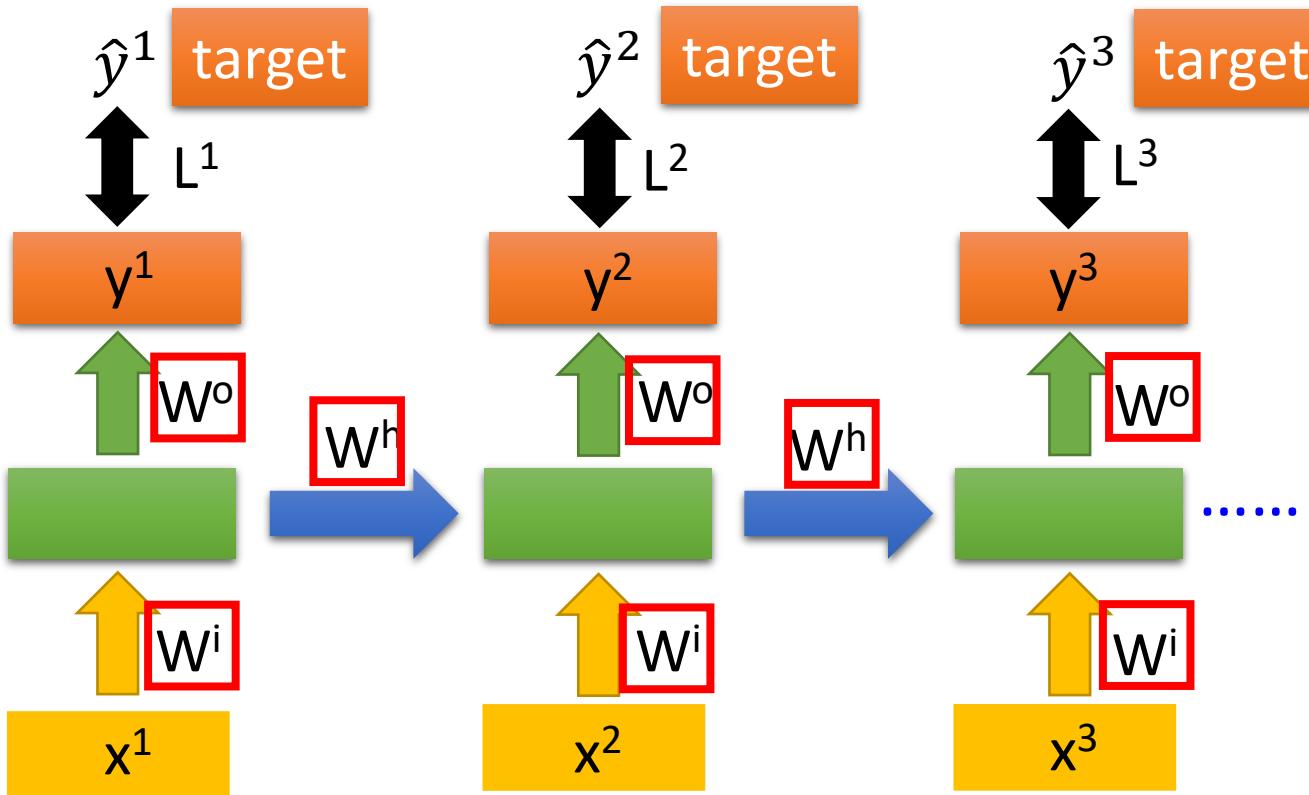


The same network is used again and again.

Output y^i depends on x^1, x^2, \dots, x^i

RNN

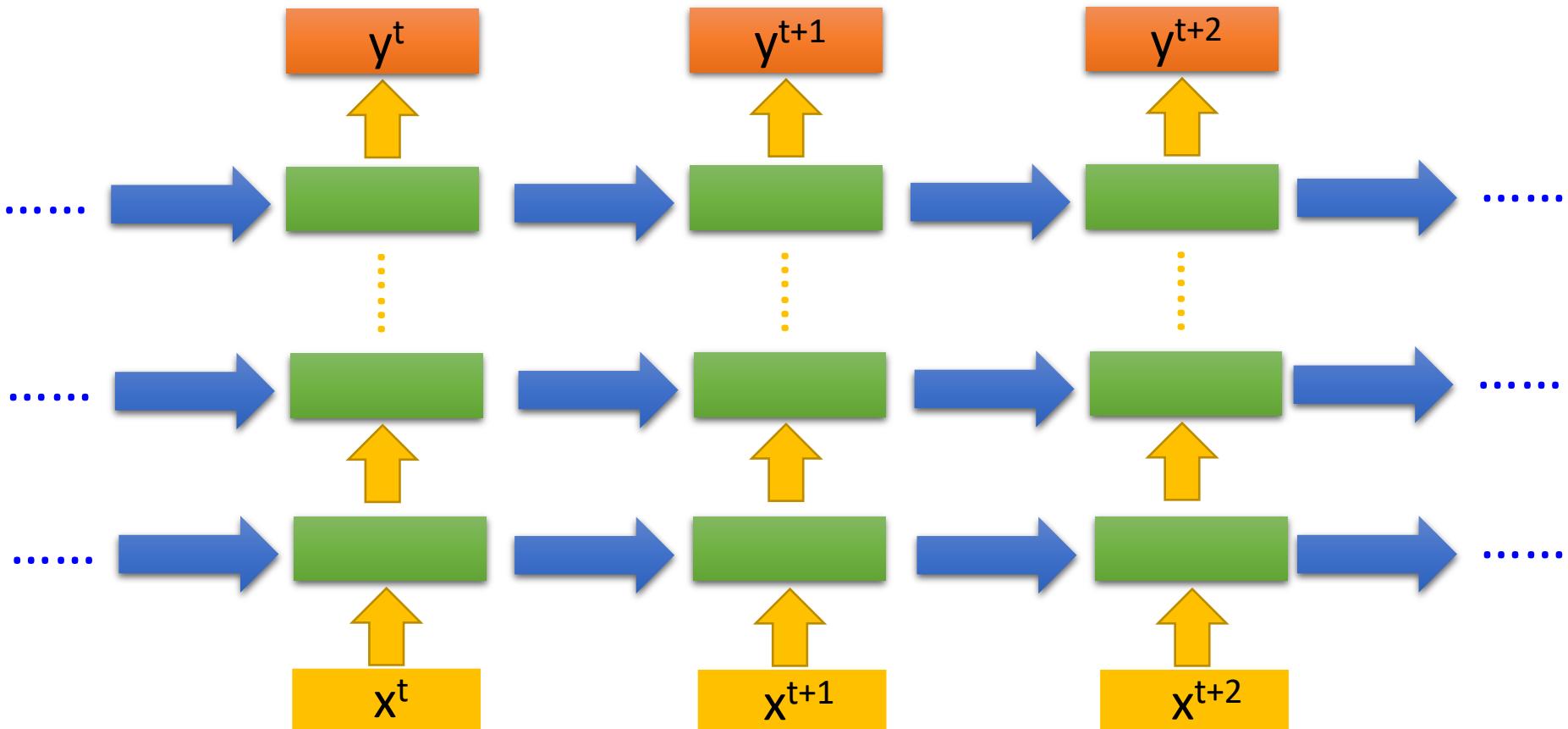
How to train?



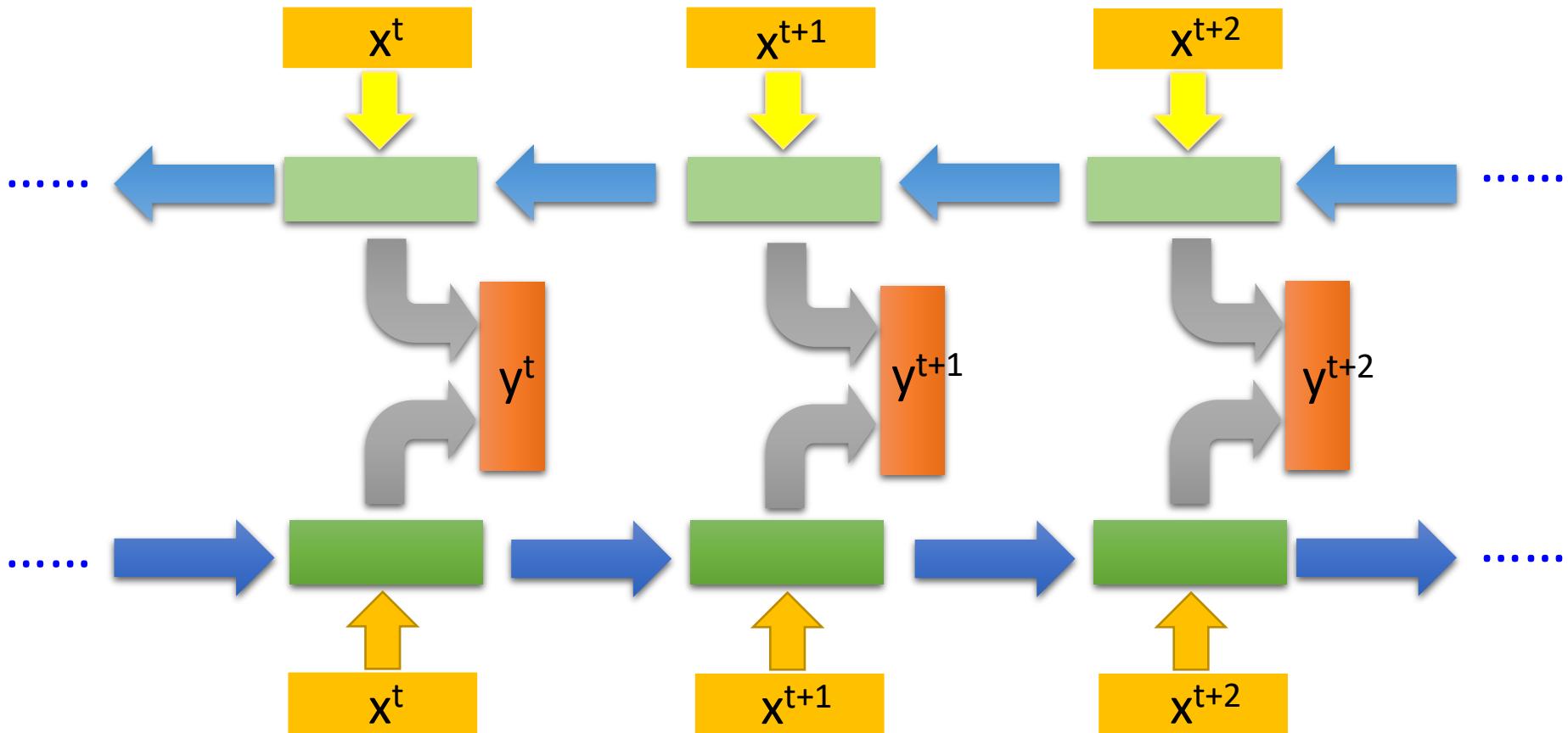
Find the network parameters to minimize the total cost:

Backpropagation through time (BPTT)

Of course it can be deep ...



Bidirectional RNN



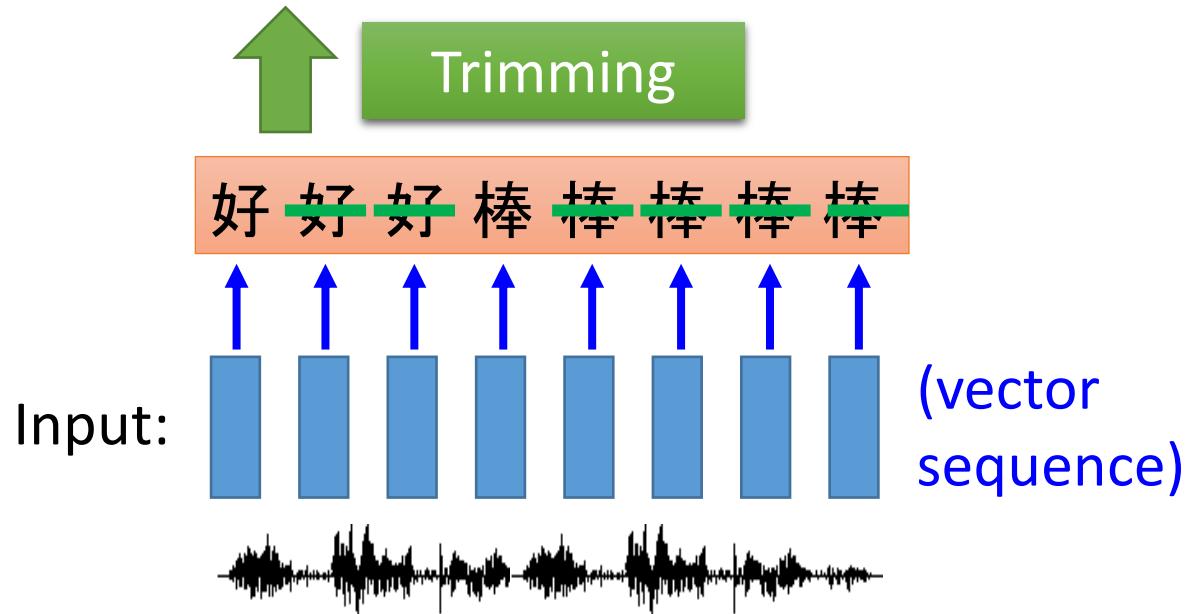
Many to Many (Output is shorter)

- Both input and output are both sequences, **but the output is shorter.**
 - E.g. **Speech Recognition**

Problem?

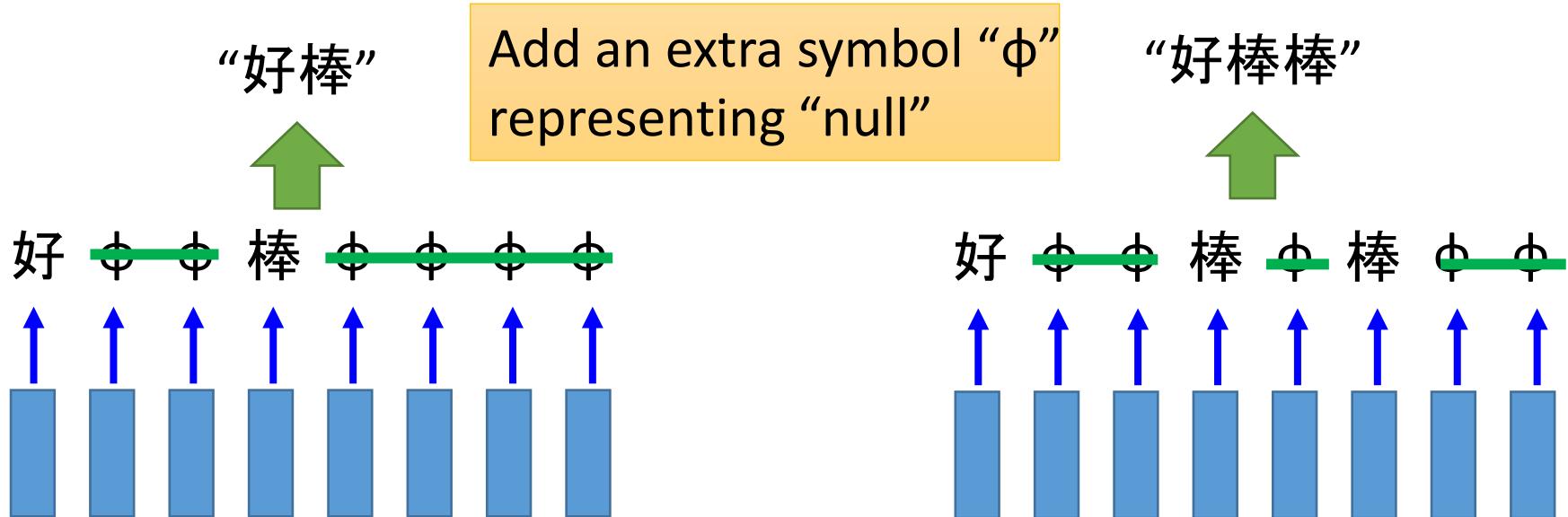
Why can't it be
“好棒棒”

Output: “好棒” (character sequence)



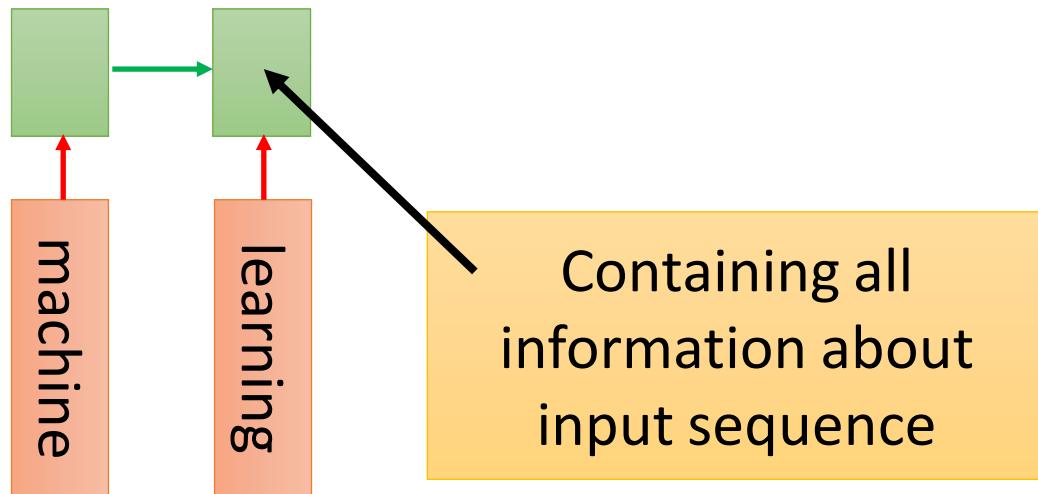
Many to Many (Output is shorter)

- Both input and output are both sequences, **but the output is shorter.**
- Connectionist Temporal Classification (CTC) [Alex Graves, ICML'06][Alex Graves, ICML'14][Hasim Sak, Interspeech'15][Jie Li, Interspeech'15][Andrew Senior, ASRU'15]



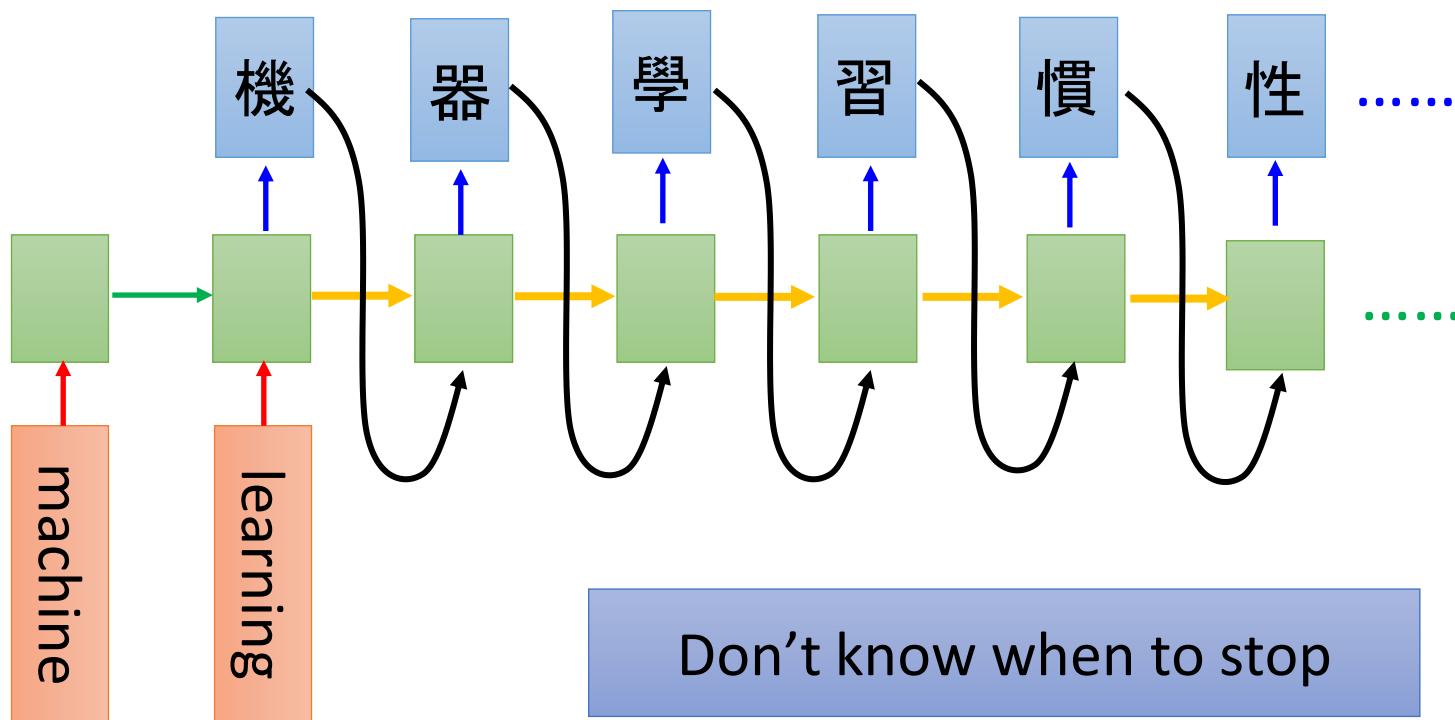
Many to Many (No Limitation)

- Both input and output are both sequences with different lengths. → Sequence to sequence learning
 - E.g. Machine Translation (machine learning→機器學習)



Many to Many (No Limitation)

- Both input and output are both sequences with different lengths. → Sequence to sequence learning
 - E.g. Machine Translation (machine learning→機器學習)



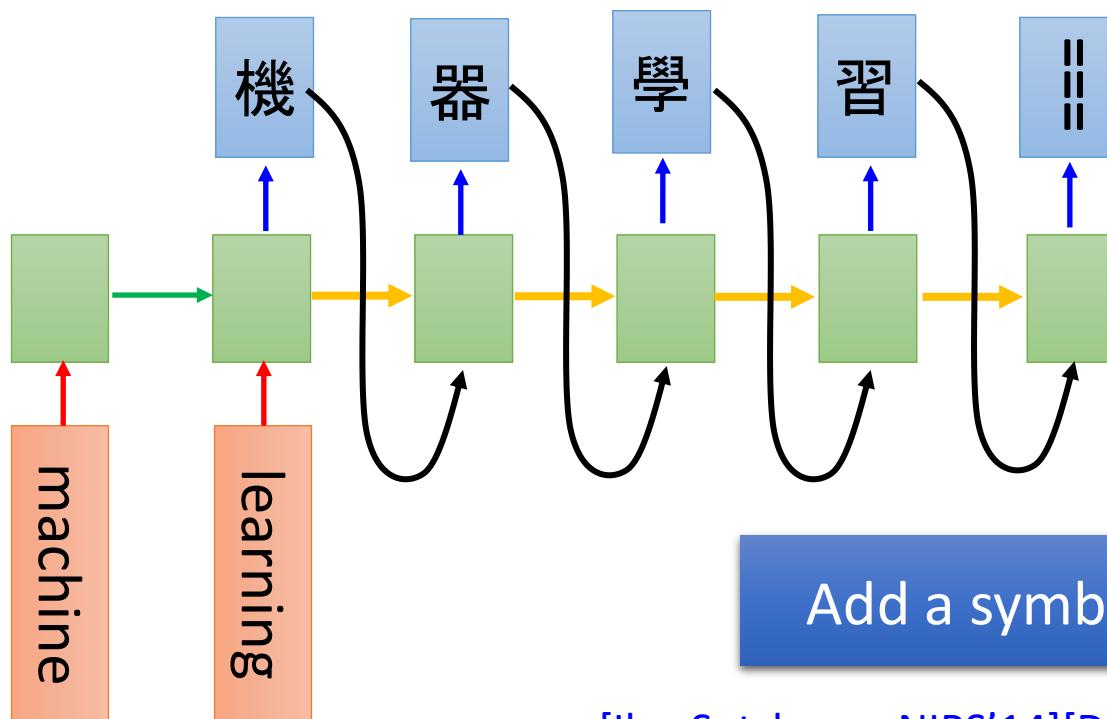
Many to Many (No Limitation)

推	: 超	06/12 10:39
推	: n: 人	06/12 10:40
推	: tion: 正	06/12 10:41
→	: host: 大	06/12 10:47
推	: 中	06/12 10:59
推	: 403: 天	06/12 11:11
推	: 外	06/12 11:13
推	: 527: 飛	06/12 11:17
→	: 990b: 仙	06/12 11:32
→	: 512: 草	06/12 12:15

推 tlkagk: =====斷=====

Many to Many (No Limitation)

- Both input and output are both sequences with different lengths. → Sequence to sequence learning
 - E.g. Machine Translation (machine learning→機器學習)



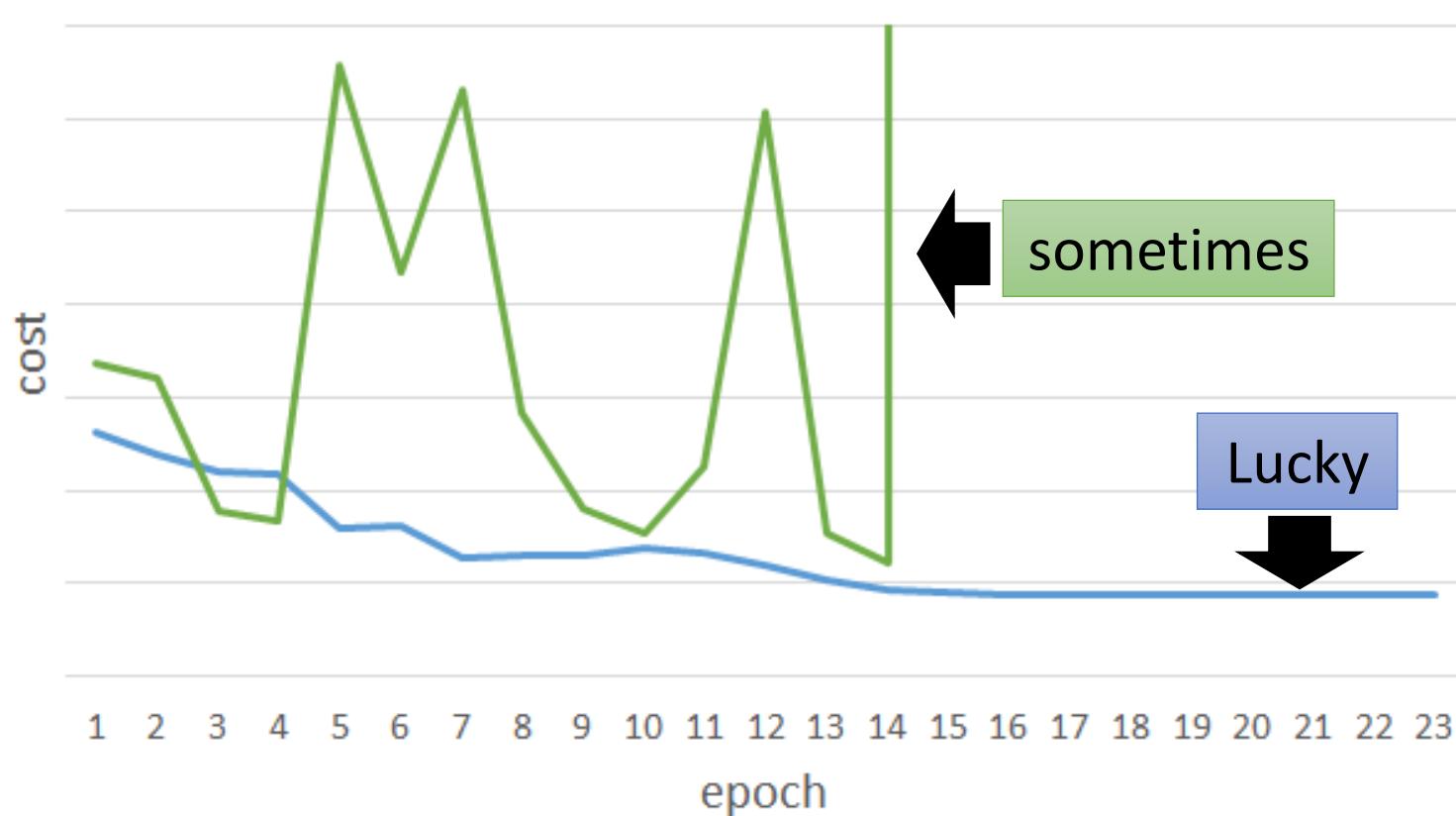
Add a symbol “==” (斷)

[Ilya Sutskever, NIPS'14][Dzmitry Bahdanau, arXiv'15]

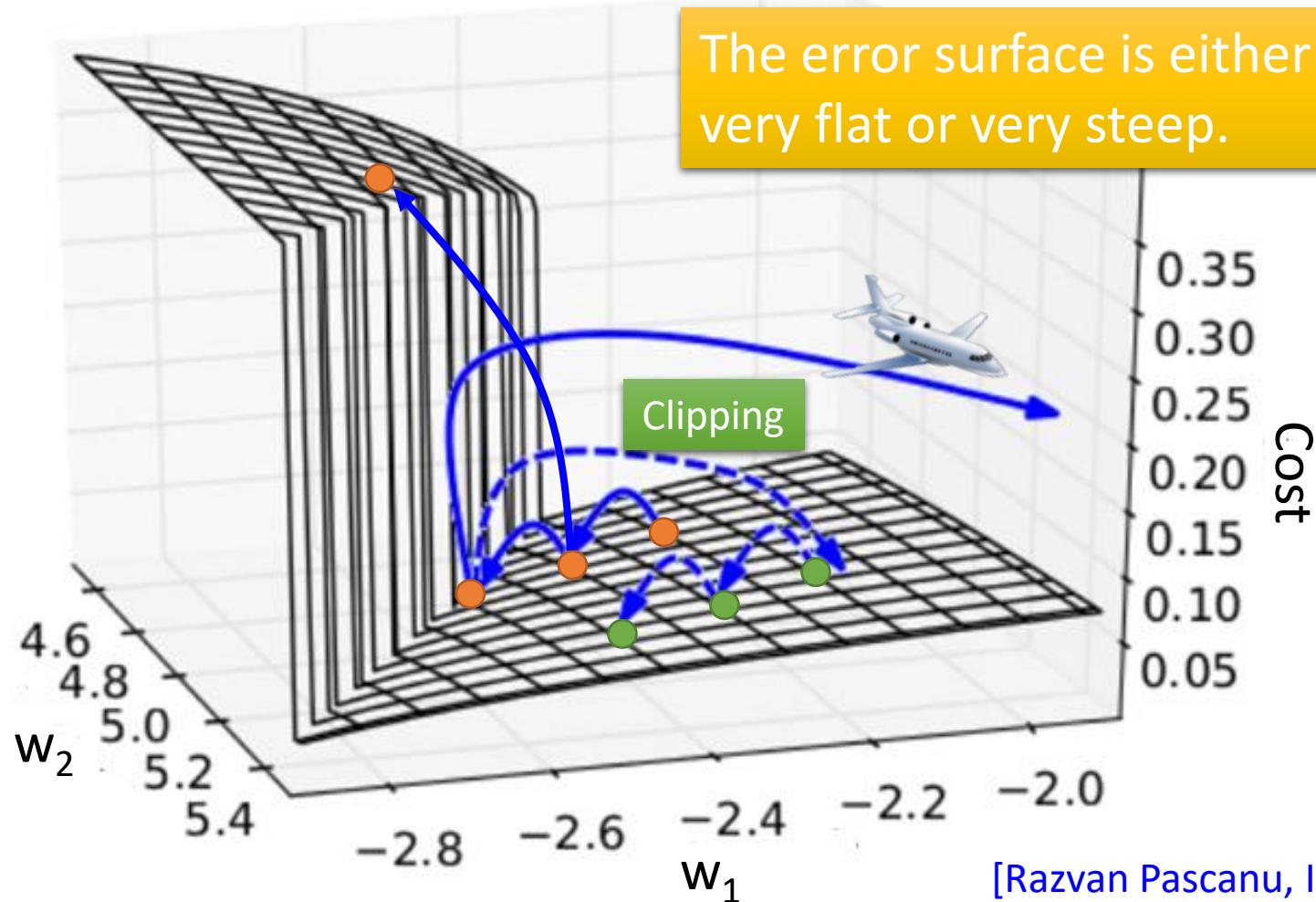
Unfortunately

- RNN-based network is not always easy to learn

Real experiments on Language modeling



The error surface is rough.



Why?

$$w = 1 \quad \rightarrow \quad y^{1000} = 1$$

$$w = 1.01 \quad \rightarrow \quad y^{1000} \approx 20000$$

Large gradient

Small Learning rate?

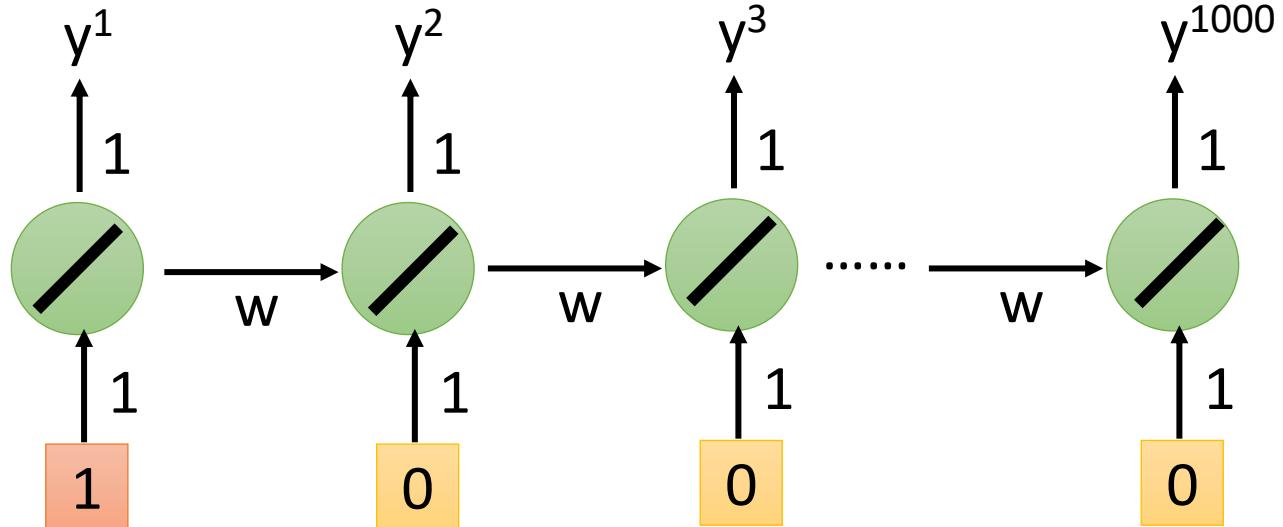
$$w = 0.99 \quad \rightarrow \quad y^{1000} \approx 0$$

$$w = 0.01 \quad \rightarrow \quad y^{1000} \approx 0$$

small gradient

Large Learning rate?

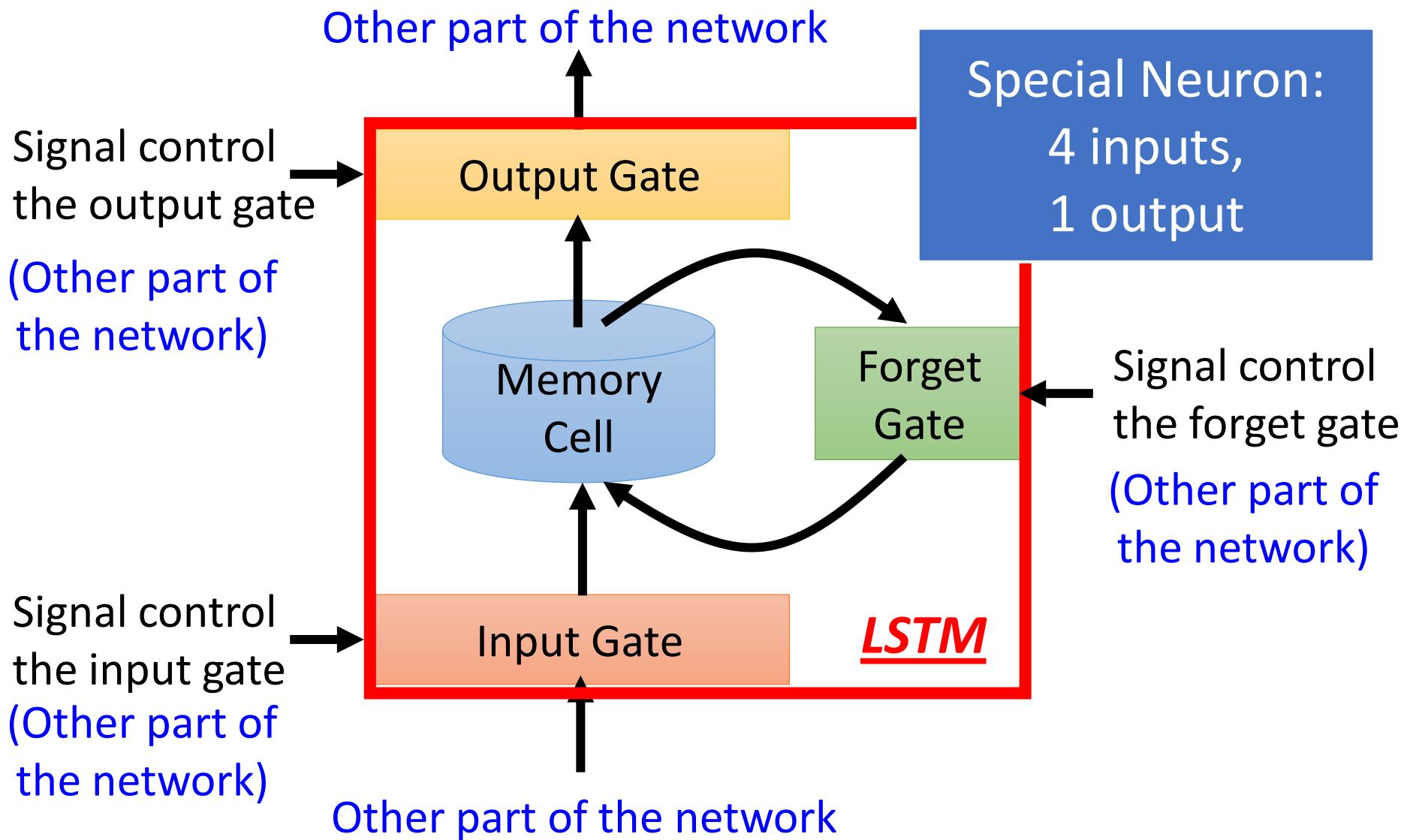
Toy Example



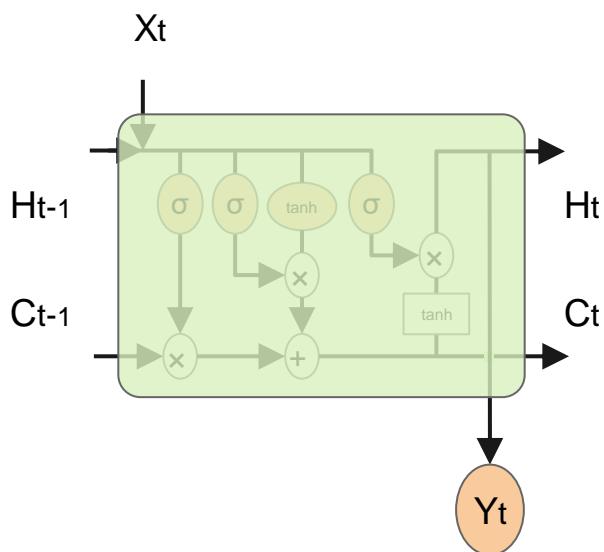
Helpful Techniques

- Nesterov's Accelerated Gradient (NAG):
 - Advance momentum method
- RMS Prop
 - Advanced approach to give each parameter different learning rates
 - Considering the change of Second derivatives
- Long Short-term Memory (LSTM)
 - Can deal with gradient vanishing (not gradient explode)

Long Short-term Memory (LSTM)



LSTM



concatenate : $x = x_t \mid H_{t-1}$

vector sizes
p+
n

forget gate : $f = \sigma(X.W_f + b_f)$

update gate : $u = \sigma(X.W_u + b_u)$

result gate : $r = \sigma(X.W_r + b_r)$

input : $X' = \tanh(X.W_c + b_c)$

new C : $C_t = f * C_{t-1} + u * X'$

new H : X'

$H_t = r * \tanh(C_t)$

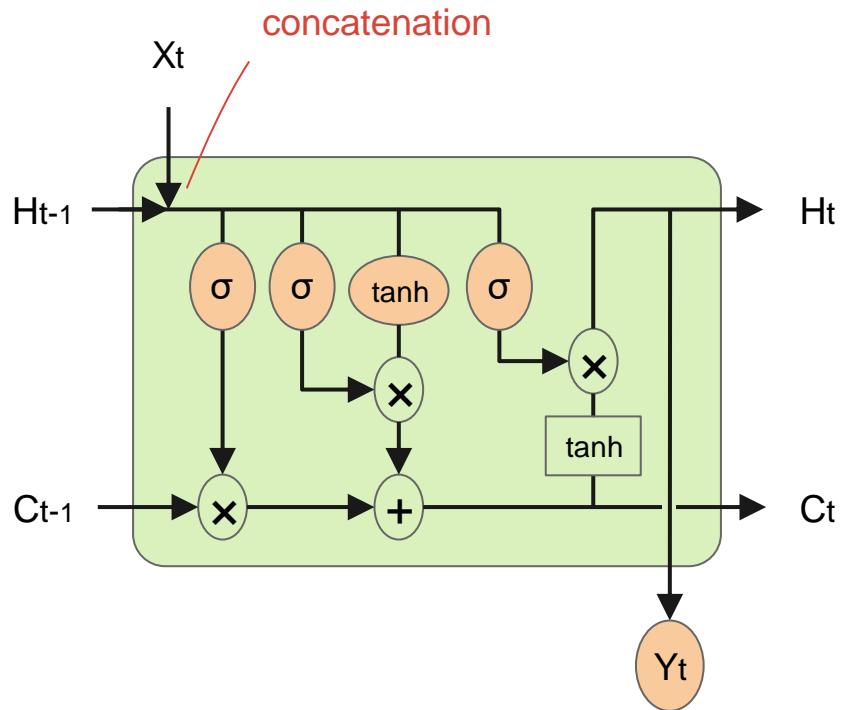
$y_t = \text{softmax}(H_t.W + b)$

m

output :

LSTM

LSTM = Long Short Term Memory



$$x = x_t \mid H_{t-1}$$

$$f = \sigma(x \cdot W_f + b_f)$$

$$u = \sigma(x \cdot W_u + b_u)$$

$$r = \sigma(x \cdot W_r + b_r)$$

$$X' = \tanh(x \cdot W_c + b_c)$$

$$C_t = f * C_{t-1} + u * X'$$

$$X'$$

$$H_t = r * \tanh(C_t)$$

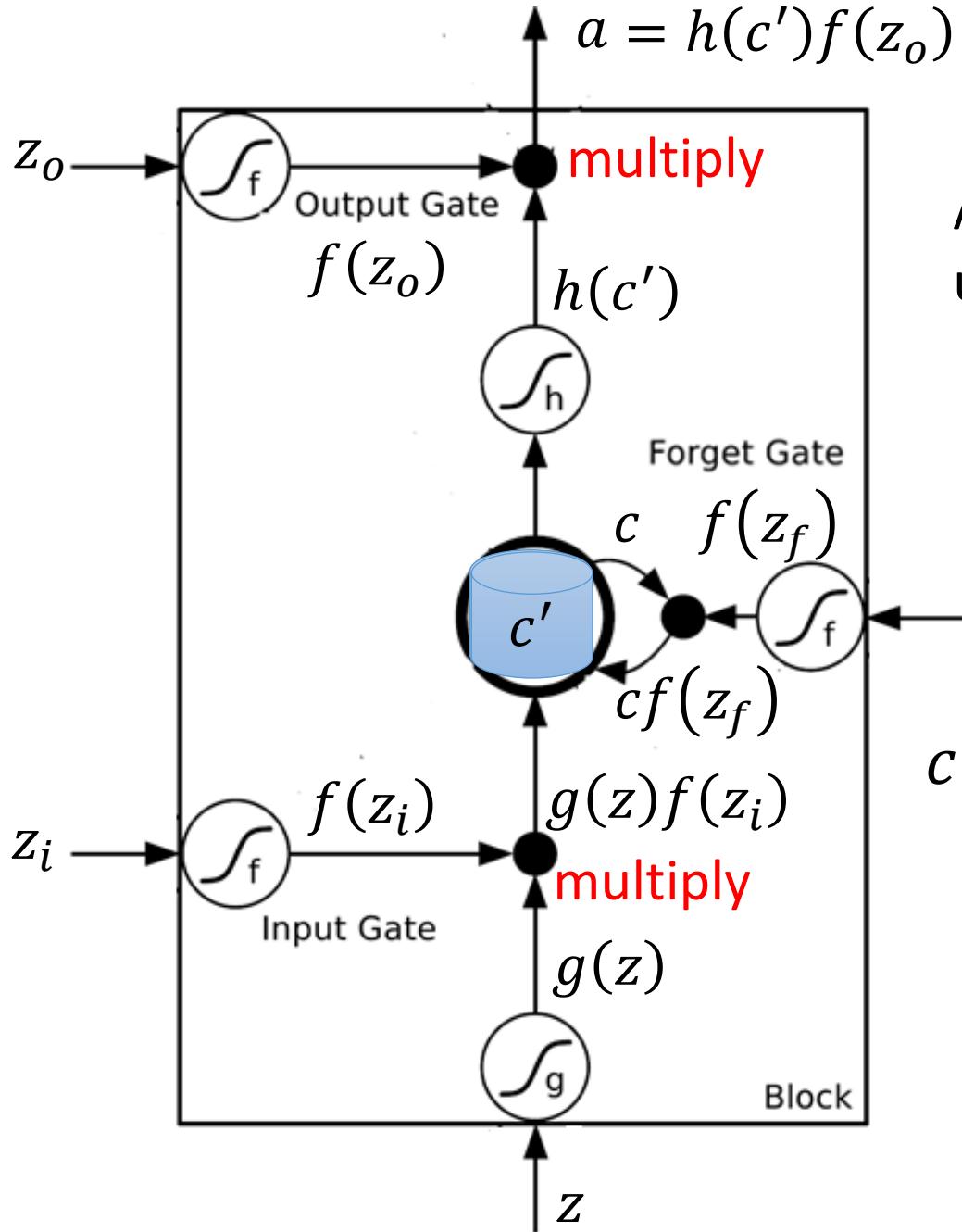
$$Y_t = \text{softmax}(H_t \cdot W + b)$$



Neural net. layers



Element-wise operations



Activation function f is usually a sigmoid function

Between 0 and 1

Mimic open and close gate

$$c' = g(z)f(z_i) + cf(z_f)$$

Machine Learning in Finance

Deep Learning

6.5 Applications

Deep Portfolio Theory - Deep Portfolio Theory – Heaton, Polson and Witte

Deep portfolio theory (DPT) provides a self-contained procedure for portfolio selection. We use training data to uncover deep feature policies (DFPs) in an auto-encoding step which fits the large data set of historical returns. In the decode step, we show how to find a portfolio-map to achieve a pre-specified goal. Both procedures involve an optimization problem with the need to choose the amount of regularization. To do this, we use an out-of-sample validation step which we summarize in an efficient deep portfolio frontier. Specifically, we avoid the use of statistical models that can be subject to model risk, and, rather than an *ex ante* efficient frontier, we judge the amount of regularization – which quantifies the number of deep layers and depth of our hidden layers – via the *ex post* efficient deep frontier.

Our approach builds on the original Markowitz insight that the portfolio selection problem can be viewed as a trade-off solved within an optimization framework (Markowitz, 1952, 2006, de Finetti, 1941). Simply put, our theory is based on first encoding the *market information* and then decoding it to form a portfolio that is designed to achieve our goal.

Deep Portfolio Theory Construction

Assume that the available market data has been separated into two (or more for an iterative process) disjoint sets for training and validation, respectively, denoted by X and \hat{X} .

Our goal is to provide a self-contained procedure that illustrates the trade-offs involved in constructing portfolios to achieve a given goal, e.g., to beat a given index by a pre-specified level. The projected real-time success of such a goal will depend crucially on the market structure implied by our historical returns. We also allow for the case where conditioning variables, denoted by Z , are also available in our training phase. (These might include accounting information or further returns data in the form of derivative prices or volatilities in the market.)

Deep Portfolio Theory Construction

I. Auto-encoding

Find the *market-map*, denoted by $F_W^m(X)$, that solves the regularization problem

$$\min_W \|X - F_W^m(X)\|_2^2 \text{ subject to } \|W\| \leq L^m. \quad (1)$$

For appropriately chosen F_W^m , this auto-encodes X with itself and creates a more information-efficient representation of X (in a form of *pre-processing*).

II. Calibrating

For a desired result (or target) Y , find the *portfolio-map*, denoted by $F_W^p(X)$, that

solves the regularization problem

$$\min_W \|Y - F_W^p(X)\|_2^2 \text{ subject to } \|W\| \leq L^p. \quad (2)$$

This creates a (non-linear) portfolio from X for the approximation of objective Y .

Deep Portfolio Theory Construction

III. Validating

Find L^m and L^p to suitably balance the trade-off between the two errors

$$\epsilon_m = \|\hat{X} - F_{W_m^*}^m(\hat{X})\|_2^2 \quad \text{and} \quad \epsilon_p = \|\hat{Y} - F_{W_p^*}^p(\hat{X})\|_2^2,$$

where X_m^* and X_p^* are the solutions to (1) and (2), respectively.

IV. Verifying

Choose *market-map* F^m and *portfolio-map* F^p such that validation (step 3) is satisfactory. To do so, inspect the implied deep portfolio frontier for the goal of interest as a function of the amount of regularization provides such a metric.

Deep Factor Models versus Shallow Factor Models

Almost all **shallow data reduction** techniques can be viewed as consisting of a low dimensional auxiliary variable Z and a prediction rule specified by a composition of functions

$$\begin{aligned}\hat{Y} &= f_1^{W_1, b_1}(f_2(W_2 X + b_2)) \\ &= f_1^{W_1, b_1}(Z), \text{ where } Z := f_2(W_2 X + b_2).\end{aligned}$$

In this formulation, we also recognize the previously introduced deep learning structure (2.1). The problem of high dimensional data reduction in general is to find the Z -variable and to estimate the layer functions (f_1 ; f_2) correctly. In the layers, we want to uncover the low-dimensional Z -structure in a way that does not disregard information about predicting the output Y .

Principal component analysis (PCA), reduced rank regression (RRR), linear discriminant analysis (LDA), project pursuit regression (PPR), and logistic regression are all shallow learners. See Wold (1956), Diaconis and Shahshahani (1984), Ripley (1996), Cook (2007), and Hastie et al. (2009) for further discussion.

Machine Learning in Finance

Deep Learning

6.5.1 Auto – Encoders Smart Indexing

Smart Indexing

When aiming to replicate (or approximate) a stock index through a subset of stocks, there are two conceptual approaches we can choose from.

- (i) Identify a small group of stocks which historically have given a performance very similar to that of the observed index.
- (ii) Identify a small group of stocks which historically have represented an over-proportionally large part of the total aggregate information of all the stocks the index comprises of. While, on the face of things,
 - (i) and (ii) may appear very similar, they characterize, in fact, very different methodologies.

Many classic approaches to index replication are essentially rooted in linear-regression, which is part of group (i). Frequently, by trial and error, we are trying to find a small subset of stocks which in-sample gives a reasonable linear approximation of the considered index.

Smart Indexing

The deep learning version of (i) allows translating the input data through a hierarchical sequence of adaptive linear layers into a desired output, which means that, in training, even non-linear relationships can be readily identified. Since every hidden layer provides a new interpretation of the input features, we refer to the resulting strategy for approximation (or prediction) as a deep feature policy (DFP), an example of which is given in Figure 1.

The availability of tailored non-linear relationships in deep learning makes the conventional objective of (i), namely good in-sample approximation, an easily achieved triviality, and takes the focus straight to training for out-of-sample performance (which brings us back to cross-validation and dropout, see Section 2).

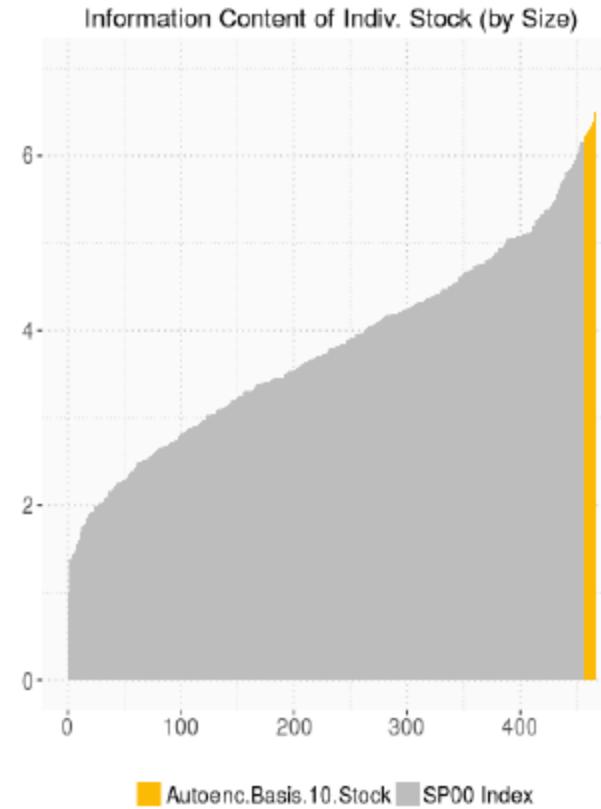
Another weakness of (i) is that it fits the finished (and through aggregation diluted) product.

A deep auto-encoder avoids this problem by directly (rather than indirectly) approximating the aggregate information contained in the considered family of index stocks. In Figure 2, a deep autoencoder for a small set of ten stocks is depicted. In Figure 3, we see the stock paths before and after compression.

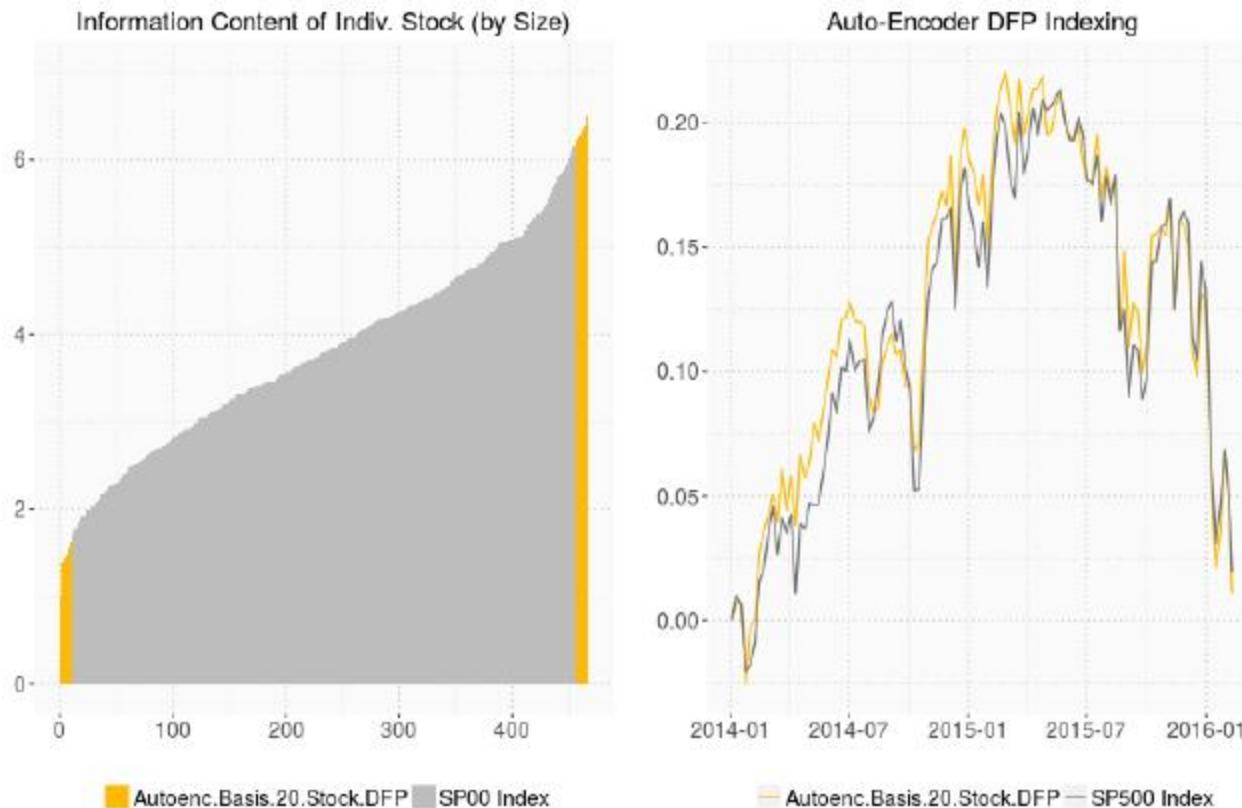
The bottleneck structure of an auto-encoder creates a compressed set of information from which all stocks are re-created (through linear and non-linear relationships). Thus, for indexing, the stocks which are closest to the compressed core of the index can be interpreted as a non-linear basis of the aggregate information of the considered family of stocks.

Smart Indexing

A deep auto-encoder compresses the stocks of the S&P500. Above, we rank all stocks by their proximity to the auto encoded information and create an equally weighted portfolio from the auto-encoder basis of the 10 leading stocks. Below, we use the leading 20 (the most communal stocks) and the bottom ten (most individualistic) stocks to create an auto-encoder basis on which we train a deep feature policy (DFP) for the approximation of the S&P500.

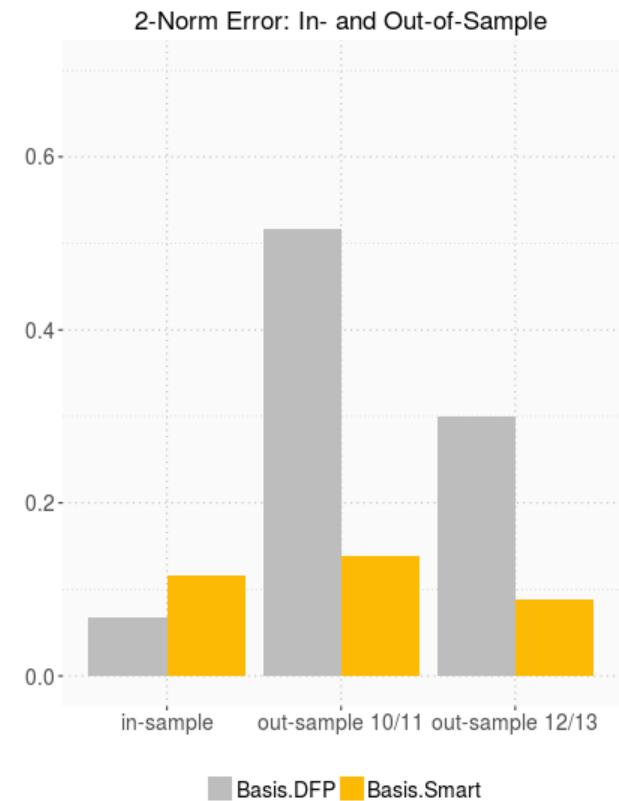


Smart Indexing



Smart Indexing

Having trained a 20 stock DFP auto encoder basis as well as a simpler 10 stock deep auto-encoder basis for the S&P500 on the two years 2014/15, we now consider the out-of-sample performance of the two approximations for the periods 2010/11 and 2012/13. We observe that the superior fitting qualities of the full DFP basis in-sample are out-weighed easily by the superior out-of-sample consistency of the generic deep auto-encoder basis.



Other applications

- Default probabilities
- Mortgage Risk
- LSTN time series
- Trading Events

Deep Neural Networks

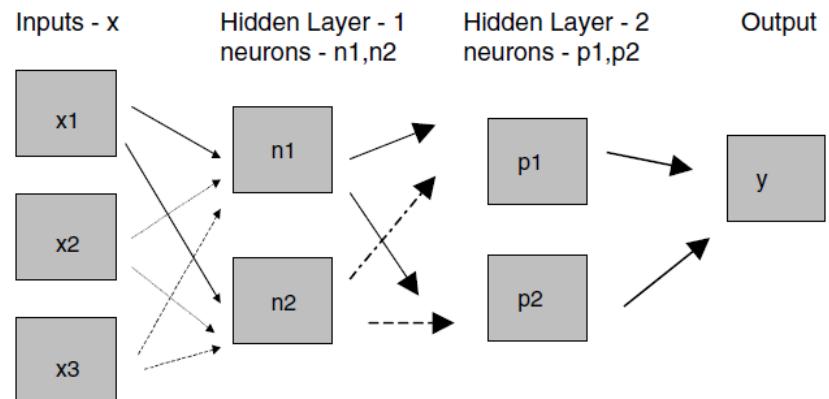
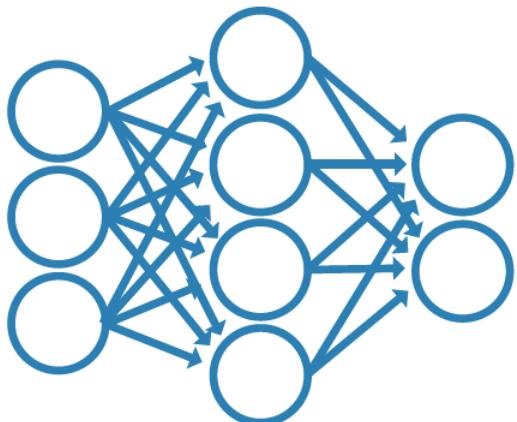
Neural Networks

How it Works

Inspired by the human brain, a neural network consists of highly connected networks of neurons that relate the inputs to the desired outputs. The network is trained by iteratively modifying the strengths of the connections so that given inputs map to the correct response.

Best Used...

- For modeling highly nonlinear systems
- When data is available incrementally and you wish to constantly update the model
- When there could be unexpected changes in your input data
- When model interpretability is not a key concern



$$n_{k,t} = \omega_{k,0} + \sum_{i=1}^{i^*} \omega_{k,i} x_{i,t}$$

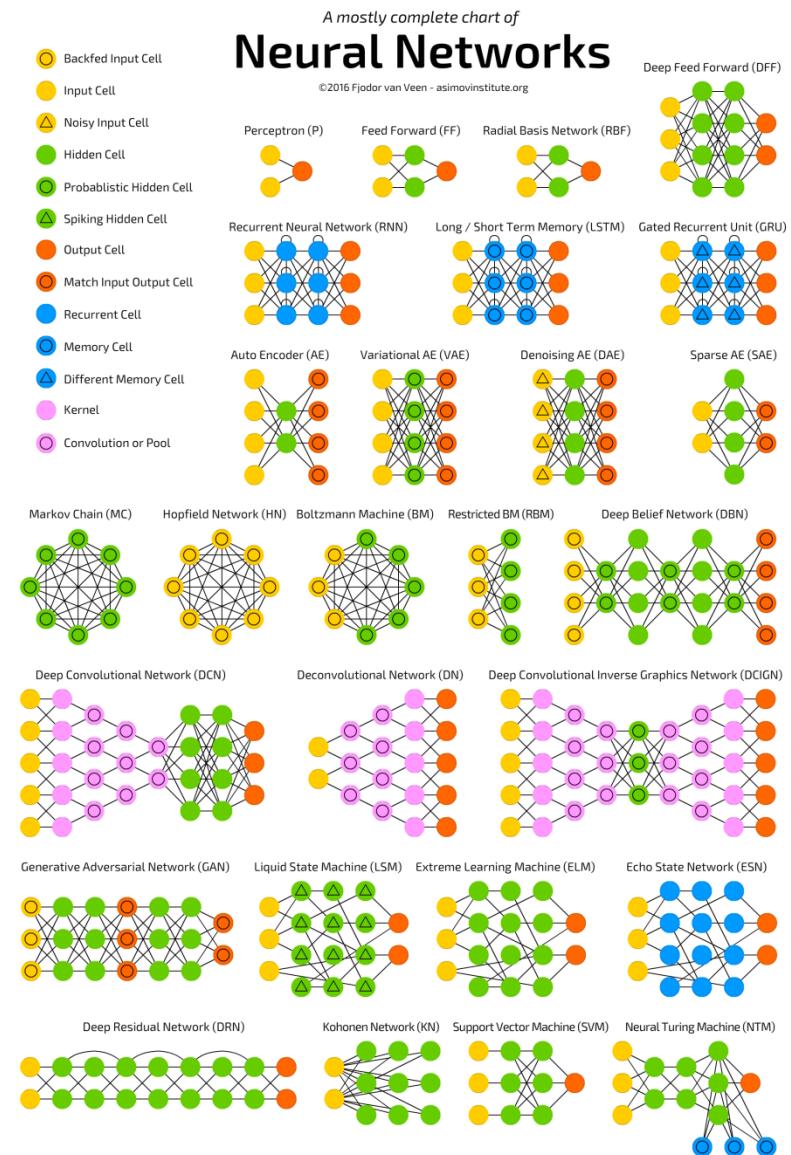
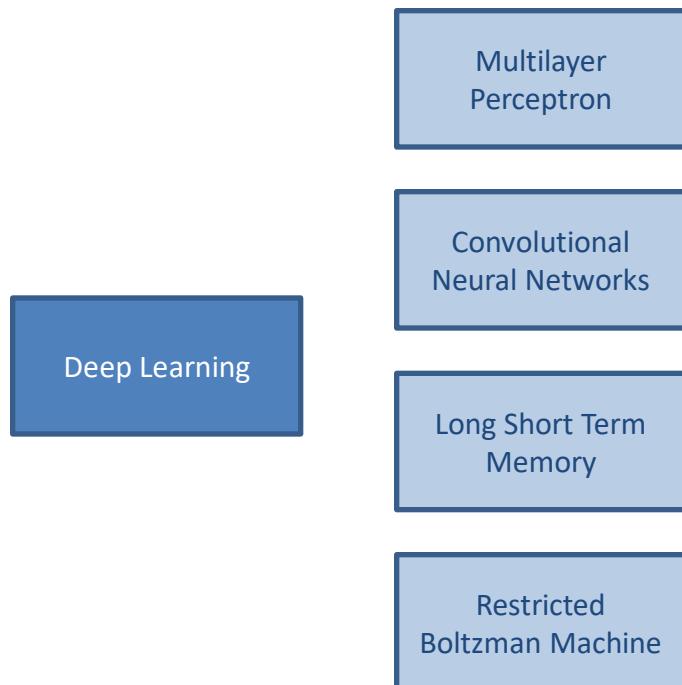
$$N_{k,t} = \frac{1}{1 + e^{-n_{k,t}}}$$

$$p_{l,t} = \rho_{l,0} + \sum_{k=1}^{k^*} \rho_{l,k} N_{k,t}$$

$$P_{l,t} = \frac{1}{1 + e^{-p_{l,t}}}$$

$$y_t = \gamma_0 + \sum_{l=1}^{l^*} \gamma_l P_{l,t}$$

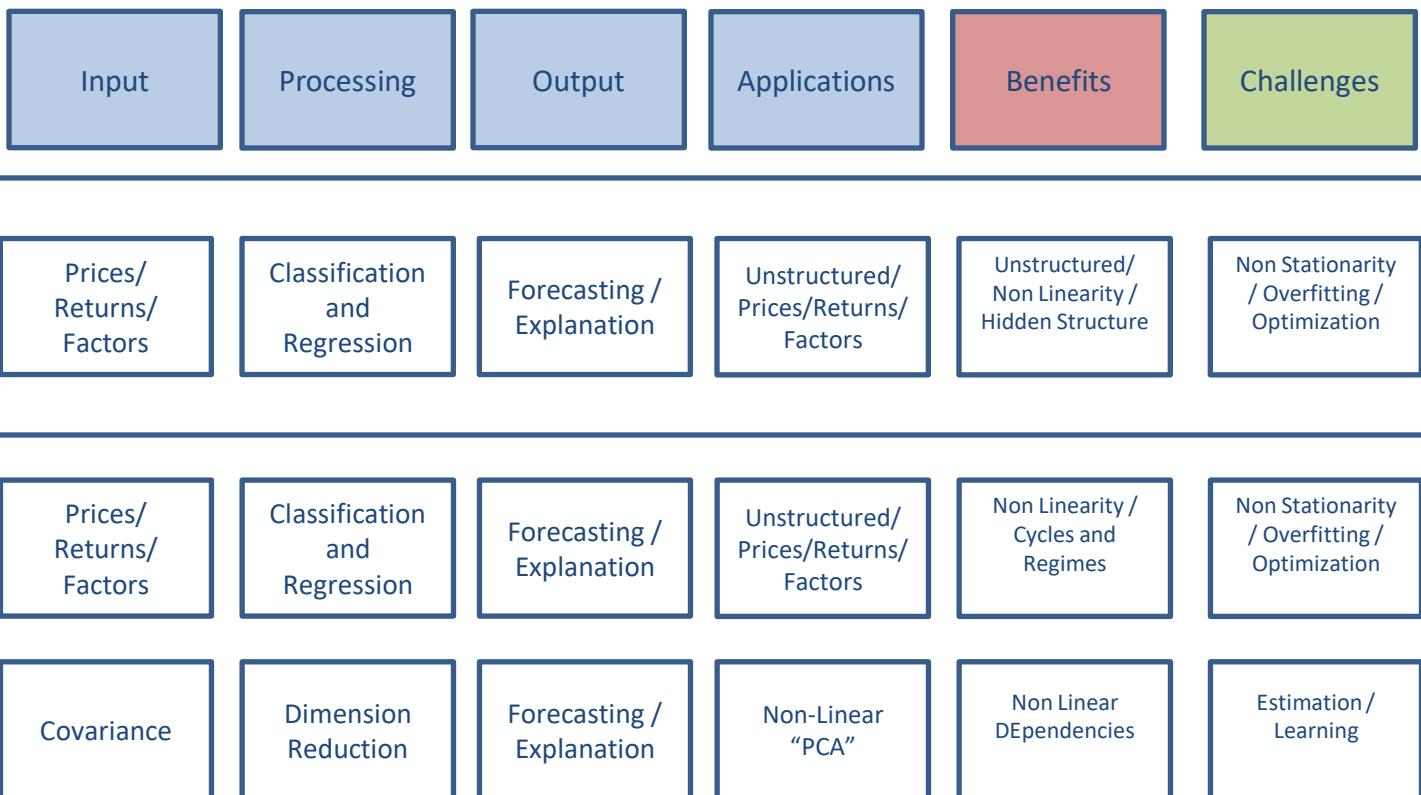
Deep Learning



Deep Learning Flow



Deep Learning in Finance Instructions of Use

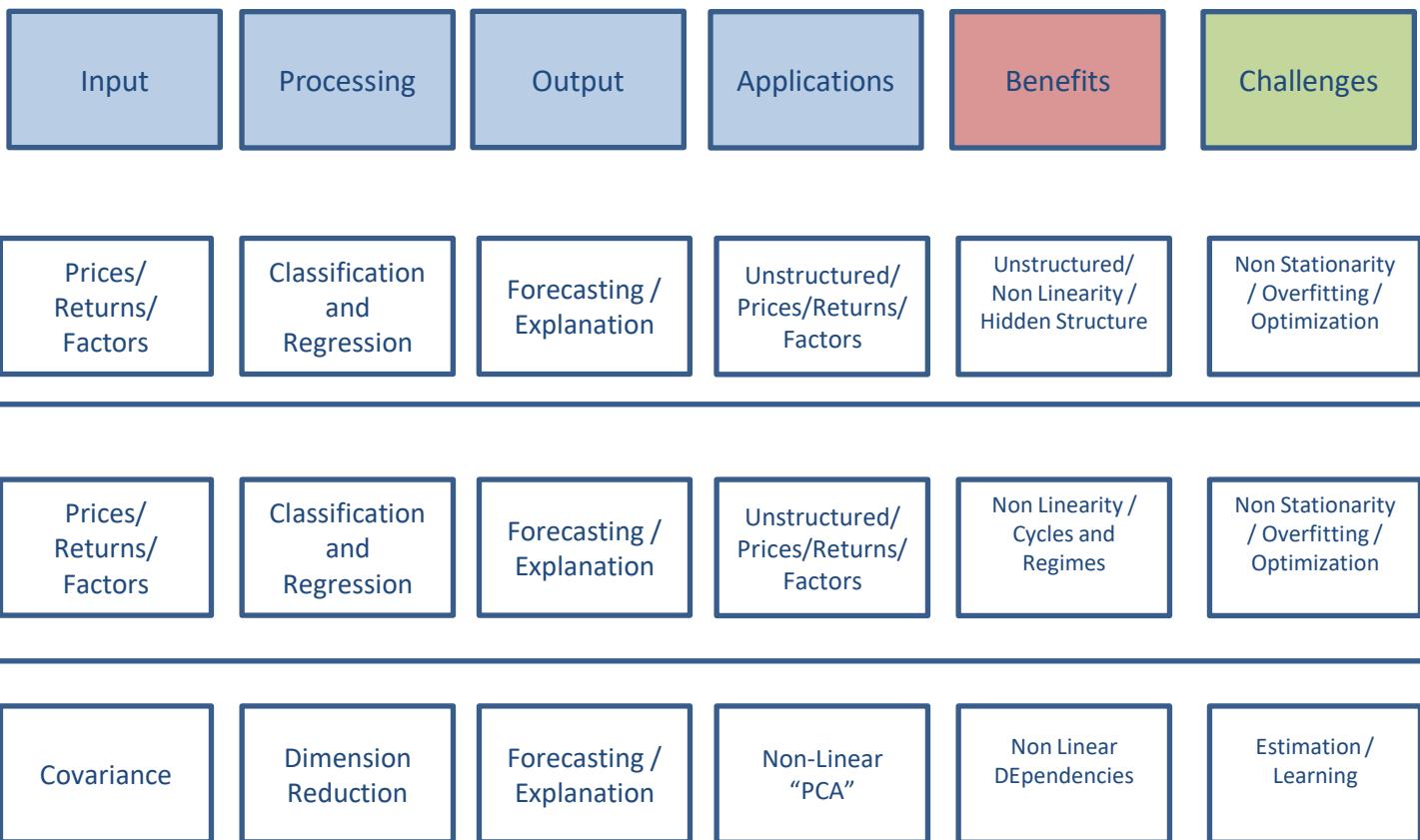


Machine Learning in Finance

Deep Learning

6.5.2 LSTM's for stock prediction

Deep Learning in Finance Instructions of Use



Long Short Term Memory Networks

Deep Learning in Finance: Prediction of Stock Returns with Long Short Term Memory Networks

Miquel Noguer Alonso ^{*1}, Gilberto Batres-Estrada ^{†2}, and Aymeric Moulin ^{‡3}

¹*Department of Industrial Engineering & Operations Research, Columbia University, New York, USA.*

Department of Economics, Finance and Accounting, ESADE, Barcelona, Spain. Institut d'Estudis Financers, IEF, Barcelona, Spain. ²*Webstep, Bergen, Norway.* ³*Department of Industrial Engineering & Operations Research, Columbia University, New York, USA.*

ABSTRACT

In this paper we investigate the profitability of a quantitative trading strategy based on Deep Learning methods. Specifically we focus on a variant of the Recurrent Neural Network (RNN), the Long Short Term Memory Network (LSTM) and show its predictive power on stock price data. We use LSTM networks for selecting stocks using historical price. The reason why RNNs are good for regression or classification of time series or data where time ordering matters is that RNNs capture the variation through time, thanks to its internal state dynamics. We made two studies, the first focuses on predicting stock returns using one stock at a time. The hit-ratio in this experiment lies in the range 0.47 and 0.60 for the worst respectively best performing stock on unseen “live” data. The second experiment looks at the whole universe of stocks simultaneously. In this experiment our model achieves a hit-ratio between 0.50 and 0.71 on unseen “live” data. From this experiment two portfolios were constructed, a long portfolio and a long-short portfolio with a Sharpe ratio of 8 respectively 10 for each of the portfolios. Our stock universe in both studies is composed of 50 stocks from the S&P 500.

Keywords: Deep Learning, Neural Networks, Recurrent Neural Networks, LSTM, Back-propagation through time, AI, finance, stock prediction, time series.

Long Short Term Memory Networks

The Long Short-Term Memory Network (LSTM) is a set of subnets with recurrent connections, known as memory blocks. Each block contains one or more self-connected memory cells and three multiplicative units known as the input, output and forget gates which respectively support read, write, and reset operations for the cells [14]. The gating units control the gradient-flow through the memory cell and when closing them allows the gradient pass without alteration for an indefinite amount of time, making the LSTM suitable for learning long time dependencies. Thus overcoming the vanishing gradient problem that RNNs suffer from. We describe in more detail the inner workings of an LSTM cell. The memory cell is composed of an input node, an input gate, an internal state, a forget gate and an output gate. First let bold letters denote vectors, each defined at time step t , then the components in a memory cell are as follows:

- The input node takes the activation from both the input layer, \mathbf{x}_t , and the hidden state \mathbf{h}_{t-1} at time $t - 1$. The input is then fed to an activation function, either a tanh or sigmoid.
- The input gate uses a sigmoidal unit that get its input from the current data \mathbf{x}_t and the hidden units at time step $t - 1$. The input gate multiplies the value of the input node, and because it is a sigmoid unit with range between zero and one, it can control the flow of the signal it multiplies.
- The internal state has a self-recurrent connection with unit weight also called the constant error carousel in [18], and is given by $\mathbf{s}_t = \mathbf{g}_t \odot \mathbf{i}_t + \mathbf{f}_t \odot \mathbf{s}_{t-1}$. The Hadamard product \odot , denotes element-wise product, and \mathbf{f}_t is the forget gate, see below.
- The forget gate, \mathbf{f}_t , was not part of the original model for the LSTM but was introduced by [11]. The forget gate multiplies the internal state at time step $t - 1$ and can in that way get

Long Short Term Memory Networks

rid of all the contents in the past as demonstrated by the equation in the list item above.

- The resulting output from a memory cell is produced by multiplying the value of the internal state s_c by the output gate o_c . Often the internal state is run through a tanh activation function.

The equations for the LSTM network can be summarized as follows. As before let \mathbf{g} stand for the input to the memory cell, \mathbf{i} for the input gate, \mathbf{f} for the forget gate, \mathbf{o} for the output gate and \mathbf{s} for the cell state, then we have, [21]

$$\mathbf{g}_t = \phi(W^{gX}\mathbf{x}_t + W^{gh}\mathbf{h}_{t-1} + \mathbf{b}_g) \quad (13)$$

$$\mathbf{i}_t = \sigma(W^{iX}\mathbf{x}_t + W^{ih}\mathbf{h}_{t-1} + \mathbf{b}_i) \quad (14)$$

$$\mathbf{f}_t = \sigma(W^{fX}\mathbf{x}_t + W^{fh}\mathbf{h}_{t-1} + \mathbf{b}_f) \quad (15)$$

$$\mathbf{o}_t = \sigma(W^{oX}\mathbf{x}_t + W^{oh}\mathbf{h}_{t-1} + \mathbf{b}_o) \quad (16)$$

$$\mathbf{s}_t = \mathbf{g}_t \odot \mathbf{i}_t + \mathbf{s}_{t-1} \odot \mathbf{f}_t \quad (17)$$

$$\mathbf{h}_t = \phi(\mathbf{s}_t) \odot \mathbf{o}_t. \quad (18)$$

where the Hadamard product \odot denotes element-wise multiplication. In the equations \mathbf{h}_t is the value of the hidden layer at time t , while \mathbf{h}_{t-1} is the output by each memory cell in the hidden layer at time $t - 1$. The weights $\{W^{gX}, W^{iX}, W^{fX}, W^{oX}\}$ are the connections between the inputs \mathbf{x}_t with the input node, the input gate, the forget gate and the output gate respectively. In the same manner $\{W^{gh}, W^{ih}, W^{fh}, W^{oh}\}$ represent the connections between the hidden layer with the input node, the input gate, the forget gate and the output gate respectively. The bias terms for each of the cell's components is given by $\{\mathbf{b}_g, \mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o\}$.

Long Short Term Memory Networks

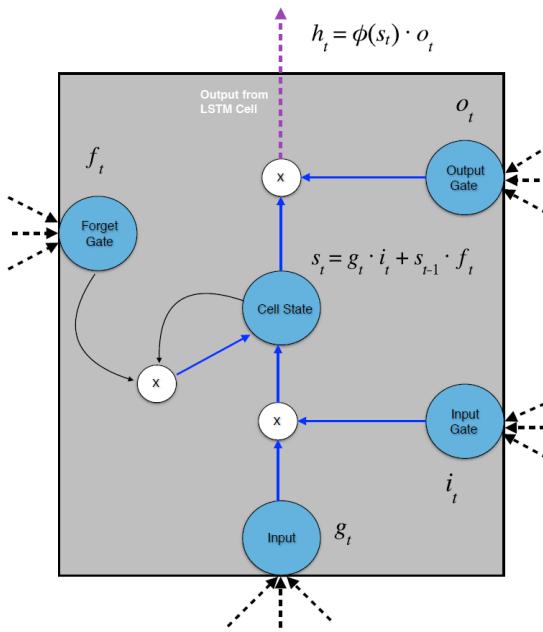


Figure 3. Memory cell or hidden unit in a Long-Short Term Memory (LSTM) recurrent neural network.

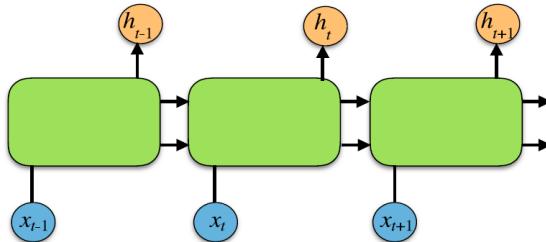


Figure 4. Long-Short Term Memory (LSTM) recurrent neural network unrolled in time.

Long Short Term Memory Networks - Results

Stock	Hidden Units	HR LSTM	HR SVM	HR NN
AAPL	150	0.53	0.52	0.52 (130)
MSFT	100	0.51	0.49	0.49 (150)
FB	100	0.58	0.58	0.56 (90)
AMZN	100	0.55	0.56	0.53 (90)
JNJ	100	0.52	0.47	0.50 (50)
BRK/B	150	0.51	0.51	0.51 (50)
JPM	100	0.52	0.51	0.50 (90)
XOM	100	0.52	0.52	0.49 (50)
GOOGL	100	0.54	0.53	0.53 (70)
GOOG	100	0.55	0.55	0.55 (50)
BAC	100	0.47	0.50	0.59 (50)
PG	100	0.50	0.50	0.50 (110)
T	150	0.52	0.48	0.50 (70)
WFC	150	0.51	0.47	0.50 (70)
GE	100	0.51	0.50	0.50 (110)
CVX	150	0.50	0.53	0.50 (70)
PFE	100	0.49	0.49	0.49 (50)
VZ	150	0.51	0.53	0.50 (50)
CMCSA	150	0.54	0.49	0.50 (110)
UNH	100	0.52	0.48	0.52 (130)
V	100	0.59	0.51	0.55 (70)
C	150	0.52	0.50	0.51 (50)
PM	100	0.56	0.56	0.52 (110)
HD	100	0.53	0.50	0.53 (70)
KO	150	0.51	0.48	0.50 (70)
MRK	200	0.54	0.49	0.50 (110)
PEP	100	0.55	0.52	0.51 (50)
INTC	150	0.53	0.45	0.51 (110)
CSCO	100	0.51	0.48	0.50 (90)
ORCL	150	0.52	0.48	0.50 (130)
DWDP	150	0.51	0.48	0.50 (90)
DIS	150	0.53	0.49	0.52 (130)
BA	100	0.54	0.53	0.51 (50)
AMGN	100	0.51	0.52	0.53 (90)
MCD	150	0.55	0.48	0.52 (130)
MA	100	0.57	0.57	0.55 (130)
IBM	100	0.49	0.49	0.50 (50)
MO	150	0.55	0.47	0.52 (50)
MMM	100	0.53	0.46	0.52 (90)
ABBV	100	0.60	0.38	0.41 (110)
WMT	100	0.52	0.50	0.51 (50)
MDT	150	0.52	0.49	0.50 (50)
GILD	100	0.50	0.52	0.51 (70)
CELG	100	0.51	0.52	0.50 (90)
HON	150	0.55	0.46	0.52 (130)
NVDA	100	0.56	0.55	0.54 (90)
AVGO	100	0.57	0.57	0.51 (130)
BMY	200	0.52	0.49	0.50 (50)
PCLN	200	0.54	0.54	0.53 (70)
ABT	150	0.50	0.47	0.50 (70)

Training period	HR %	Avg Ret % (L)	Avg Ret % (L/S)	Sharpe ratio (L)	Sharpe ratio (L/S)
2000-2003	49.7	-0.05	-0.12	-0.84	-1.60
2001-2004	48.1	0.05	-0.02	2.06	-0.73
2002-2005	52.5	0.11	0.10	6.05	3.21
2003-2006	55.9	0.10	0.16	5.01	5.85
2004-2007	54.0	0.14	0.12	9.07	5.11
2005-2008	61.7	0.26	0.45	7.00	9.14
2006-2009	59.7	0.44	1.06	3.10	6.22
2007-2010	53.8	0.12	0.12	5.25	2.70
2008-2011	56.5	0.20	0.26	6.12	6.81
2009-2012	62.8	0.40	0.68	6.31	9.18
2010-2013	55.4	0.09	0.14	3.57	3.73
2011-2014	58.1	0.16	0.21	5.59	6.22
2012-2015	56.0	0.15	0.21	5.61	5.84

Long Short Term Memory Networks - Conclusions

These ratios show a good behavior in-sample and out-of-sample. Sharpe ratios of our portfolio experiments are 8 for the long only portfolio and 10 for the long short version, an equally weighted portfolio would have provided a 2.7 Sharpe ratio using the model from 2014 to 2017. Results show consistency when using the same modeling approach in different market regimes. No trading costs have been considered.

We can conclude that LSTM networks are a promising modeling tool in financial time series especially in the multivariate LSTM networks with exogenous variables. These networks can enable financial engineers to model time dependencies, non-linearity, feature discovery with a very flexible model that might be able to offset the very challenging estimation and non-stationarity in finance and the potential of overfitting. These issues can never be underestimated in finance even more in models with a high number of parameters, non-linearity and difficulty to interpret like LSTM networks.

We think financial engineers should then incorporate deep learning not only to model unstructured but also to structured data. We have very interesting modeling times ahead of us.

Machine Learning in Finance

7. Reinforcement Learning

Reinforcement learning

An especially promising approach to Machine Learning is **reinforcement learning**.

The goal of reinforcement learning is to choose a course of actions that maximizes some reward. For instance, one may look for a set of trading rules that maximizes PnL after 100 trades.

Unlike supervised learning (which is typically a one-step process), the model doesn't know the correct action at each step, but learns over time which succession of steps led to the highest reward at the end of the process.

A fascinating illustration is a performance of the reinforcement learning algorithm in playing a simple Atari videogame (Google's DeepMind). After training the algorithm on this simple task, the machine can easily beat a human player. While most human players (and similarly traders) learn by maximizing rewards, humans tend to stop refining a strategy after a certain level of performance is reached. On the other hand, the machine keeps on refining, learning, and improving performance until it achieves perfection.

<https://hackernoon.com/the-self-learning-quant-d3329fcc9915>

Reinforcement learning

At the core of **reinforcement learning** are two challenges that the algorithm needs to solve: 1) Explore vs. Exploit dilemma – should the algorithm explore new alternative actions that may not be immediately optimal but may maximize the final reward (or stick to the established ones that maximize the immediate reward); 2) Credit assignment problem – given that we know the final reward only at the last step (e.g. end of game, final PnL), it is not straightforward to assess which step during the process was critical for the final success. Much of the reinforcement learning literature aims to answer the twin questions of the credit assignment problem and exploration-exploitation dilemma.

When used in combination with Deep Learning, reinforcement learning has yielded some of the most prominent successes in machine learning, such as self-driving cars. Within finance, reinforcement learning already found application in execution algorithms and higher-frequency systematic trading strategies.

Reinforcement learning has attributes of both supervised and unsupervised learning. In supervised learning, we have access to a training set, where the correct output “y” is known for each input. At the other end of the spectrum was unsupervised learning where we had no correct output “y” and we are learning the structure of data. In reinforcement learning we are given a series of inputs and we are expected to predict y at each step. However, instead of getting an instantaneous feedback at each step, we need to study different paths/sequences to understand which one gives the optimal final result.

REINFORCEMENT LEARNING Q-Learning

In the case where the Q-function is known, the optimal action is trivial to determine using

$$\pi(s) = \operatorname{argmax}_a Q(s, a)$$

The catch is that the Q-function is not known a priori. The classical solution for the problem is to iteratively approximate the Q-function using Bellman's equation, which claims that

$$Q(s_t, a_t) = r_t + \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

The Bellman equation simply says that the maximum reward for an action in the current state is equal to the sum of the immediate reward and the maximum reward possible from the succeeding state.

If the number of states and actions were small, the above equation would work fine. In many cases, the number of states and actions are large. This motivates the use of a neural network to learn the Q-function. A neural network can process a large number of states and extract the best features from them. Often, it may be more efficient in learning the function than a mere tabular approach that matches values of state and action pairs with possible value output. Putting all this together leads to the deep Q-learning algorithm.

REINFORCEMENT LEARNING Q-Learning

In the case of reinforcement learning, we retain the assumption of partial observability of the system, but now actively endeavor to control the outcome. In this section, we give a brief introduction to deep Q-learning, which is an approach to reinforcement learning using neural networks.

We do not discuss mathematical details corresponding to (Partially Observed) Markov Decision Processes.

Let the sequence of states, actions and rewards be denoted by $s_0, a_0, r_0, \dots, s_n, a_n, r_n$. From any step t in the process, one can define a discounted future reward as

$$R_t = r_t + \gamma r_{t+1} + \dots + \gamma^{n-t} r_n$$

In Deep Q-learning, we define a Q-function to denote the maximum future reward from performing a certain action now, i.e.

$$Q(s_t, a_t) = \max R_{t+1}$$

REINFORCEMENT LEARNING Q-Learning

Algorithm 1 Q-Learning

```
1: 1. Initialisation:  
    Load a simulation environment: price series, fill probability;  
    Initialise the value function  $V_0$  and set the parameters:  $\alpha, \epsilon$ ;  
2: 2. Optimisation:  
3: for episode = 1,2,3... do  
4:   for  $t = 1,2,3\dots T$  do  
5:     Observe current state  $s_t$ ;  
6:     Take an action  $a_t(Q_t, s_t, \epsilon)$ ;  
7:     Observe new state  $s_{t+1}$ ;  
8:     Receive reward  $r_t(s_t, a_t, s_{t+1})$ ;  
9:     Update value function using  $r_t$  and current estimate  $Q_t$ :  
        a) compute  $y_t = r_t + \max_a Q_t(s_{t+1}, a)$   
        b) update the function  $Q_t$  with target  $y_t$   
10:   end for  
11: end for
```

Partially Observable Markov Decision Process

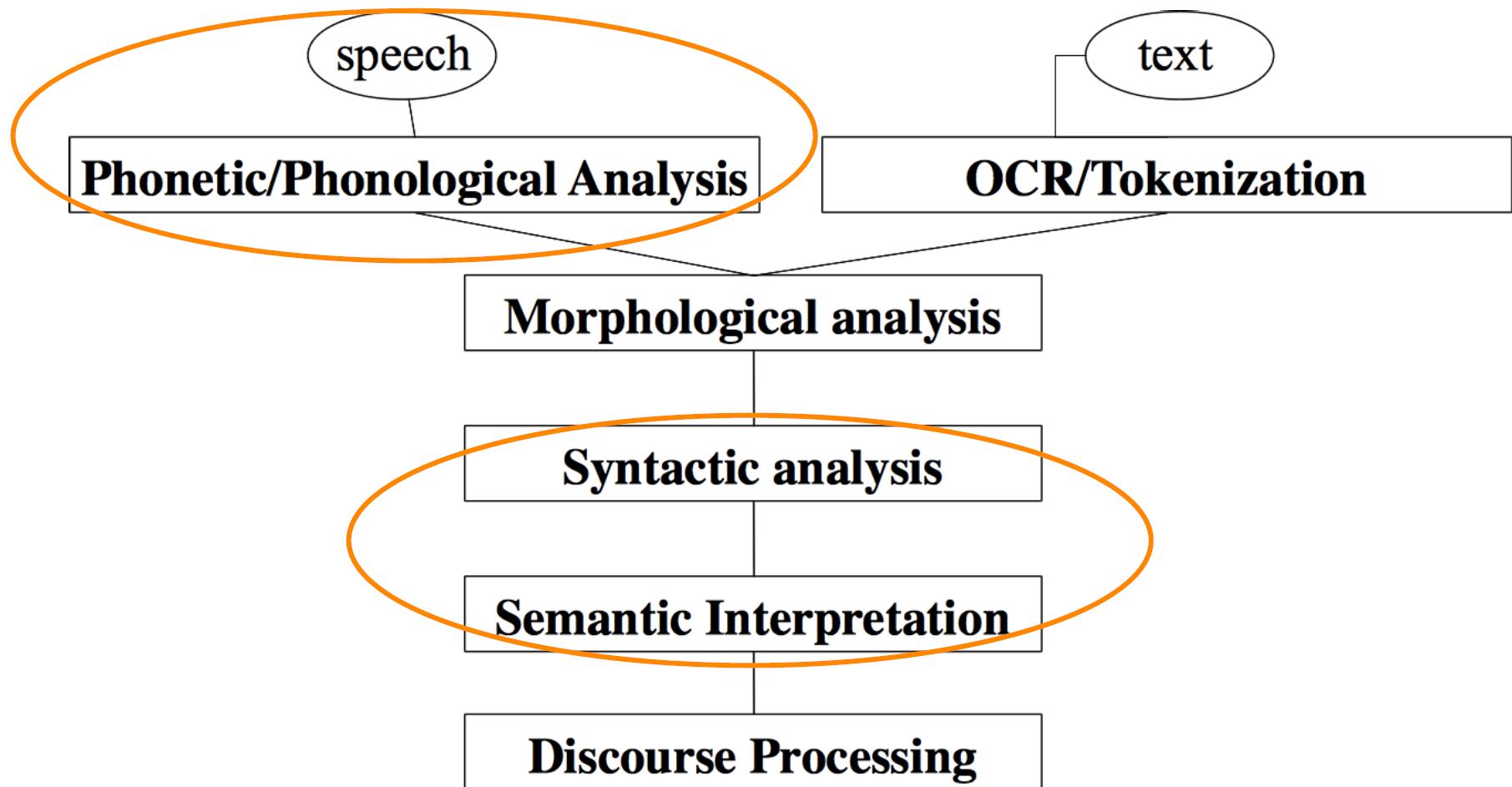
A POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \mathcal{Z}, \gamma \rangle$

- \mathcal{S} is a finite set of states
- \mathcal{A} is a finite set of actions
- \mathcal{O} is a finite set of observations
- \mathcal{P} is a state transition probability matrix,
$$\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$
- \mathcal{R} is a reward function, $\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
- \mathcal{Z} is an observation function,
$$\mathcal{Z}_{s'o}^a = \mathbb{P}[O_{t+1} = o \mid S_{t+1} = s', A_t = a]$$
- γ is a discount factor $\gamma \in [0, 1]$.

Machine Learning in Finance

7. Natural Language Processing

Natural Language Processing Levels



Natural Language Processing Applications

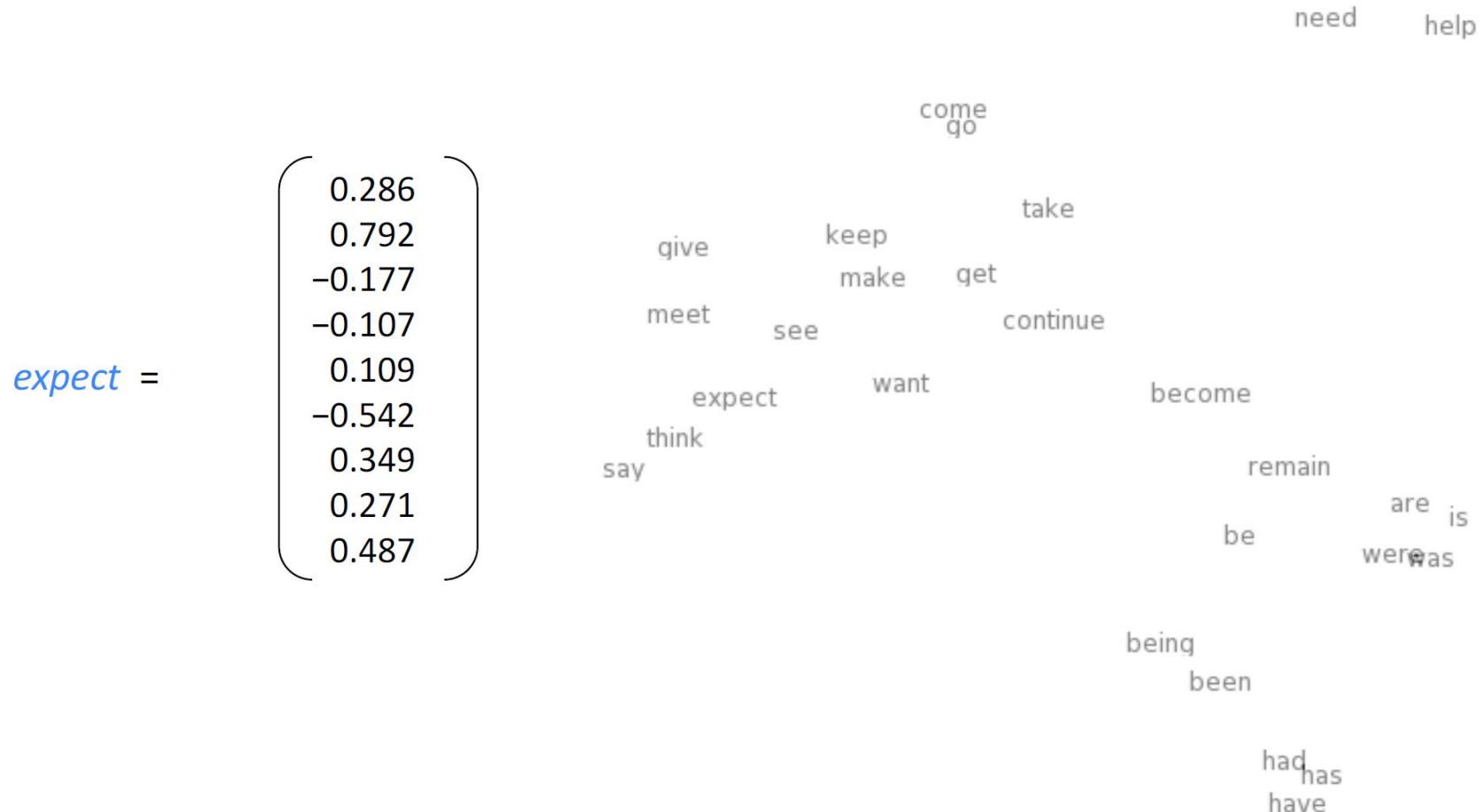
Applications range from simple to complex:

- Spell checking, keyword search, finding synonyms
- Extracting information from websites such as product price, dates, location, people or company names
- Classifying: reading level of school texts, positive/negative sentiment of longer documents
- Machine translation
- Spoken dialog systems
- Complex question answering

NLP in Industry

- Online advertisement matching
- Search
- Automated/assisted translation
- Sentiment analysis for marketing or finance/trading
- Speech recognition
- Chatbots / Dialog agents
- Automating customer support
- Controlling devices
- Ordering goods

Word meaning as a neural word vector – visualization



Word similarities

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus



litoria



leptodactylidae



rana



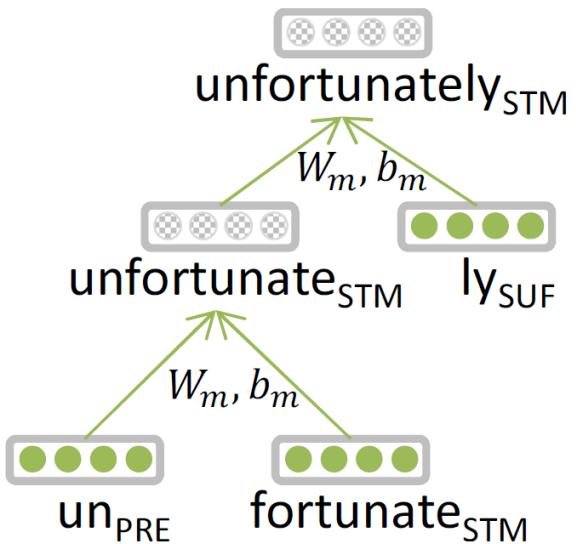
eleutherodactylus

Representations of NLP Levels: Morphology

- Traditional: Words are made of morphemes

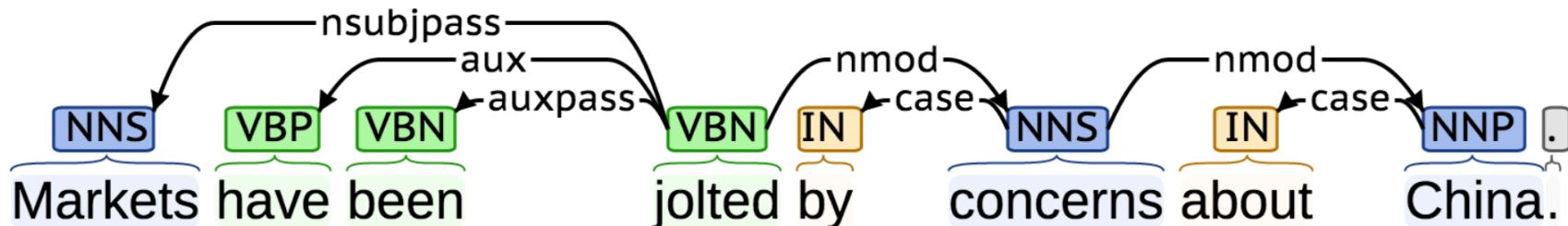
prefix	stem	suffix
un	interest	ed

- DL:
 - every morpheme is a vector
 - a neural network combines two vectors into one vector
 - Luong et al. 2013



NLP Tools: Parsing for sentence structure

Neural networks can accurately determine the structure of sentences, supporting interpretation



Softmax layer:

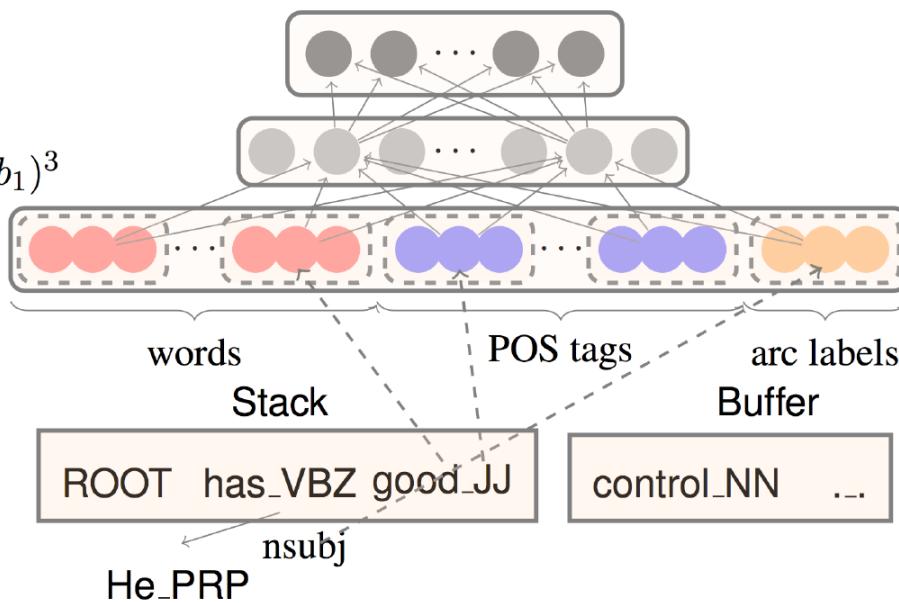
$$p = \text{softmax}(W_2 h)$$

Hidden layer:

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$

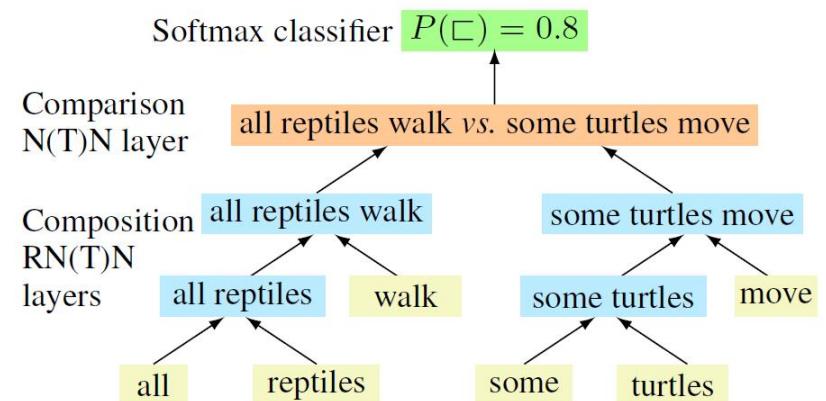
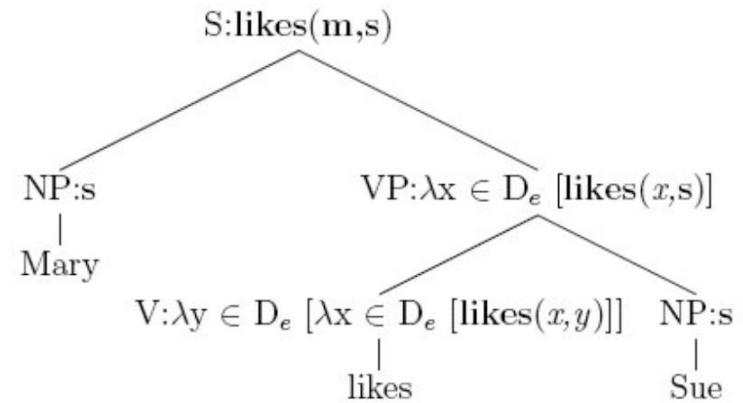
Input layer: $[x^w, x^t, x^l]$

Configuration



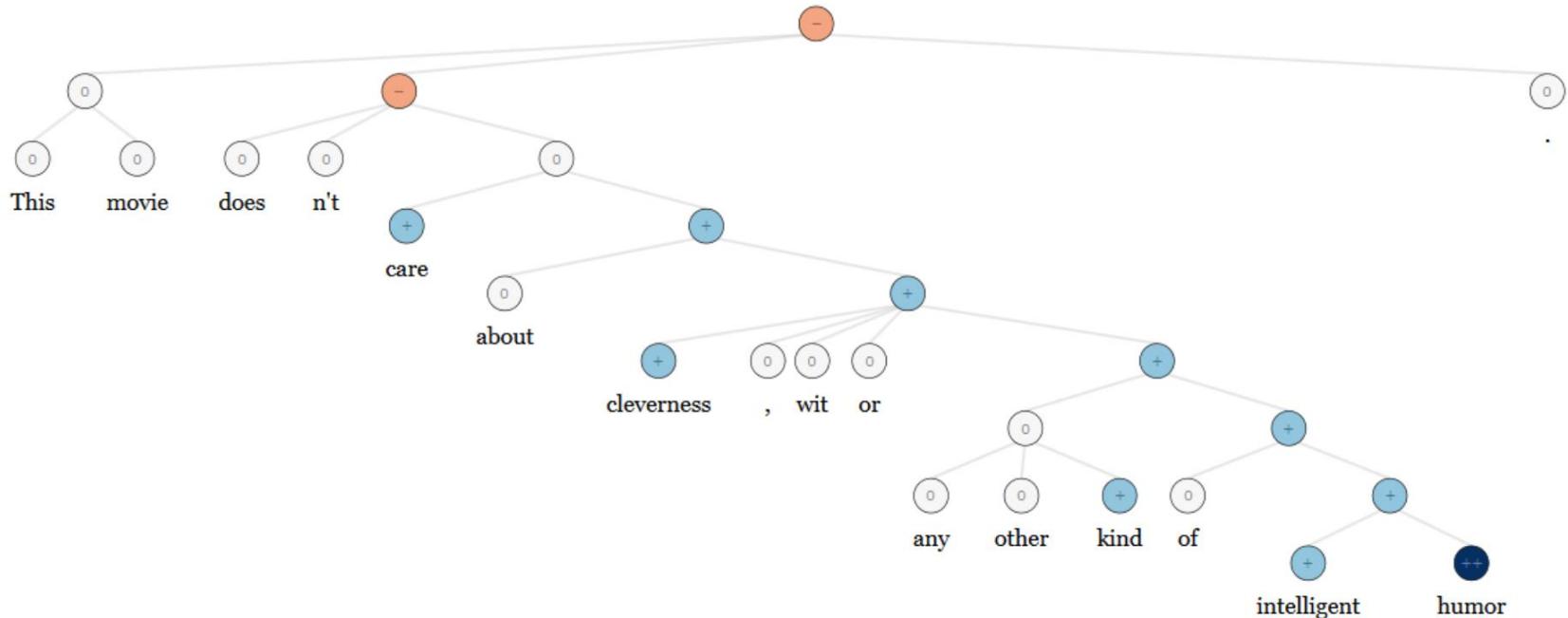
Representations of NLP Levels: Semantics

- Traditional: Lambda calculus
 - Carefully engineered functions
 - Take as inputs specific other functions
 - No notion of similarity or fuzziness of language
- DL:
 - Every word and every phrase and every logical expression is a vector
 - a neural network combines two vectors into one vector
 - Bowman et al. 2014



NLP Applications: Sentiment Analysis

- Traditional: Curated sentiment dictionaries combined with either bag-of-words representations (ignoring word order) or handdesigned negation features (ain't gonna capture everything)
- Same deep learning model that was used for morphology, syntax and logical semantics can be used! - RecursiveNN



Machine Learning in Finance

Natural Language Processing

Word2Vec

Word meaning is defined in terms of vectors

We will build a dense vector for each word type, chosen so that it is good at predicting other words appearing in its context

... those other words also being represented by vectors ... it all gets a bit recursive

$$\text{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Basic idea of learning neural network word embeddings

We define a model that aims to predict between a center word w_t and context words in terms of word vectors

$$p(\text{context} | w_t) = \dots$$

which has a loss function, e.g.,

$$J = 1 - p(w-t | w_t)$$

We look at many positions t in a big language corpus

We keep adjusting the vector representations of words to minimize this loss

Main idea of word2vec (Mikolov et al. 2013)

Predict between every word and its context words!

Two algorithms

1. Skip-grams (SG) - Predict context words given target (position independent)
2. Continuous Bag of Words (CBOW) - Predict target word from bag-of-words context

Two (moderately efficient) training methods

1. Hierarchical softmax
2. Negative sampling

Naïve softmax

Machine Learning in Finance

Natural Language Processing in Finance

Natural Language Processing Literature Survey

Earnings Call Transcripts

1. Tips and Tells From Managers: How Analysts And The Market Read Between The Lines Of Conference Calls – Druz, Wagner, Zeckhauser 2015
2. Playing Favorites: How Firms Prevent the Revelation of Bad News – Cohen, Lou, and Malloy 2014

10-K and Management Disclosure & Analysis

3. Lazy Prices – Cohen, Malloy, and Nguyen 2016
4. Word Power: A New Approach for Content Analysis – Jegadeesh and Wu 2013
5. Do Stock Market Investors Understand the Risk Sentiment of Corporate Annual Reports? – Li 2006
6. Management's Tone Change, Post Earnings Announcement Drift and Accruals – Feldman, Govindaraj, Livnat, Segal 2009

News/Press Release

7. News versus Sentiment: Predicting Stock Returns from News Stories – Heston and Sinha 2016
8. BlackRock Applied Finance Project: Using Natural Language Processing techniques for Stock Return Predictions – Chew, Puri, Sood, and Wearne 2017
9. News vs. Sentiment: Predicting Stock Returns from News Stories – Heston and Sinha 2017

Natural Language Processing

Definitions

Tokens: words or entities (e.g., punctuations, numbers, URLs, etc.) present in a text.

Tokenization: the process of converting a text (a list of strings) into tokens.

General Steps in NLP

There are generally three major steps in NLP :

- Text preprocessing, includes noise removal, lexicon normalization and objective standardization.
- Text to features includes syntactical parsing, entity parsing, statistical features and word embedding.
- Testing & refinement includes the designing, the calibrating and the refining of a model that is the engine which helps users to extract useful information from a content set or perform an automated task.

Natural Language Processing

Definitions

Tokens: words or entities (e.g., punctuations, numbers, URLs, etc.) present in a text.

Tokenization: the process of converting a text (a list of strings) into tokens.

General Steps in NLP

There are generally three major steps in NLP :

- Text preprocessing, includes noise removal, lexicon normalization and objective standardization.
- Text to features includes syntactical parsing, entity parsing, statistical features and word embedding.
- Testing & refinement includes the designing, the calibrating and the refining of a model that is the engine which helps users to extract useful information from a content set or perform an automated task.

1 - Text PreProcessing

- **Noise removal** is cleaning of textual data by stripping away anything that is not relevant to the task at hand including but not limited to the removal of acronyms, punctuations and numbers. Usually, URLs, hashtags, acronyms and stopwords (e.g., words like the) are removed. In fact, anything that isn't relevant to one's analysis could be considered as noise. One solution that we used is to come up with a defined set of words and entities such that everything else was removed as noise.
- **Lexicon normalization** is the method of standardizing multiple representations that are exhibited by a single word (e.g., different inflections of a word: e.g., play, plays, played, etc.). Two commonly used solutions are stemming and lemmatization. The goal of both is to reduce inflectional forms and derivationally related forms of a word to its root. Stemming is a cruder process of chopping off the suffix of different inflections of a word whereas lemmatization removes inflectional endings to return a word to its root morphologically.
- **Object standardization** is the process of converting shorthand forms or variant spellings to the formal spelling (e.g., luv to love). One solution is to use a comprehensive dictionary and all misspellings and abbreviations are considered noise and removed.

2 - Text to Features

- **Syntactical Parsing** is commonly broken into two types of analysis: dependency grammar and part of speech tagging (PoS). They are used to model out the structure of sentences. Dependency is in the context of grammar. Each sentence is broken down into a triplet relation (i.e., relation, governor and dependent). One can think of the triplet relation in grammar terms as the subject, the verb and the direct (or indirect) object in a sentence. Typically a tree is used to represent the triplet graphically. The main idea of syntactical parsing is to take into account sentence structures before processing a text.
- Another common tool to understand syntax is PoS. For instance, many English words can take on different parts-of-speech depending on context (e.g., book – book a flight or reading a book). By having PoS taggings, NLP has additional information to process and understand words from the PoS dimension.
- **Statistical features** are the numerical results of converting a text into quantifiable characteristics. Common examples are a count of words, sentences or syllables from a text. From there, one can derive, for instance, the sentiment (i.e., the tone) of a text by counting the proportion of negative words in that text or the readability of a text (e.g., Gunning Fog Index) by calculating the average word count per sentence and the proportion of polysyllabic words in that text.

2 - Text to Features - Word Embedding

- **Syntactical Parsing** is commonly broken into two types of analysis: dependency grammar and part of speech tagging (PoS). They are used to model out the structure of sentences. Dependency is in the context of grammar. Each sentence is broken down into a triplet relation (i.e., relation, governor and dependent). One can think of the triplet relation in grammar terms as the subject, the verb and the direct (or indirect) object in a sentence. Typically a tree is used to represent the triplet graphically. The main idea of syntactical parsing is to take into account sentence structures before processing a text.
- Another common tool to understand syntax is PoS. For instance, many English words can take on different parts-of-speech depending on context (e.g., book – book a flight or reading a book). By having PoS taggings, NLP has additional information to process and understand words from the PoS dimension.
- **Statistical features** are the numerical results of converting a text into quantifiable characteristics. Common examples are a count of words, sentences or syllables from a text. From there, one can derive, for instance, the sentiment (i.e., the tone) of a text by counting the proportion of negative words in that text or the readability of a text (e.g., Gunning Fog Index) by calculating the average word count per sentence and the proportion of polysyllabic words in that text.

2 - Text to Features - Word Embedding

	Words			
Dimensions	King	Queen	Woman	Princess
Royalty	0.990	0.990	0.020	0.980
Masculinity	0.990	0.050	0.010	0.020
Femininity	0.050	0.930	0.999	0.960
Age	0.700	0.600	0.500	0.100

Note: values are fictitious for illustration purposes and the dimensions are unknown in reality

As the number of dimensions increases, word embedding would effectively map out all the different contexts that a word can be used with numerical values indicating the strength of that particular association. For instance by applying vector math to our example, word embedding has learned the concept of 'gender' without us explicitly teaching it that concept (e.g., when you do the vector math: king - masculinity + femininity = queen). Word embedding is one of the fastest growing, cutting-edge techniques where it has been the main tool for a number of major breakthroughs in the NLP space.

3 - Testing, Refinement & Assessment of Efficacy

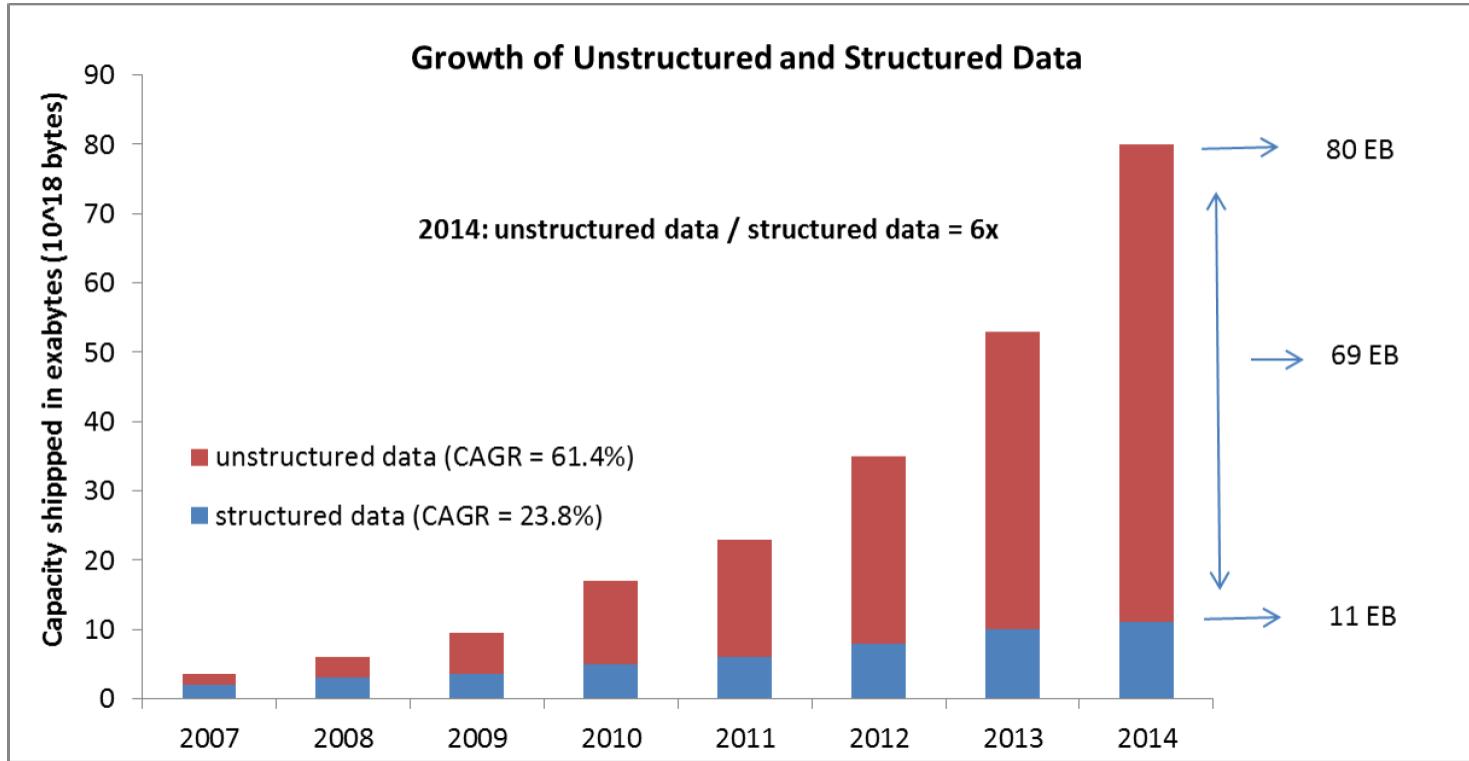
Testing & refinement is the final step in the process. Testing refers to whether NLP has performed a task well using a threshold (e.g., 60% of actual incoming spam emails are categorized as spam emails) that is pre-defined by the user. Refinement is the repeated calibration of the NLP algorithm until it meets the modeler's efficacy threshold. In order to mitigate the risk of data mining, one usually takes the entire input data set and divides it into at least three parts. For instance, fifty percent of the data (perhaps randomly chosen) are designated as the training set and the other fifty percent is set aside as the evaluation set. The training set is then further divided into two equal halves where one of which (basically 25% of the overall data) is labeled as the development set and the other (the other 25%) is designated as the development test set.

$$\text{Total data set (100 \%)} = \text{Development Set (50\%)} + \text{Evaluation Set (50\%)}$$

$$\text{Development Set (50\%)} = \text{Training Set (25\%)} + \text{Development Testing Set (25\%)}$$

The idea is that a model is calibrated on the development training set and the testing of that model's efficacy is done on the development test set. Once the modeler is happy with all the refinements then she takes her model (where she no longer makes any changes at least for the time being) to the evaluation set for the final efficacy testing. For those who are familiar with empirical research, the concept is the same as setting aside in- and out-of-sample data sets. Now using our earlier spam email illustration, the modeler validates her algorithm by looking at the emails that are sorted to the spam email folder. Her algorithm is deemed successful if its success rate in the evaluation set is equal to or greater than her pre-defined threshold for success of 60%.

Historical Growth of Unstructured & Structured Data



Machine Learning in Finance

Natural Language Processing

Applications in Finance

Natural Language Processing – Use Cases

1. An analysis of the historical relationship between a stock's sentiment changes vs. its forward returns
2. A heat-map of industry-level sentiment trends
3. Language complexity of earnings calls and sell-side analyst selectivity went hand-in-hand in Q2 2017, when managers wanted to soften bad news.

Details on Loughran and McDonald (2011) Financial Dictionary

There are many ways to define sentiment. We use a bag-of-words approach where the sentiment word lists are from the Loughran and McDonald (2011) financial dictionary. Their dictionary has become the de facto financial dictionary for NLP analysis due to its accessibility, its comprehensiveness, its financial-specific context, its lack of dependency on the transitory nature of its words and, lastly and perhaps most importantly, its unambiguous and singularly connotated words. Details below.

- **Accessibility** – their word lists are readily accessible because they are freely posted online.
- **Comprehensiveness** – the dictionary is comprehensive such that it is difficult for managers to game the system (i.e., circumvent certain words that have empirically been shown to lead to future stock underperformance) because they start with every conceivable English word with all inflections of a word, totaling 80,000+ distinct words in the master word list.
- **Financial-specific context** - they filter their initial master word list down to their sentiment word lists by examining 10-K filings between 1994 and 2008 inclusively.
- **Permanence of words** - their master and sentiment word lists are less transitory because they start with the most comprehensive list of English words possible and, more importantly, the master word list doesn't rely on transitory terms such as iphone.
- **Unambiguous and Singularly Connnotated Words** - they arrive at their sentiment word lists containing unambiguous and singularly connnotated words by looking at the most frequently occurring words in the 10-Ks from the master word list. From there, they went word-by-word and assessed each of the word's meaning in a business context. At the end of their process, the words that ended up in their word lists are less ambiguous in their meaning with singular connotation.

Details on Loughran and McDonald (2011) Financial Dictionary

Their three most important lists of words for our use cases are the master word list, positive and negative sentiment word lists with distinct word counts of 80,000+, 350+ and 2300+, respectively. Examples of positive words are able, abundance, acclaimed, accomplish and so forth. Examples of negative words are abandon, abdicate, aberrant, abetting and so forth.

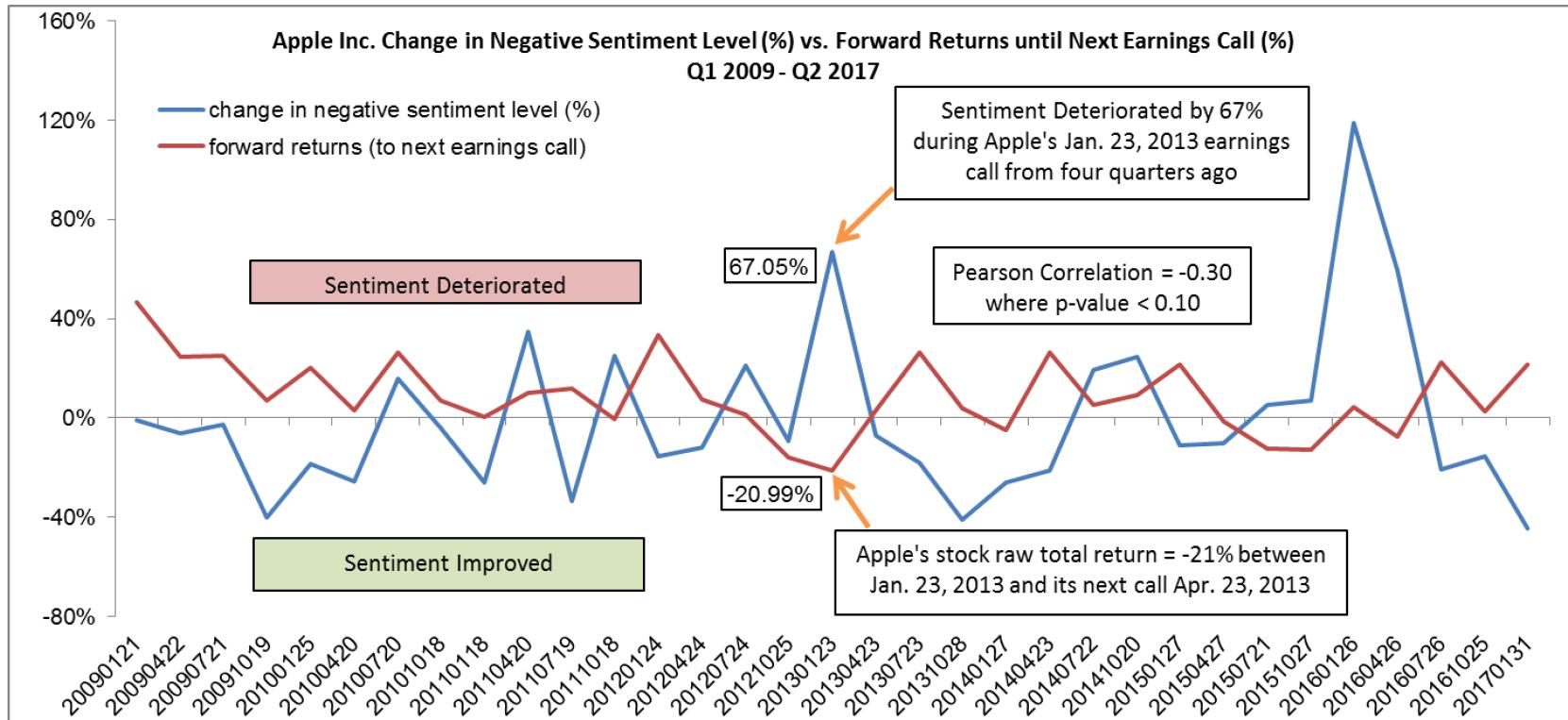
Case 1 - Sentiment Change vs. Forward Returns

The first use case is to capture the historical relationship of Apple Inc.'s sentiment level changes from its earnings calls and its forward returns until the next call (Exhibit next slide). The changes in sentiment are defined as quarter-over-quarter (QoQ) changes from four quarters ago (to account for seasonality).

The sentiment of each of Apple's earnings calls is defined by the proportion of negative words in its earnings call transcript where the classification of both the negative and the master word list is based on the Loughran and McDonald (2011) financial dictionary.

Because sentiment in this use case is measured with negative words, positive (negative) changes reflect sentiment deterioration (improvement). Apple's forward returns until its future calls have been shifted back a quarter such that its sentiment changes and its forward returns are aligned vertically in the Exhibit. One promising observation is that the Pearson correlation is about -0.30 since Q1 2009, which suggests that Apple's forward returns historically go down when its sentiment deteriorates.

Case 1 - Sentiment Change vs. Forward Returns



Note: Sentiment is defined as the proportion of negative words in an earnings call using [Lougrahan and McDonald](#)

Case 2 - S&P 500 Trends in Sentiment Change

The second use case provides a heat map showing sentiment trends for S&P 500 GICS industry groups. Exhibit next slide shows quarterly sentiment changes for 24 GICS industry groups by calendar quarter between Q3 2016 and Q2 2017 inclusively.

Sentiment is defined as the proportion of negative words in an earnings call using Lougrahan and McDonald (2011). Sentiment changes are measured quarter-over-quarter from four quarters ago where the values are multiplied by -1 to make results easier to interpret. Industry group level values are rolled up equal-weighted from the stock-level. Source: Data as of 08/08/2017.

The industry groups are sorted by their sentiment changes from Q2 2017 in descending order. Similar to the Apple's example from above, all the sentiment values are QoQ changes from four quarters ago where each industry group's sentiment is aggregated, on an equal-weighted basis, from the stock-level where the sentiment is measured using the proportion of negative words in a stock's earnings call. In order to make the interpretation more intuitive, we multiplied the values by negative one so green (red) values denote improvement (deterioration) in sentiment for the industry groups and the different color shades reflect the magnitude of sentiment changes.

One could easily visualize sentiment trends for an industry group and spot potential inflection points and accelerations. For example, the sentiment improved substantially for banks between calendar quarter Q4 2016 and Q1 2017. Investors would notice this as an inflection point to the upside and could potentially use the insight as an additional piece of information in their investment decision making process.

Case 2 - S&P 500 Trends in Sentiment Change

GICS Industry Groups	Calendar Quarters			
	Q3 2016	Q4 2016	Q1 2017	Q2 2017
Consumer Services	-4.1%	-4.0%	16.9%	20.1%
Software & Services	-0.2%	4.1%	-3.9%	19.2%
Transportation	-3.6%	-2.8%	14.6%	19.1%
Diversified Financials	5.4%	13.8%	16.5%	18.0%
Technology Hardware & Equipment	4.8%	7.1%	18.1%	17.5%
Banks	-3.7%	-0.4%	18.1%	16.8%
Capital Goods	-4.9%	9.1%	16.9%	15.5%
Commercial & Professional Services	7.6%	-4.4%	-6.2%	14.6%
Utilities	-4.6%	7.2%	7.0%	14.1%
Insurance	0.0%	14.4%	10.2%	11.9%
Energy	0.1%	13.6%	25.8%	10.3%
Real Estate	-11.5%	-2.4%	0.5%	5.5%
Media	-11.6%	-12.6%	5.0%	4.2%
Materials	5.7%	-1.8%	12.2%	1.4%
Semiconductors & Semiconductor Equipment	6.3%	-3.7%	14.5%	0.5%
Retailing	-18.4%	-0.3%	2.4%	-1.6%
Consumer Durables & Apparel	10.7%	0.2%	-4.7%	-2.6%
Pharmaceuticals, Biotechnology & Life Sciences	-5.7%	-3.6%	-1.3%	-2.9%
Household & Personal Products	-3.3%	-2.9%	-7.8%	-3.9%
Food & Staples Retailing	17.7%	-0.8%	-16.5%	-4.0%
Food, Beverage & Tobacco	2.2%	0.4%	-3.6%	-5.0%
Health Care Equipment & Services	-8.2%	-2.7%	-1.0%	-7.5%
Autos & Components	-14.6%	-4.3%	34.2%	-8.5%
Telecommunication Services	9.1%	-13.6%	-36.9%	-29.4%



Note: Sentiment is defined as the proportion of negative words in an earnings call using [Lougrahan and McDonald \(2011\)](#). Sentiment changes are measured quarter-over-quarter from four quarters ago where the values are multiplied by -1 to make results easier to interpret. Industry group level values are rolled up equal-weighted from the stock-level. Source: S&P Global Market Intelligence Quantamental Research. Data as of 08/08/2017.

Case 3 - Language Complexity of Earnings Calls and the Sell-Side Analyst Selectivity Ratio

The language complexity of earnings calls and the selectivity of sell-side analysts who are picked to ask questions during earnings calls in the form of a ratio.\

Language complexity

The language complexity of earnings calls is measured using the **(Gunning) fog index** (see the y-axis of next table), which measures the number of years of formal education that one needs to understand the diction used in the analyzed text (e.g., 16 is equivalent to someone who has completed an undergraduate degree), in our case an earnings call transcript. The fog index has two inputs: the average number of words per sentence and the proportion of polysyllabic words (threshold is 3 syllables or higher). A higher number of words per sentence and/or a higher proportion of polysyllabic words in a text would result in an increase in the fog index.

Why might a **fog index** be a good proxy to assess whether managers are trying to soften bad news? First, we surmise that when news is bad (e.g., missed earnings) managers need to disclose it due to their legal or fiduciary obligations. Hoping to mitigate the effect of the news on their stock prices, they may provide long-winded ‘explanations’ of the news, whereas in the quarters with good news answers to sell-side analysts tend to be shorter and more direct. Secondly, managers may give more long-winded answers to drain the time remaining in the Q&A section of an earnings call where the historical average duration of earnings calls is about an hour.

Case 3 - Language Complexity of Earnings Calls and the Sell-Side Analyst Selectivity Ratio

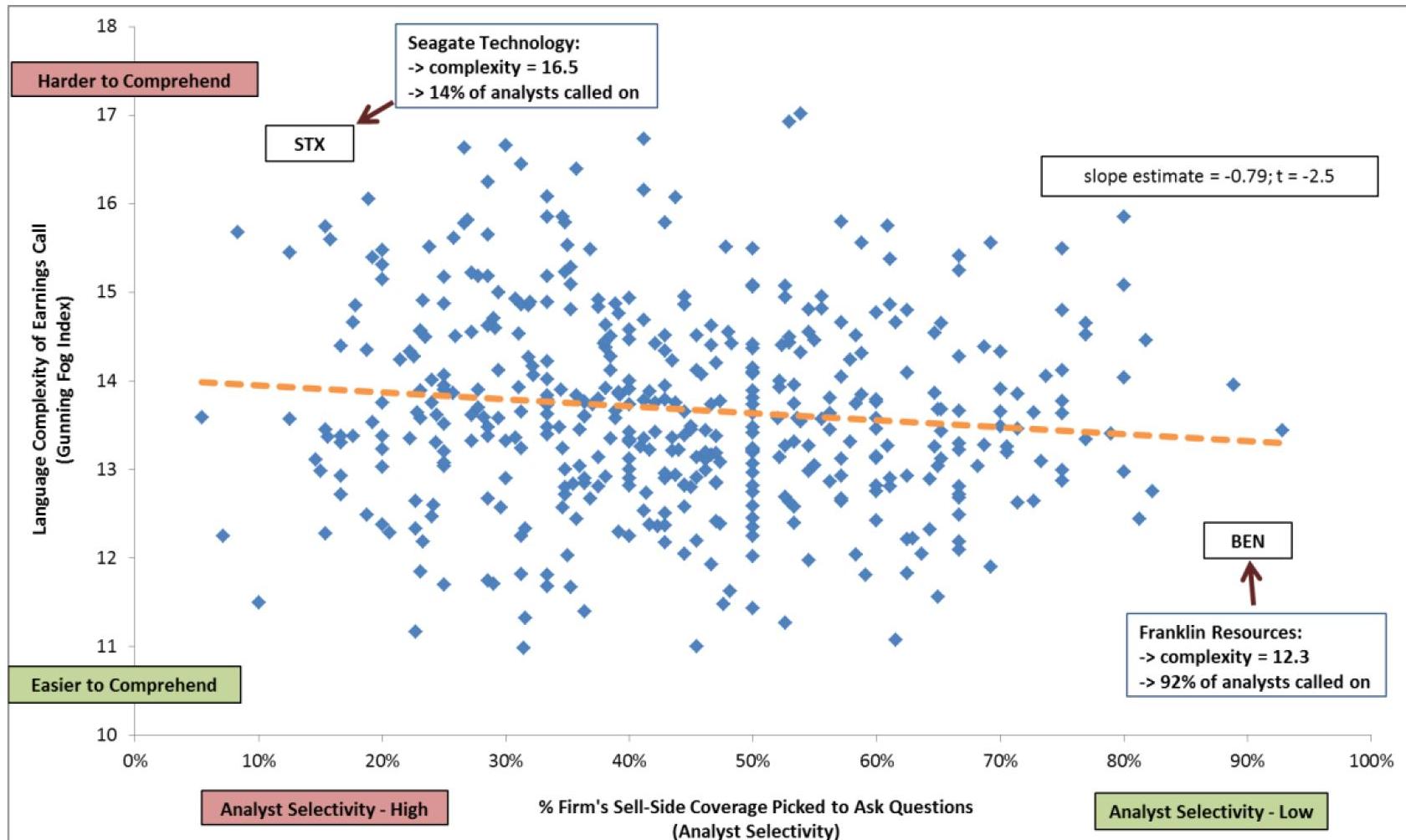
The **sell-side analyst selectivity ratio** is defined as the percent of the active sell-side coverage that are allowed to ask questions during an earnings call. For instance, Apple Inc. has forty active sell-side coverages prior to one of its earnings calls and its management allowed 10 of the analysts to ask questions, which translates to an analyst selectivity ratio of 25% and in our narrative is interpreted as a high analyst selectivity ratio and viewed negatively.

Our hypothesis is that when a firm has good financial results, it wants everyone to know, especially sell-side analysts because they are great messengers to buy-side money managers and traders. Answers to sell-side questions tend to be shorter and more direct when firms are doing well, which in turn enables managers to allow more analyst questions. When financial results are less favorable, the percent of sell-side analysts who get to ask questions declines due to two possible reasons. One is that managers tend to have lengthier explanations as to why the negative results are perhaps transitory, which consumes additional time that would otherwise be allocated to taking additional analyst questions. Secondly, managers may take multiple questions from analysts who may have a positive view on their firms.

Case 3 - Language Complexity of Earnings Calls and the Sell-Side Analyst Selectivity Ratio

Naturally, there are other ways of measuring analyst selectivity ratio (or language complexity). For instance, one potential issue with this analyst selectivity ratio is that it may have a market-capitalization bias. If a firm has a great number of sell-side analysts covering it, it is generally harder for a firm to call on every analyst who has an active coverage (e.g., Apple Inc. called only 7 of the 43 analysts during its calendar quarter Q2 2017). Because generally larger market-caps have a greater number of sell-side coverage, larger (smaller) market-cap firms may always have a higher (lower) analyst selectivity ratio. In calendar Q2 2017 using a snapshot of market-caps on Mar. 31, 2017, the spearman correlation between each firm's analyst selectivity ratio and its market-cap is -0.14.¹⁰ The -0.14 correlation result suggests that there is a relationship between a firm's analyst selectivity ratio and its market-cap (i.e., firms with larger market-caps have lower analyst selectivity ratios), but it isn't very strong. Thus, we are using this flavor of analyst selectivity ratio for the use case without further consideration, mainly because we want to keep the narrative as simple as possible.

Case 3 - Language Complexity of Earnings Calls and the Sell-Side Analyst Selectivity Ratio



Machine Learning in Finance

Natural Language Processing Tools

Natural Language Processing Tools

1. Python
2. https://www3.nd.edu/~mcdonald/Word_Lists.html
3. Format earnings call transcripts in the following way. There are five columns in the input files where the columns going from left to right are: stock identifier, earning call date, hour and minute, sequence identifier of every earnings call and the content of every earnings call
4. Python packages

```
#####
import os # import the os module
import pandas as pd # using pandas to create data type 'DataFrame' use
import time # time module can be used now e.g., time.time()
import nltk

#####
fpath = 'C:/Users/fzhao/Desktop/NLP/rawData'
os.chdir(fpath) # chg working directory to fpath
```

Natural Language Processing Tools

5. Load in Loughran and McDonald (2011) word lists

```
fn = 'wordLists\\lmMasterList.txt' # input file
tmpWordListPandaObj = pd.read_csv(fn, sep='\t', header=None) # read in tab-delimited
tmpWordListPandaObj.columns = ['master'] # rename cols
master_list = list(tmpWordListPandaObj.master)
master_list = [itr for itr in master_list if not isinstance(itr, float)]
master_list = [itr.lower() for itr in master_list] # make all lowercase
```

6. Define the functions that you are going to use. The one function included below outputs the frequency count of words

```
def _out_word_freq1(list1, list2):
    wordFreqDictObj = {}
    for item in list2:
        if item in list1:
            wordFreqDictObj[item] = wordFreqDictObj.get(item, 0) + 1
    return wordFreqDictObj
```

Natural Language Processing Tools

7. Have a control structure to traverse through time periods and firms. We decided to traverse one quarter at a time across all firms for that quarter. We used the while loop control structure whereas one could also use the for loop control structure. Relative to other languages like Matlab, C++, etc., the for loop in Python is very robust and powerful, which also makes some for loop syntaxes unintuitive. If you decide to use the for loop instead, perhaps consider using the key word enumerate where you can actually observe and utilize the iterator in the for loop

```
start_time = time.time(); # record time when a piece of code is executed
yrItr = 0
while yrItr < len(yrVec):

    qtrItr = 0
    while qtrItr < len(qtrVec):

        #creating filename to read in
        fn = str('transcriptsAll\\sp9_q' + str(qtrVec[qtrItr]) + 

        #print(yrVec[yrItr], qtrVec[qtrItr])

        #check file exists
        if os.path.isfile(fn) and os.stat(fn).st_size != 0:

            tmpPandaObj = pd.read_csv(fn, sep='\t', header=None)      # read
            tmpPandaObj.columns = ['stockId','dt','hhmm','seq','ecalls']  # ren...
```

Natural Language Processing Tools

8. This step is about tokenizing and preprocessing of an earnings call. In the screenshot below, we use the non-native, but well-known, nltk NLP library and specifically the method word_tokenize to parse an earnings call into tokens where the function parses whenever it encounters a (user-defined) punctuation or white space. We also show other text preprocessing that we did: i) removal of punctuation ii) removal of empty spaces between words and iii) standardizing all text to lower case and so forth

```
#####
# nltk mthds
tokensListObj      = nltk.word_tokenize(dsRawSeriesObj) # tokenize all words
tokensListObj      = [itr.lower() for itr in tokensListObj] # make all lowercase

#
# punct
punct              = string.punctuation # native punctuation string
punct              = '!"#$%&\'()*+,.:/;<=>?@[\\]^_`{|}~'; # string.punctuation less '-'
tokensListObj      = ["".join([j for j in i if j not in punct]) for i in tokensListObj]
tokensListObj      = [itr for itr in tokensListObj if itr] # keep on concat if item in list isnt empty
```

Natural Language Processing Tools

9. This step is about tokenizing and preprocessing of an earnings call. In the screenshot below, we use the non-native, but well-known, nltk NLP library and specifically the method word_tokenize to parse an earnings call into tokens where the function parses whenever it encounters a (user-defined) punctuation or white space. We also show other text preprocessing that we did: i) removal of punctuation ii) removal of empty spaces between words and iii) standardizing all text to lower case and so forth

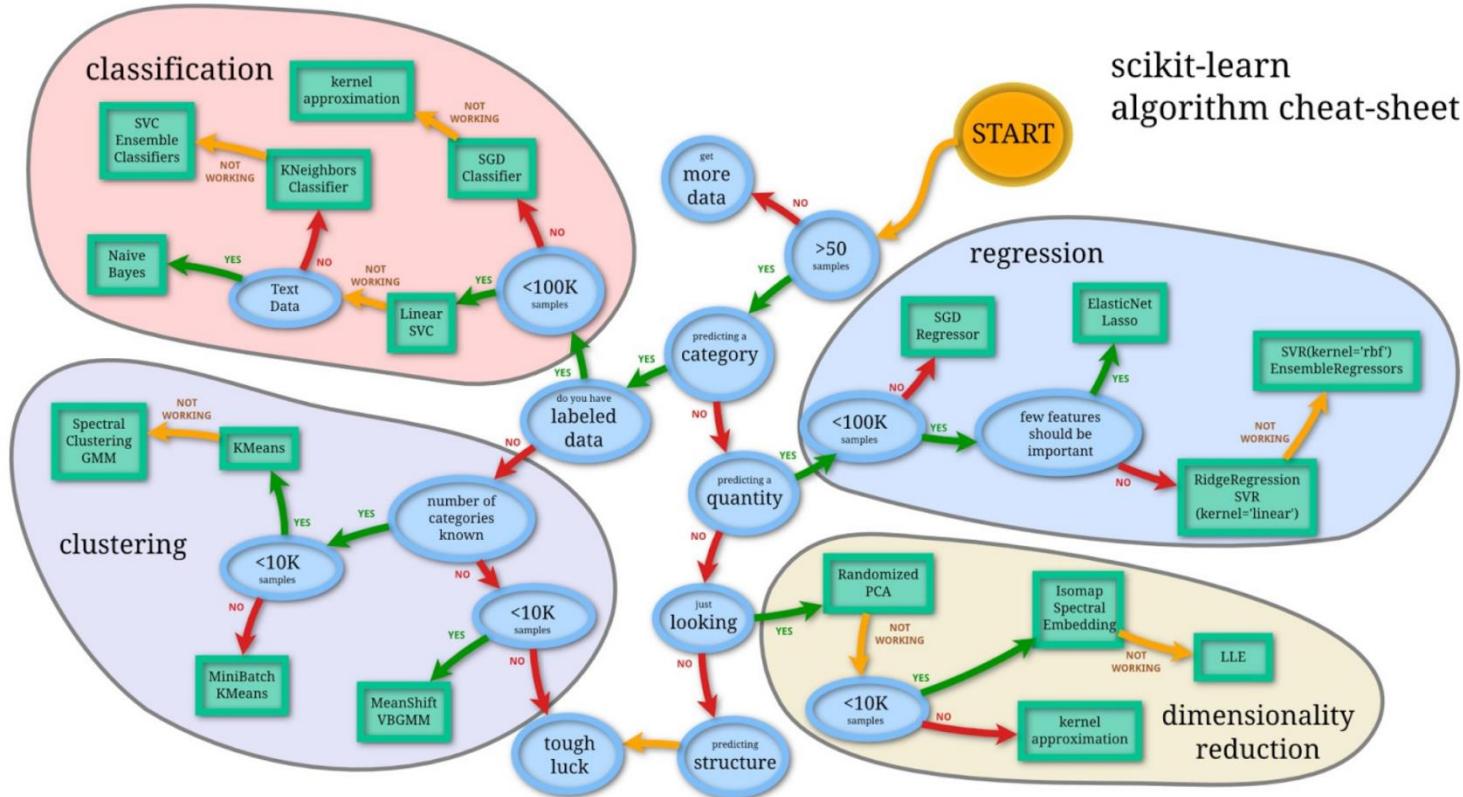
```
tmpWordsList          = list(set(tokensListObj) & set(master_list))
tokensNmasterList    = tmpWordsList
# tmpWordsList2
# set

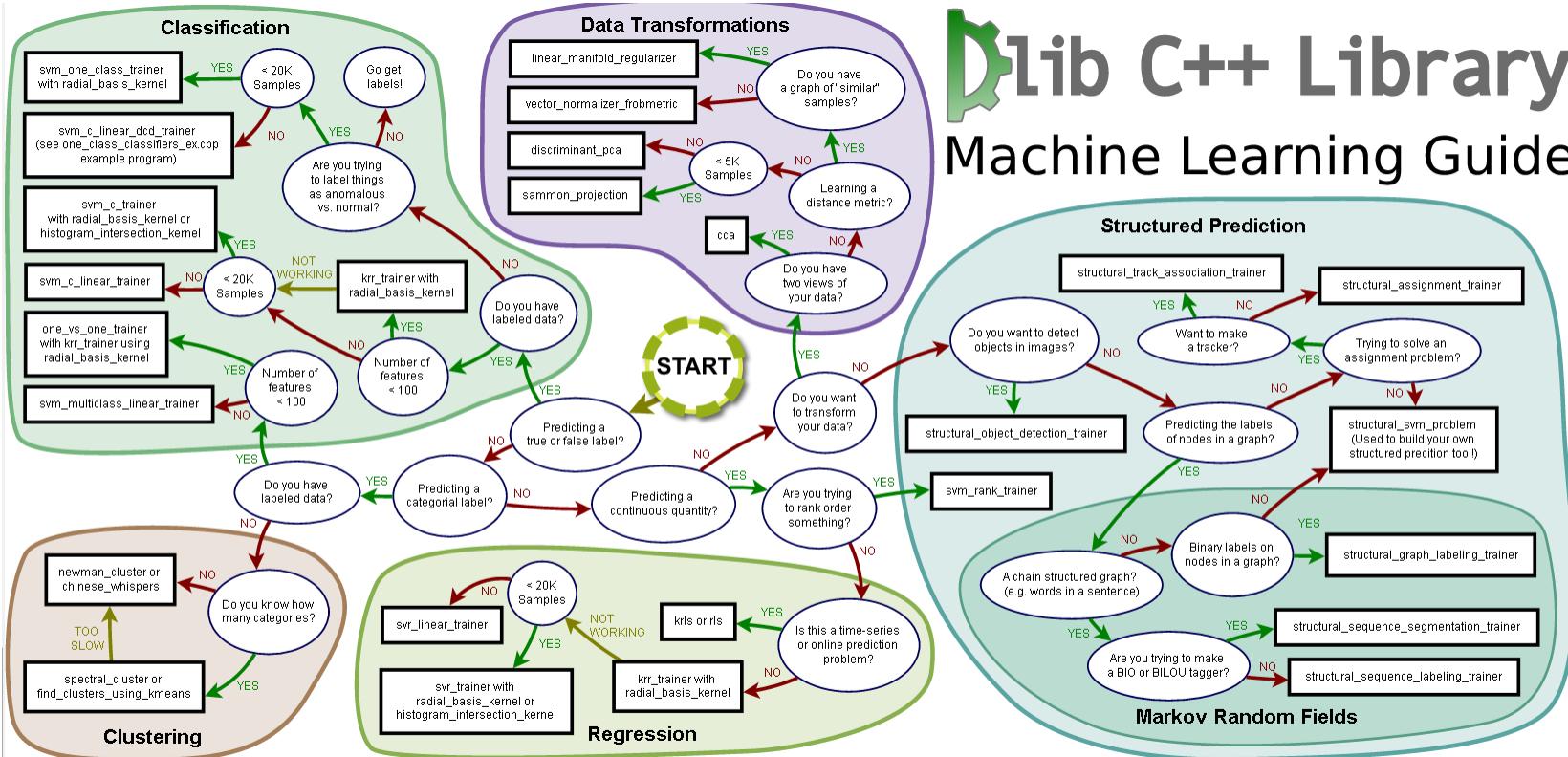
if not tmpWordsList:
    tmpInt           = 0
else:
    tmpDfObj         = _out_word_freq1(tmpWordsList, tokensListObj)
    masterWordsDsDictObj = tmpDfObj
    tmpInt           = sum(tmpDfObj.values())
    masterCnt        = tmpInt
```

Machine Learning in Finance

8. Code

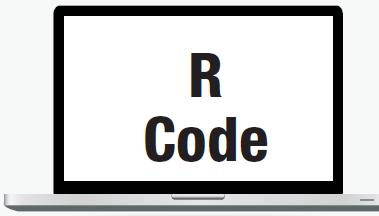
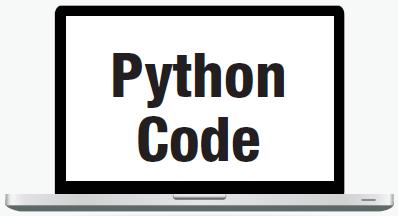
Scikit Learn





Linear Regression

```
#Import Library  
#Import other necessary libraries like pandas,  
#numpy...  
from sklearn import linear_model  
#Load Train and Test datasets  
#Identify feature and response variable(s) and  
#values must be numeric and numpy arrays  
x_train=input_variables_values_training_datasets  
y_train=target_variables_values_training_datasets  
x_test=input_variables_values_test_datasets  
#Create linear regression object  
linear = linear_model.LinearRegression()  
#Train the model using the training sets and  
#check score  
linear.fit(x_train, y_train)  
linear.score(x_train, y_train)  
#Equation coefficient and Intercept  
print('Coefficient: \n', linear.coef_)  
print('Intercept: \n', linear.intercept_)  
#Predict Output  
predicted= linear.predict(x_test)
```

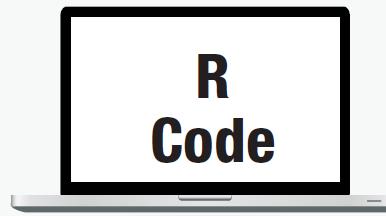
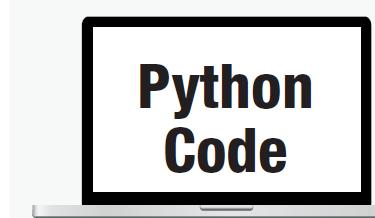


```
#Load Train and Test datasets  
#Identify feature and response variable(s) and  
#values must be numeric and numpy arrays  
x_train <- input_variables_values_training_datasets  
y_train <- target_variables_values_training_datasets  
x_test <- input_variables_values_test_datasets  
x <- cbind(x_train,y_train)  
#Train the model using the training sets and  
#check score  
linear <- lm(y_train ~ ., data = x)  
summary(linear)  
#Predict Output  
predicted= predict(linear,x_test)
```

Logistic Regression

```
#Import Library
from sklearn.linear_model import LogisticRegression
#Assumed you have, X (predictor) and Y (target)
#for training data set and x_test(predictor)
#of test_dataset
#Create logistic regression object
model = LogisticRegression()
#Train the model using the training sets
#and check score
model.fit(X, y)
model.score(X, y)
#Equation coefficient and Intercept
print('Coefficient: \n', model.coef_)
print('Intercept: \n', model.intercept_)
#Predict Output
predicted= model.predict(x_test)

x <- cbind(x_train,y_train)
#Train the model using the training sets and check
#score
logistic <- glm(y_train ~ ., data = x,family='binomial')
summary(logistic)
#Predict Output
predicted= predict(logistic,x_test)
```



Decision Tree

```
#Import Library  
#Import other necessary libraries like pandas, numpy...  
from sklearn import tree  
#Assumed you have, X (predictor) and Y (target) for  
#training data set and x_test(predictor) of  
#test_dataset  
#Create tree object  
model = tree.DecisionTreeClassifier(criterion='gini')  
#for classification, here you can change the  
#algorithm as gini or entropy (information gain) by  
#default it is gini  
#model = tree.DecisionTreeRegressor() for  
#regression  
#Train the model using the training sets and check  
#score  
model.fit(X, y)  
model.score(X, y)  
#Predict Output  
predicted= model.predict(x_test)
```

```
#Import Library  
library(rpart)  
x <- cbind(x_train,y_train)  
#grow tree  
fit <- rpart(y_train ~ ., data = x,method="class")  
summary(fit)  
#Predict Output  
predicted= predict(fit,x_test)
```

**Python
Code**

**R
Code**

Python Code

R Code

SVM (Support Vector Machine)

```
#Import Library
from sklearn import svm
#Assumed you have, X (predictor) and Y (target) for
#training data set and x_test(predictor) of test_dataset
#Create SVM classification object
model = svm.svc()
#there are various options associated
#with it, this is simple for classification.
#Train the model using the training sets and check
#score
model.fit(X, y)
model.score(X, y)
#Predict Output
predicted= model.predict(x_test)
```

```
#Import Library
library(e1071)
x <- cbind(x_train,y_train)
#Fitting model
fit <-svm(y_train ~ ., data = x)
summary(fit)
#Predict Output
predicted= predict(fit,x_test)
```

Naive Bayes

```
#Import Library
from sklearn.naive_bayes import GaussianNB
#Assumed you have, X (predictor) and Y (target) for
#training data set and x_test(predictor) of test_dataset
#Create SVM classification object model = GaussianNB()
#there is other distribution for multinomial classes
#like Bernoulli Naive Bayes
#Train the model using the training sets and check
#score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

```
#Import Library
library(e1071)
x <- cbind(x_train,y_train)
#Fitting model
fit <-naiveBayes(y_train ~ ., data = x)
summary(fit)
#Predict Output
predicted= predict(fit,x_test)
```

Python Code

R Code

kNN (k- Nearest Neighbors)

```
#Import Library
from sklearn.neighbors import KNeighborsClassifier
#Assumed you have, X (predictor) and Y (target) for
#training data set and x_test(predictor) of test_dataset
#Create KNeighbors classifier object model
KNeighborsClassifier(n_neighbors=6)
#default value for n_neighbors is 5
#Train the model using the training sets and check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

```
#Import Library
library(knn)
x <- cbind(x_train,y_train)
#Fitting model
fit <- knn(y_train ~ ., data = x,k=5)
summary(fit)
#Predict Output
predicted= predict(fit,x_test)
```

k-Means

```
#Import Library
from sklearn.cluster import KMeans
#Assumed you have, X (attributes) for training data set
#and x_test(attributes) of test_dataset
#Create KNeighbors classifier object model
k_means = KMeans(n_clusters=3, random_state=0)
#Train the model using the training sets and check score
model.fit(X)
#Predict Output
predicted= model.predict(x_test)
```

```
#Import Library
library(cluster)
fit <- kmeans(X, 3)
#5 cluster solution
```

Python Code

R Code

```
#Import Library
from sklearn.ensemble import RandomForestClassifier
#Assumed you have, X (predictor) and Y (target) for
#training data set and x_test(predictor) of test_dataset
#Create Random Forest object
model= RandomForestClassifier()
#Train the model using the training sets and check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

```
#Import Library
library(randomForest)
x <- cbind(x_train,y_train)
#Fitting model
fit <- randomForest(Species ~ ., x,ntree=500)
summary(fit)
#Predict Output
predicted= predict(fit,x_test)
```

```
#Import Library
from sklearn import decomposition
#Assumed you have training and test data set as train and
#test
#Create PCA object pca= decomposition.PCA(n_components=k)
#default value of k =min(n_sample, n_features)
#For Factor analysis
#fa= decomposition.FactorAnalysis()
#Reduced the dimension of training dataset using PCA
train_reduced = pca.fit_transform(train)
#Reduced the dimension of test dataset
test_reduced = pca.transform(test)
```

```
#Import Library
library(stats)
pca <- princomp(train, cor = TRUE)
train_reduced <- predict(pca,train)
test_reduced <- predict(pca,test)
```

Python Code

R Code

Random Forest

```
#Import Library
from sklearn.ensemble import RandomForestClassifier
#Assumed you have, X (predictor) and Y (target) for
#training data set and x_test(predictor) of test_dataset
#Create Random Forest object
model= RandomForestClassifier()
#Train the model using the training sets and check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

```
#Import Library
library(randomForest)
x <- cbind(x_train,y_train)
#Fitting model
fit <- randomForest(Species ~ ., x,ntree=500)
summary(fit)
#Predict Output
predicted= predict(fit,x_test)
```

Dimensionality Reduction Algorithms

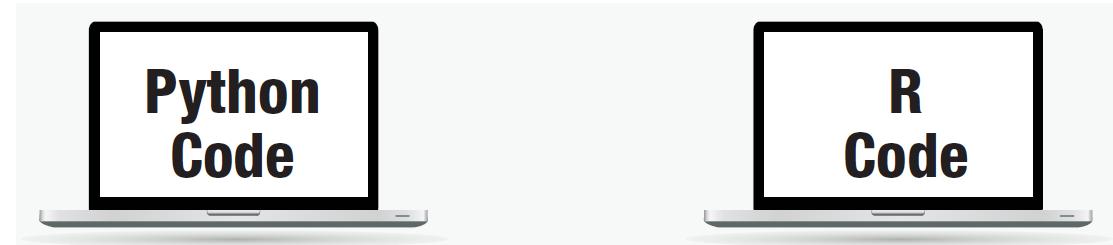
```
#Import Library
from sklearn import decomposition
#Assumed you have training and test data set as train and
#test
#Create PCA object pca= decomposition.PCA(n_components=k)
#default value of k =min(n_sample, n_features)
#For Factor analysis
#fa= decomposition.FactorAnalysis()
#Reduced the dimension of training dataset using PCA
train_reduced = pca.fit_transform(train)
#Reduced the dimension of test dataset
test_reduced = pca.transform(test)
```

```
#Import Library
library(stats)
pca <- princomp(train, cor = TRUE)
train_reduced <- predict(pca,train)
test_reduced <- predict(pca,test)
```

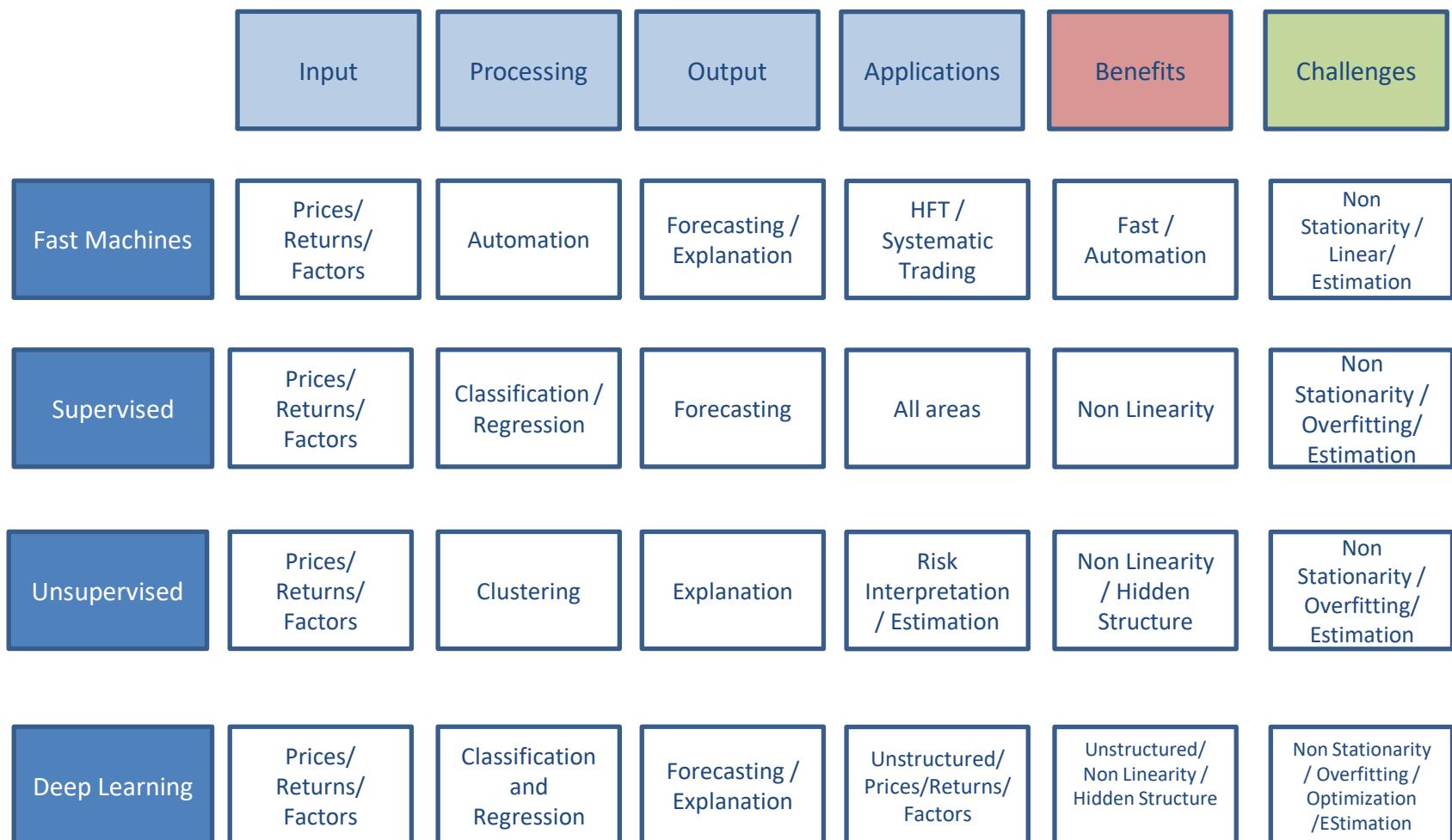
Gradient Boosting & AdaBoost

```
#Import Library
from sklearn.ensemble import GradientBoostingClassifier
#Assumed you have, X (predictor) and Y (target) for
#training data set and x_test(predictor) of test_dataset
#Create Gradient Boosting Classifier object
model= GradientBoostingClassifier(n_estimators=100, \
        learning_rate=1.0, max_depth=1, random_state=0)
#Train the model using the training sets and check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

```
#Import Library
library(caret)
x <- cbind(x_train,y_train)
#Fitting model
fitControl <- trainControl( method = "repeatedcv",
+ number = 4, repeats = 4)
fit <- train(y ~ ., data = x, method = "gbm",
+ trControl = fitControl,verbose = FALSE)
predicted= predict(fit,x_test,type= "prob")[,2]
```



Machine Learning in Finance - Instructions of Use



Machine Learning in Finance

9. Concluding Remarks

Quantitative Strategies Evaluation

Data Mining Biases

Essentially, bias arises from three processes:

1. optimization of parameters,
 2. selection/rejections of models
 3. and reuse of data.
- Optimization can result in overfitting to noise, selection/rejection introduces survivorship bias and data reuse introduces data snooping bias.
 - The combination of these biases is what is usually referred to as data mining bias. Trading strategy development is a complex process that is path dependent and there are no general quantitative solutions.
 - One sound approach in dealing with the complexity of this process and with the potential of being fooled by random results is by limiting its application.

“Solutions”

- Probabilistic Programming - Bayesian Methods
- Shrinkage
- Complexity of the models
- Data snooping
- Market regimes ?

References

- Backtesting, Harvey and Liu
- Limitations of quantitative claims about trading strategy evaluation Author: Michael Harris
- All that glitters is not gold: Comparing backtest and out-ofsample performance on a large cohort of trading algorithms Authors: Dr. Thomas Wiecki, Andrew Campbell, Justin Lent, Dr. Jessica Stauth

Integrated Big Data Trading Strategy - Background

Return Forecasting

- ◆ Models based on exogenous predictors : LFM, Auto-Encoders, ANN, VAR, GA etc..
- ◆ Momentum, Reversal, Cointegration and Mean reversion models
- ◆ Time series:
AR,ARIMA,FARIMA,GARCH,FIGARCH
- ◆ Nonlinear models : ANN, SVM, CART. Markov/Regime - Switching Models
- ◆ Chaos Theory – Strange attractors. Takens Theorem
- ◆ PCA , RMT and higher moment dynamics.
- ◆ State space models / Kalman Filter
- ◆ Deep Learning & Reinforcement Learning

Model Risk

- Estimation :
 - ◆ Bayesian estimation
 - ◆ Averaging / shrinkage
 - ◆ Random coefficient models
- ◆ Overfitting and Robustness
 - ◆ Regularization
 - ◆ X-Validation – Out-of-Sample

Optimization

- ◆ Robust optimization
- ◆ Multistage stochastic optimization
- ◆ Genetic algorithms and Quantum search algorithms

Big Data Opportunities and Challenges

Opportunities

- New Data bases
- High Frequency Macroeconomics
- Faster structured and unstructured information processing
- Non-Linear Modeling
- Adaptive Models
- Interpretation

Challenges

- Data Hoarding
- Non-stationarity
- Overfitting
- Estimation
- The curse of dimensionality
- Interpretation