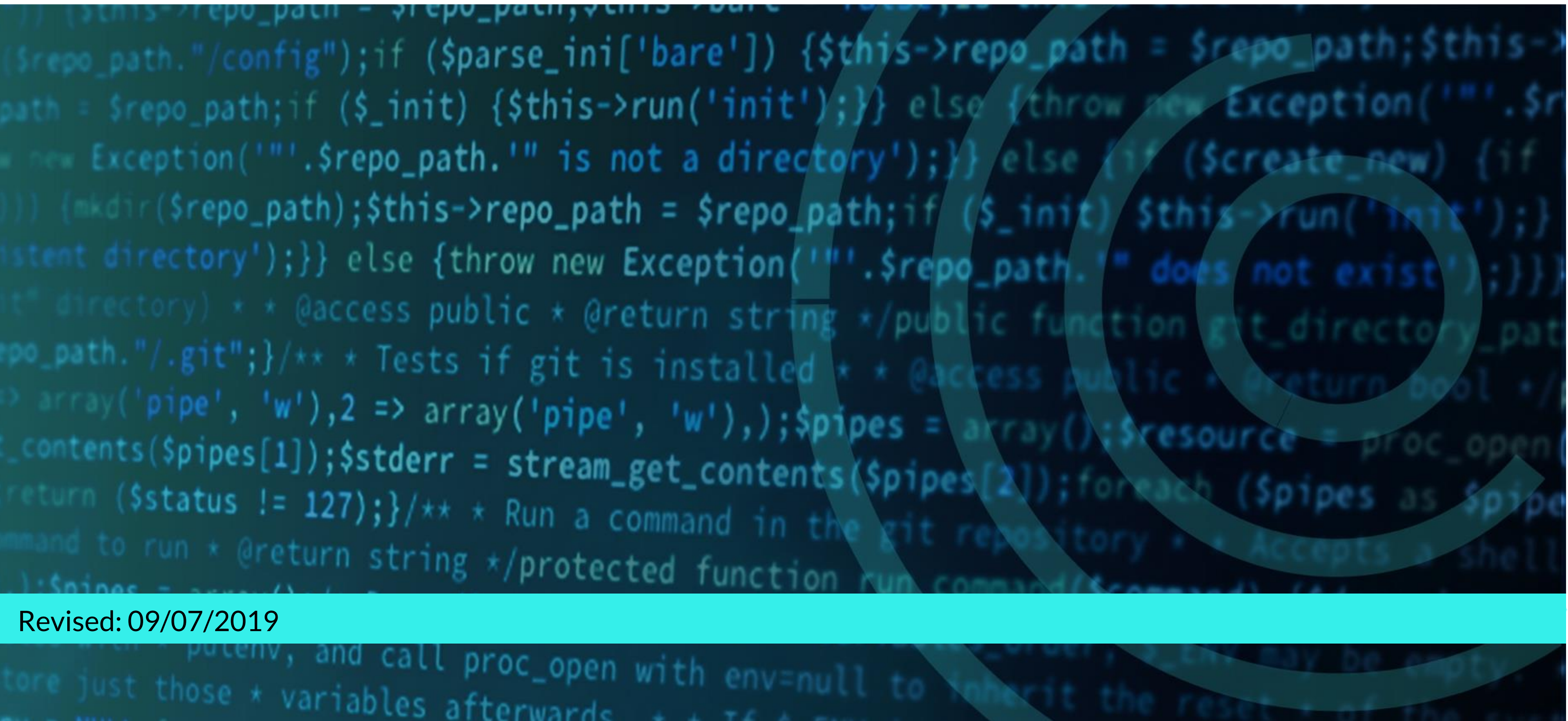
WORLDQUANT UNIVERSITY

Compiled Content Module 1

MScFE 650

Machine Learning in Finance



Revised: 09/07/2019

Table of Contents

Module 1: Introduction to Machine Learning 3

Unit 1: Introduction to Machine Learning 4

Unit 2: The Community, Projects, Events and Resources 14

Unit 3: Machine Learning in Financial Contexts 22

Unit 4: Academic Papers in Review 28

Bibliography 34

Collaborative Review Task 36

Module 1: Introduction to Machine Learning

Beginning with an overview of the course as a whole, the first module invites you to explore the field of machine learning as it applies in technology and financial companies. As a general introduction, the module seeks to describe how machine learning is used, while outlining key concepts and categories that fall within the field, and familiarizing you with the wider machine learning community.

Unit 1: Introduction to Machine Learning

Definitions of machine learning

There are two common definitions of machine learning:

1. The older definition, described by Arthur Samuel: "The field of study that gives computers the ability to learn without being explicitly programmed."
2. A more modern definition by Tom Mitchell: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

   The following is an example of the second definition:

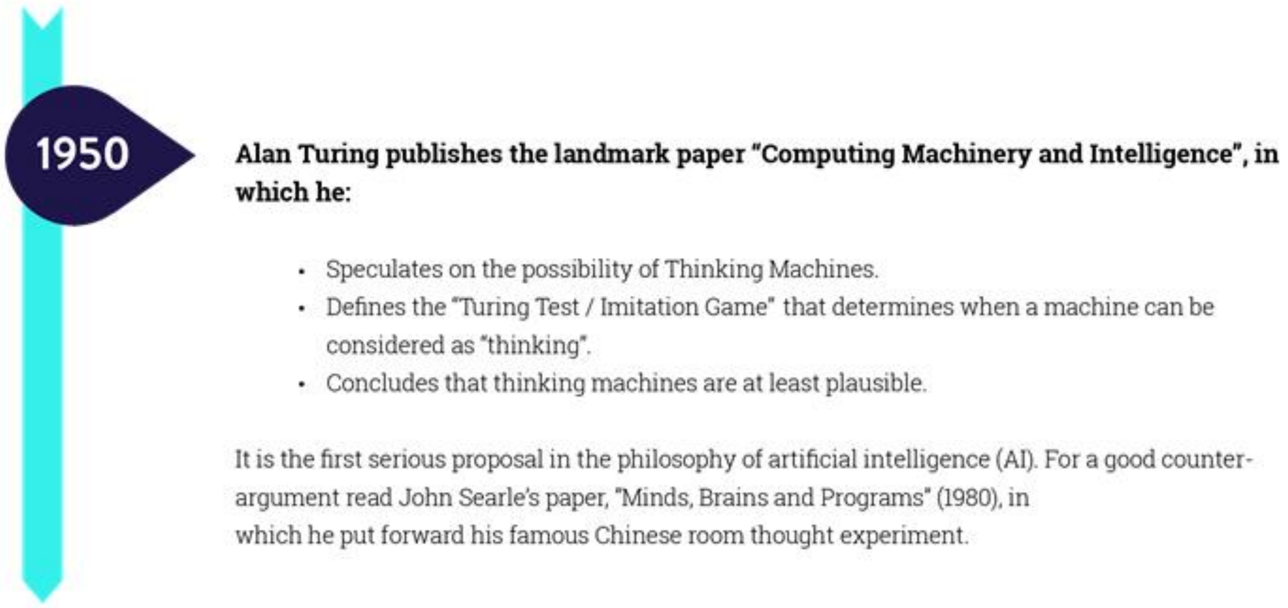
   E = Playing checkers

   T = Percent of games won against opponents

   P = Playing practice games against itself

It is the science of getting computers to act without being explicitly programmed to do so.

History of machine learning



1950

Alan Turing publishes the landmark paper "Computing Machinery and Intelligence", in which he:

- Speculates on the possibility of Thinking Machines.
- Defines the "Turing Test / Imitation Game" that determines when a machine can be considered as "thinking".
- Concludes that thinking machines are at least plausible.

It is the first serious proposal in the philosophy of artificial intelligence (AI). For a good counter-argument read John Searle's paper, "Minds, Brains and Programs" (1980), in which he put forward his famous Chinese room thought experiment.

1951

Christopher Strachey creates the first checkers program:

This establishes game AI as one of the primary metrics for progress within the wider field of AI.

1956

The Dartmouth conference establishes artificial intelligence as a scientific field:

- This enthusiastically sparks off the golden era of artificial intelligence.
- The conference boldly makes the assertion: "Every aspect of learning or any other feature of intelligence can be so precisely described that a machine can be made to simulate it".
  – McCarthy et al. (1955)

This is the philosophy of functionalism, which is the dominant theory in the philosophy of the mind. Functionalism introduces the idea of mental states being functional states and how these functional states could then be instantiated into a machine – that "we would be able to do psychology by doing robotics". Despite all the strengths of functionalism, it faces the major problem of phenomenal consciousness. For more on the philosophy of mind, refer to the following MIT course: Minds and Machines

1956 – 1974

The golden years of artificial intelligence

This was an era of intensive research, discovery and optimism, when it is relatively easy to secure funding for practically anything related to AI.

- 1958: "Within ten years a digital computer will be the world's chess champion."
  – H. A. Simon and Allen Newell
- 1965: "Machines will be capable, within twenty years, of doing any work a man can do."
  – H.A. Simon
- 1967: "Within a generation … the problem of creating 'artificial intelligence' will substantially be solved." – Marvin Minsky
- 1970: "In from three to eight years we will have a machine with the general intelligence of an average human being."
  – Marvin Minsky

1974 – 1980

The first 'AI winter' strikes

This term later coined to describe a period of dramatic stagnation owing to:

- Limited computing power.
- The curse of dimensionality.
- Moravec's paradox: High-level reasoning requires very little computation, but low-level sensorimotor skills require enormous computational resources.

Funding draws to a halt and the earlier optimism is replaced with a culture of critique and pessimism.

1980 – 1987

The second AI boom

- Expert systems restricted to very specific problems, allowing domain knowledge to be applied.
- New architectures are introduced – e.g. Hopfield neural networks.

1987 – 1993

The second AI winter

- Enthusiasm dies down, mirroring a classic economic bubble.
- The term 'AI winter' is coined.

1993 – 2001

The era of invisibility, a period defined by:

- Waiting for computation times to speed up. In 1965 Gordon Moore noticed that the number of transistors per square inch on integrated circuits had doubled every year since their invention. Moore's law predicts that this trend will continue into the foreseeable future – that computational speed will double approximately every 18 months.
- Algorithms being applied as parts of greater systems.
- AI systems receiving little to no credit.

2001 – Today

The era of deep learning

- Big data and cloud computing.
- Deep learning:
  - – Convolutional neural networks.
  - – Recurrent neural networks.
  - – Adversarial learning.
  - – Machines regularly rival or exceed human performance.

Today – Future

A subject for debate and speculation – many believe we are entering the fourth industrial revolution:

The end of Moore's law – molecular and quantum computers.

Machine learning algorithms

There are three main classes of machine learning algorithms:

1 Supervised learning

The first class we will look at is supervised learning, which refers to machine learning that is based on a training set of labeled examples.

A supervised learning model trains on a dataset containing features that explain a target variable. "The object of this kind of learning is for the system to extrapolate, or generalize, its responses so that it acts correctly in situations not present in the training set." – Sutton & Barto (1998).

There are two different types of tasks in general, namely classification and regression tasks.

1.1 Classification Tasks

Firstly, the goal of classification tasks is to learn a mapping from inputs x to outputs y, where $y \in \{1, \ldots, C\}$. Here C denotes the number of classes we are using for classification, for example $C = 2$. This is called binary classification, in which case we often assume $y \in \{0,1\}$. Classification tasks, where $C > 2$, are called multi-class classification. Another form that classification tasks can take is multi-label classification, which means that the classes we are using are not mutually exclusive. Multi-label classification "is best viewed as predicting multiple related binary class labels (a so-called multiple output model)." – Murphy (2012)

The vast majority of papers about applying supervised learning to finance involve classification-type problems. For example, in finance this may be to classify stock according to their returns which can either move up or down. In this context the system is not predicting the price or returns, but instead classifying whether it moves above or below a certain threshold (e.g. 5% increase or decrease). Accordingly, the data would then be labeled as either 1 (an up-day), –1 (a down-day), or 0 (no condition met).

Another example is the use of propensity models for targeted marketing. For instance, retail banks or insurance companies may use customer data (e.g. age, bureau rating, education, and salary) as independent variables, and whether they have opted in for a credit card as a dependent variable.

A model is then created to identify these clients (typically a logistic regression). Once the model is trained it is then used to find other clients that are likely to take up a credit card – given you phone them to market your product.

1.2 Regression

The second type of supervised learning relates to regression tasks. The difference between classification and regression is that, for classification, the response variable – the potential output of the algorithm – is discrete, whereas in regression tasks the response variable is continuous.

The most common example taught in machine learning classes for regression is a model for predicting property prices. Almost every machine learning course will touch on it, and usually base it on a multivariate linear regression. Hereby, the model is passed different features to treat as independent variables – for example postal code, size in square meters, the number of floors, and total rooms. Finally, the dependent variable is also passed to the model for training, which in this case are the properties' selling price.

We see a lot of examples of linear models in finance and econometrics. The Capital Asset Pricing Model (CAPM) as well as Arbitrage Pricing Theory (APT) come to mind.

In fact, the Chartered Financial Analyst (CFA) Institute did a study that confirmed that linear regression remains the primary workhorse for financial modeling (Zhu, Philpotts & Stevenson (2012)). A benefit of this is that making use of non-linear models allows the user to harvest more alpha and make more accurate forecasts than the majority of the competition – this is the skill set we hope you pick up in this module.

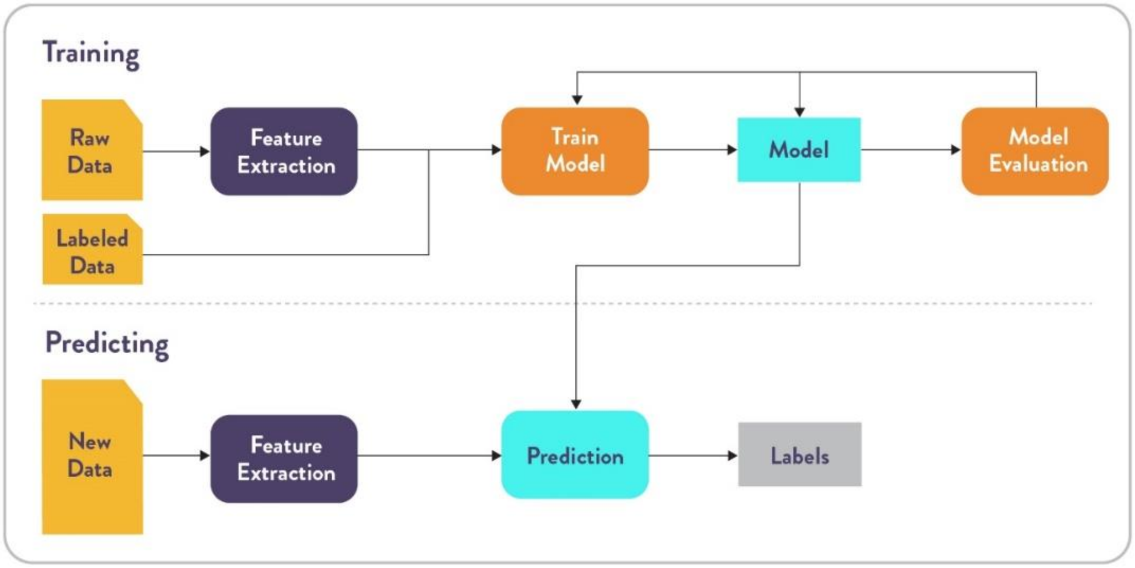


Figure 1: Supervised learning pipeline (Adapted from HortonWorks)

As you can see from the chart above, we have a typical pipeline for a supervised learning algorithm. Supervised learning is the machine learning task of learning a function that maps an input, the feature vector, to an output based on an example of input-output pairs. So, we must have labeled data.

As you can see in the pipeline, we have raw data; we then perform feature extraction/engineering and combine it with the labeled data. Thereafter, we can use this data to train the model.

Feature engineering refers to the task of transforming your data, using domain knowledge, in order to attempt to improve the performance of the machine learning algorithms. The model then gets evaluated, based on its in-sample accuracy and loss, as well as its validation accuracy and loss, then based on that we iterate by changing the model parameters and retrain the model.

When we want to score an out-of-sample set, we will give the model new data, perform the feature engineering on it, provide it to the model as input features, and then the model will give us the expected labels.

2 Unsupervised learning

The second class of machine learning we will look at is unsupervised learning. Unsupervised learning refers to learning based only on input data and no corresponding output variables, or labels. With unsupervised learning algorithms, the goal is for the system to generate its own model of the underlying structure or distribution.

We use the name unsupervised learning, because there is neither a knowledgeable expert to label data nor correct answers given to the system. In unsupervised learning, “algorithms are left to their own devices to discover and present the interesting structure in the data” (Brownlee, 2016). Unsupervised learning is also split into two subclasses – namely, clustering and representation learning.

2.1 Clustering

Grouping similar datasets together helps the user to discover interesting structures. Our task is to assign each observation to an appropriate class, more commonly referred to in this setting as ‘clusters’.

Clustering algorithms are commonly used for client segmentation to uncover hidden groupings for anomaly detection and image compression.

Examples of clustering are found in the field of medicine. Breast cancer can be better detected based on images in which certain parts of tissue will light up based on their features, aiding in the identification of tumors and abnormalities. Another example, in genetics, is the hierarchical clustering used in gene expression.

2.2 Representation Learning

"Representation learning is a set of techniques that allows a system to automatically discover the representations needed for feature detection or classification from raw data. This replaces manual feature engineering and allows a machine to both learn the features and use them to perform a specific task." – Bengio, Courville & Vincent (2014)

This is where algorithms, like Principal Component Analysis, Linear Discriminant Analysis, and T-distributed Stochastic Neighbor Embedding algorithms, come into play. They generally act to help extract features which form part of feature engineering.

Generally, they try to reduce large feature space (dimensionality reduction). This is popular within factor models where analysts will pass through many features that they think are important for predicting an asset, like economic data and zero investment portfolios (which act as a proxy for financial accounting ratios). They may then use principal component analysis or some other dimensionality reduction technique to reduce it to fewer features, but those features are very information rich.

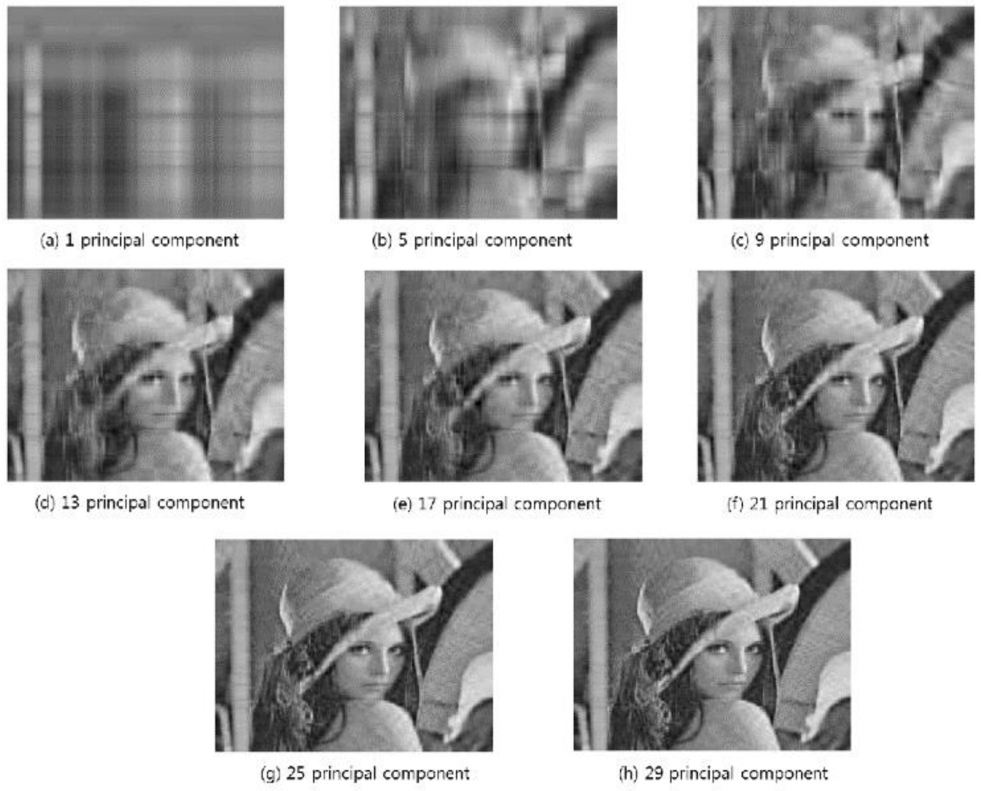

(a) 1 principal component
(b) 5 principal component
(c) 9 principal component
(d) 13 principal component
(e) 17 principal component
(f) 21 principal component
(g) 25 principal component
(h) 29 principal component

Figure 2: Principal component analysis is used for image compression

3 Reinforcement learning

Our third class of machine learning is reinforcement learning, where the algorithms are "learning what to do — how to map situations to actions — so as to maximize a numerical reward signal." – Sutton & Barto (1998)

In reinforcement learning, the algorithm is not taught which actions to take when it encounters a certain situation. Instead, the algorithm is given a situation and must apply different actions to it, each of which yields a numerical reward. These different actions yield multiple rewards and the algorithm's implicit task is to find the action which maximizes its total reward. However, these are not always linked to isolated situations as, in some cases, actions may affect consequent situations, and therefore, the potential rewards.

According to Sutton and Barto (1998) these are the two distinguishing characteristics of reinforcement learning: trial-and-error search and delayed reward. Reinforcement learning is especially useful for the optimization of strategies for tasks and learning objectives from behavior.

A very common example of the optimization of a strategy for a task is AI applied to video games (for instance, with Atari Games, video games in which you traditionally play as an agent – say, Mario from the Mario Brothers). Here you need to take certain actions like jumping to acquire coins or to avoid in-game monsters. Atari Games can be trained using the reinforcement learning paradigm. Another game-based example is from OpenAI, which released a bot for Dota 2 that is capable of beating world-class players (follow this link for a video example).

In the context of finance, reinforcement learning can be used for portfolio management, although this is a complex problem outside of our current scope. An interesting paper you can read is "A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem".

Unit 2: The Community, Projects, Events and Resources

Assorted Links and Projects

In this set of notes, we are going to take a look at some of the projects, events, and resources that are part of the machine learning community. These are especially useful to anyone interested in getting started with machine learning.

Open-source projects

The open-source machine learning projects we'll be looking at are all hosted on GitHub, and include OpenPose, Person Blocker, and an unnamed globally and locally consistent image completion algorithm. For the finance projects, we add Zipline and PGPortfolio. To begin, let's look at OpenPose.

OpenPose is an open-source algorithm that, in real time, can detect a human's body, hand, and facial key points in single images. It allows users to input an image, a video, or a camera feed, and it not only detects whether people are present but maps the key points of their body to create a 3D model. Although it may at first seem purely for fun, this type of algorithm can be used in practical and commercial applications such as body-detection for security systems, and body-capturing for gaming such as the Wii.

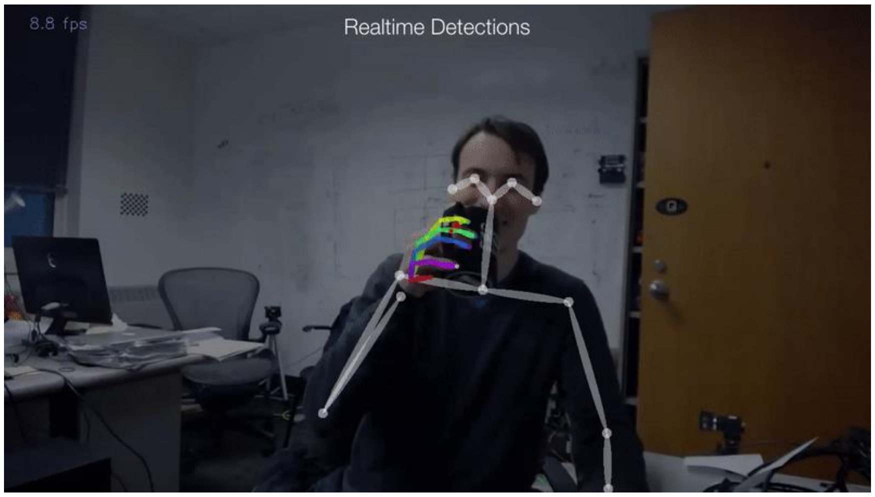


Figure 3: OpenPose body and facial recognition

Person Blocker, on the other hand, is another machine learning algorithm which uses a region-based convolutional neural network to analyze images. However, instead of mapping a person's key points like OpenPose, Person Blocker detects certain objects in the image and replaces it with a region of ‘video noise’ (or static). This can be used to block a person out of the picture, and a model could even be trained to learn what ‘Kenny’ looks like and in every photo crop out ‘Kenny’.

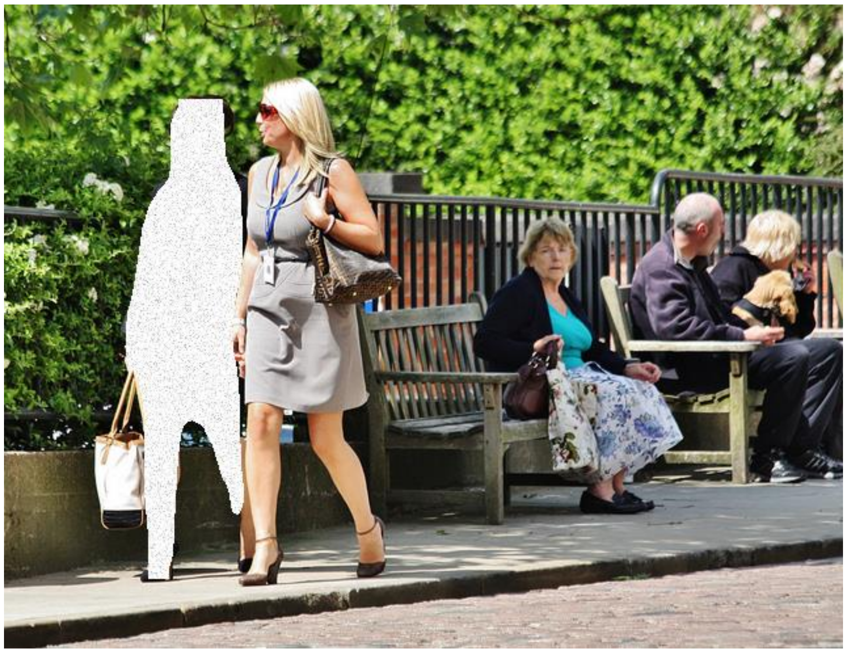


Figure 4: Image analysis and replacement by Person Blocker

Our third example is of a globally and locally consistent [image completion algorithm](), which allows users to erase parts of an image which are then replaced. The algorithm analyzes the partial image to figure out what it should look like and replaces the empty space with another consistent image.

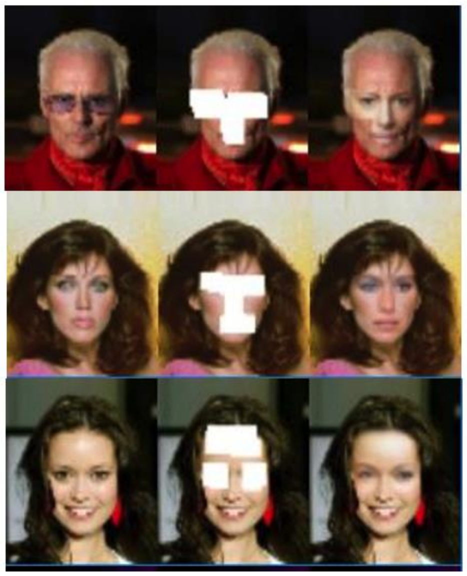
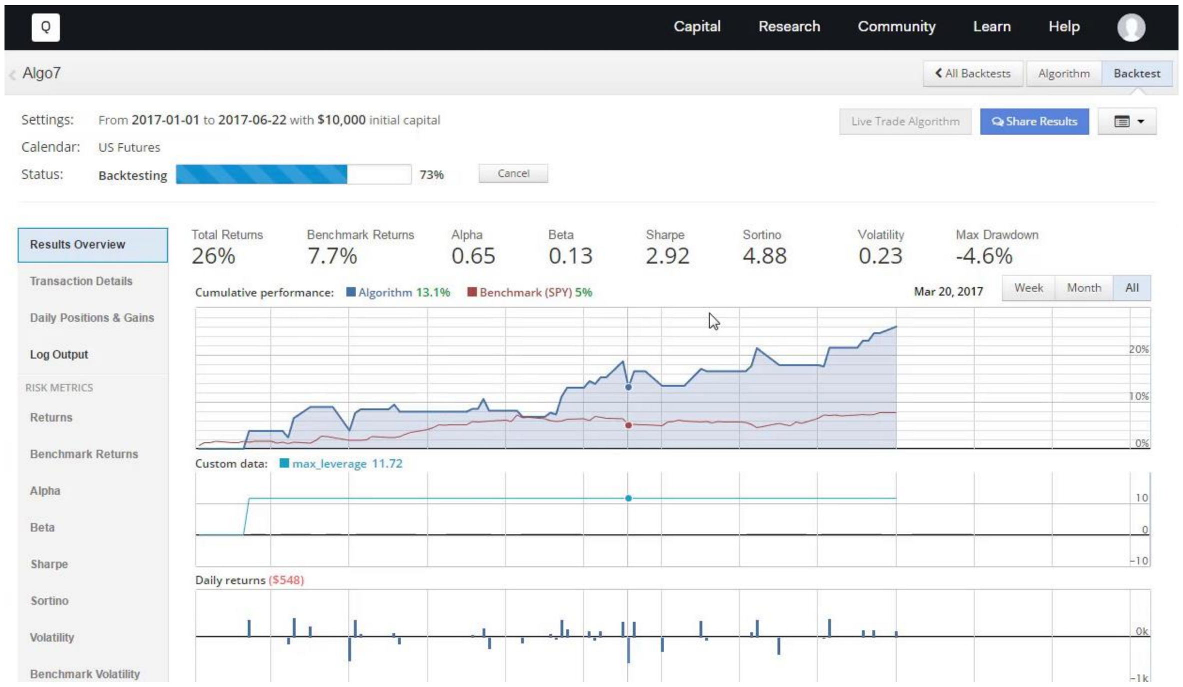


Figure 5: Examples of image replacement by machine learning algorithm

Despite the seemingly limited applicability of these three algorithms in finance, computer vision nonetheless has been put to use in financial contexts. For instance, optical character recognition (OCR) is used on bank statements, receipts, and invoices to digitally record any printed text. Another example is of photos of oil tankers. The photos are used to determine how much supply is available to match the underlying demand for oil.

Moving on to the finance examples, we chose two open-source projects: Zipline and PGPortfolio. While Zipline is not an example of machine learning, it is a very powerful tool for setting up a research environment for quantitative finance projects.

Zipline is a Pythonic algorithmic trading library. It is an event-driven system for backtesting, currently used in production as the backtesting and live-trading engine powering Quantopian – a free, community-centered, hosted platform for building and executing trading strategies.



Settings: From 2017-01-01 to 2017-06-22 with $10,000 initial capital
Calendar: US Futures
Status: Backtesting 73%

Total Returns	Benchmark Returns	Alpha	Beta	Sharpe	Sortino	Volatility	Max Drawdown
26%	7.7%	0.65	0.13	2.92	4.88	0.23	-4.6%

Cumulative performance: Algorithm 13.1% Benchmark (SPY) 5%
Custom data: max_leverage 11.72
Daily returns ($548)

Figure 6: Zipline

In relation to PGPortfolio, the paper "A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem" explains clearly how reinforcement learning can be used in managing a portfolio of stocks. The paper goes on to show how the suggested framework outperforms other online learning techniques for portfolio management. It is also worth noting that the author implemented it on a portfolio of cryptocurrencies.

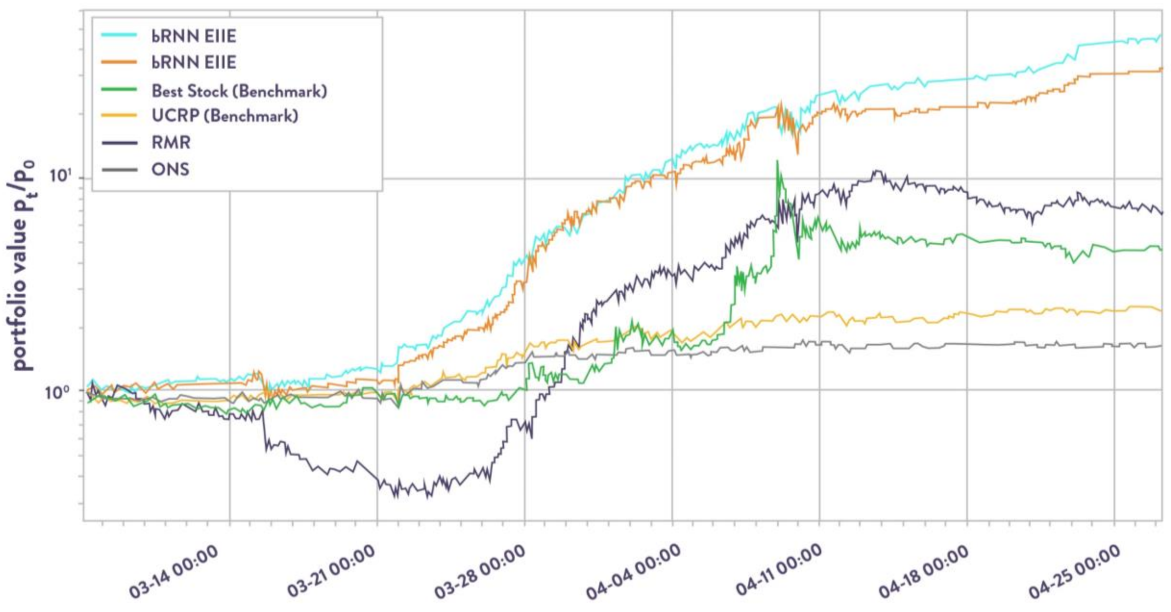


Figure 7: Back Test 3 – 2017-03-07-4:00 to 2017-04-27-8:00 (UCT), log-scale accumulated wealth.

All algorithms struggle and consolidate at the beginning of this experiment, and both of the EIIEs experience two major dips on March 15 and April 9. This diving contributes to their high Maximum Drawdown in the text. Nevertheless, this is the best month for both EIIEs in terms of final wealth. (Adapted from Jiang, Xu and Lang, 2017)

If you are interested in finding similar projects, GitHub is an ideal place for other open-source machine learning projects. The computer vision examples were sourced from the following link.

Open-source frameworks

A framework in the context of machine learning refers to any set of tools, which may include an interface or library, that supports developers and allows us to quickly build machine learning models. Frameworks make this possible as they can allow us to circumvent difficult programming challenges and simplify the process significantly.

There are many popular open-source frameworks in Python such as scikit-learn, Keras, TensorFlow, and Spark ML, to mention a few.

At this stage, exploring scikit-learn, Keras, and TensorFlow in depth is beyond the scope of introducing you to machine learning. We'll examine scikit-learn to showcase what these open-source packages offer us as developers. Later in the course, you will be introduced to Keras and TensorFlow.[1]

Scikit-learn

Scikit-learn is a free machine learning library, and is essentially a simple and efficient tool used for data mining and data analysis. The library includes a variety of algorithms used for supervised learning, unsupervised learning, model selection and evaluation, and dataset transformations, among others (scikit-learn user-guide, 2017).

The scikit-learn library is accessible to everybody and can be used in various contexts. On the technical side, the library is predominantly written in Python, and built on NumPy, SciPy, and matplotlib. It is an open-source, commercially usable library and, since it has a BSD license, scikit-learn could even be incorporated in proprietary products.

Scikit-learn not only includes a library of tools, but also some datasets for developers to play around with and test its built-in algorithms. For an example, let's look at the image below, where we can see the results of clustering algorithms applied to different datasets. The image shows us how well different algorithms perform with respect to clustering. In the first row, you'll notice a smaller circle within a larger circle, which depicts how well the different algorithms do at clustering the data. In the fourth column (under SpectralClustering), where the inner circle is orange and the outer circle is blue, we can see what the output from an effective algorithm should look like.

For the other types of patterns, it is clear that spectral clustering performs well. Our brains achieve the same results automatically.

[1] We will be focused on Python-based frameworks in this course. It is strongly recommended to use Python for machine learning, as it has become the de facto language for machine learning and offers many advantages over alternatives.

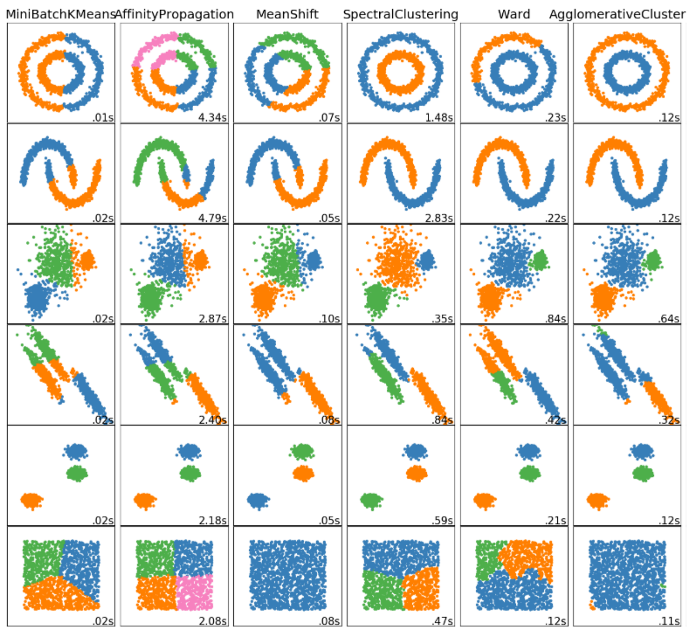


Figure 8: Results of clustering algorithms applied to different datasets

Now, let's look at some of the modeling tools included in scikit-learn. In this example we've imported a linear regression model and a function that allows us to split the data between training and testing sets.

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_selection import train_test_split
import mglearn
```

Once we've imported the model and functions, we can generate our own data, and use them in combination. After the data has been generated in the first line, random sampling is performed to sample 60 observations, which are then split into training and test data.

```
X, y = mglearn.datasets.make_wave(n_samples=60)

X_train,   X_test,   y_train,   y_test   =   train_test_split   (X,   y,
random_state+42)
```

Thereafter, a linear regression model is fitted.

```
lr = LinearRegression().fit(X_train, y_tran)
```

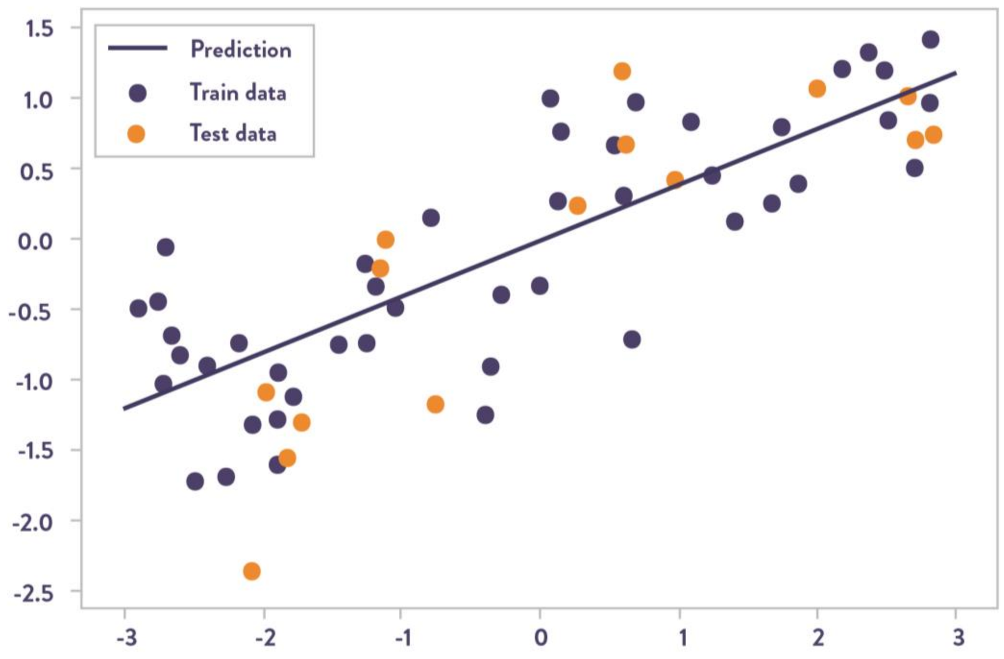


Figure 9: Linear regression

Finally, the coefficient, the intercept, and how well the model scored on in-sample (training set score) and out-of-sample (test set score) data can be printed out.

```
print("lr.coef_: {}".format(lr.coef_))
print("lr.intercept_: {}".format(lr.intercept_))
print("Training   set   score:   {:.2f}".format(lr.score(X_train,
y_train)))
print("Test set score: {:.2f}".format(lr.score(X_test, y_test)))
```

This returns:

```
lr.coef_: [ 0.394]
lr.intercept_: -0.031804343026759746
Training set score: 0.67
Test set score: 0.66
```

The take-away from these examples should be that scikit-learn allows us to fit a model to the data in six lines of code. Rather than having to create everything from scratch, open-source libraries like scikit-learn, Keras, and TensorFlow are immensely helpful, and help speed up the process by circumventing the need to code similar models and functions from first principles.

Research groups

Once you have completed your MSc in Financial Engineering, many of you will perhaps want to consider doing a PhD program or joining a research group. Here are a few of the most popular options:

- Machine Learning Group, University of Toronto
- MILA Lab
- CILVR at NYU
- Stanford Machine Learning Group
- Oxford Deep Learning Group
- Google Research
- Google DeepMind
- OpenAI
- Facebook AI Research (FAIR)
- Microsoft Research

Other online courses

One of the major objectives of this course is to help you develop skills to help yourself. This means that you will frequently be prompted to explore the internet and other resources. For those who wish to go beyond this seven-week course and explore machine learning in more depth, some highly recommended courses include:

Machine Learning: Introduction (Stanford University), taught by Andrew Ng, which is where enthusiasts usually start. From that point you can decide on two things either to learn more about machine learning in general – in which case there is the Machine Learning Specialization (University of Washington) course on Coursera – or for those who are interested in deep learning, there is a specialization dedicated to the topic also taught by Andrew Ng via an organization called Deep Learning Specialization (Deeplearning.ai).

And finally, to expand on many of the topics covered in this course, there is also Machine Learning for Finance (New York University). This program has several examples of reinforcement learning.

Unit 3: Machine Learning in Financial Contexts

To continue developing our understanding of how machine learning applies to finance, we will look at some of the ways different machine learning algorithms are used.

An important point to note is that the majority of academic papers on the topic of alpha design with machine learning focus on using classification techniques rather than regression. Regression techniques are commonly used in econometric models such as ARIMA (discussed in Module 3 of Econometrics) and GARCH (discussed in Module 4 of Econometrics). In the classification setting, it is typical to see the author predict the direction of a next move above some threshold, such as the daily volatility. Accordingly, if the stock goes above threshold it is marked as '1' and if it falls below the threshold it is marked as '–1' or a '0' (depending on how your model is structured).

Regression

The following are some examples of regression, used in finance:

1 **Forecasting ZIPs**: These are zero investment portfolios. One of the advantages about using a zero-investment portfolio is that you can represent financial accounting ratios as an engineered feature. You can read more about this in the book: *Quantitative Equity Portfolio Management.*
Fama and French also use the idea in their famous paper "Common Risk Factors in the Returns on Stocks and Bonds" (1993). What they found is that expected returns could be well explained by three factors – namely, market, size, and value.

2 **The Capital Asset Pricing Model** and **Arbitrage Pricing Theory** are based on linear regression.

3 **Supporting decisions:** Where analysts would typically fit a univariate or multivariate linear regression to a dataset, they would then, on seeing how well the factor loading explains it, decide to use it in their decision-making process. A good example of this can be found in the book: *An Introduction to Statistical Learning*. On page 18, there is an example of regression in which income as a function of education and seniority is analyzed.

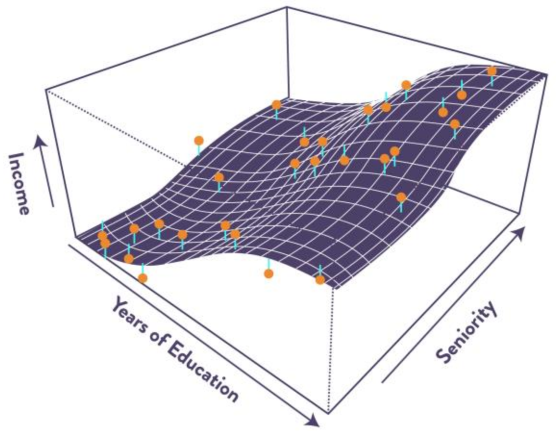


Figure 12: The plot displays income as a function of years of education and seniority in the Income dataset. The purple surface represents the true underlying relationship between income and years of education and seniority, which is known since the data are simulated. The orange dots indicate the observed values of these quantities of 30 individuals (Adapted from Hastie, Tibshirani and Friedman, 2009).

4 **Pricing interest rates.**

5 **Forecasting accounting ratios** like Earning per Share (EPS): Neural network earnings per share forecasting models: A comparative analysis of alternative methods.

Classification

On the classification side, we see the following examples:

1 **Predicting the direction of the next move for a portfolio or individual stocks**
One such paper, by Matthew Dixon, Diego Klabjan, and Jin Hoon Bang, titled "*Classification-based Financial Markets Prediction using Deep Neural Networks*", provides a good basis.

2 **Propensity modeling for targeted marketing**
This technique commonly uses logistic regression to identify which clients have a high probability of taking up a specific product, should the firm market to them. An example is scoring clients in your database to see who will likely take up a credit card.

3 **Anti-money laundering**

This is used in retail banking to identify clients who are taking part in illegal activities.

4 **Credit scoring**

In countries like South Africa, retail banks have their own credit-scoring models that take in a range of features about a client and then predict the likelihood of that client repaying the loan. These models are also used to set interest rates for clients.

5 **Regime detection**

A market can be in various regimes throughout a given period. Classification models can be used to identify which market regime we are in – for example, a trending or mean reverting regime, and then analysts can apply alpha models which will best exploit it.

Clustering

Within the class of unsupervised learning, clustering models are used for:

1 **Client segmentation**

It is typical to apply client segmentation when trying to model clients using a linear model like logistic regression. This allows the analyst to fit a different model to each of the different clusters.

2 **Regime clustering**

Unsupervised learning can also be used to identify which regime is currently active.

3 **Document retrieval**

A common example of this is when one has a document or a piece of information, such as a Stock Exchange News Service announcement, but seeks other resources like it.

Representation learning

For representation learning, we see:

1 **Factor models using PCA**

A good example is found on Quantopian: Using PCA to trade a Long/Short Basket.

2 **Identifying pairs for statistical arbitrage**

For more on this, see Pairs Trading with Machine Learning.

Reinforcement learning

3 **Portfolio management**

For more on this, see Policy Gradient Portfolio: A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem.

4 **Automated trading**

This is specifically on the sell-side, or you could market it as generating execution alpha. So, how can you spend less on transaction fees or how could you get a better price, closer to the volume-weighted average price?

5 **Options Pricing**

For more on this, see A Reinforcement Learning Approach for Pricing Derivatives.

Funds that use machine learning

The above examples not only reflect how machine learning can be used; they are also widely used in industry. Various entities in the financial sphere have incorporated machine learning into their practices. Let's take a look at six different funds that are known to use machine learning.

Bridgewater Associates

Bridgewater is famous for being the largest hedge fund in the world – using a systematic global macro approach, Ray Dalio, the founder of Bridgewater, mentions in his book, Principals: Life and Work, how they use AI to aid in decision support.

Citadel

Citadel is an American global financial institution, founded in 1990 by Kenneth Griffin in his dormitory at Harvard. It is one of the flagship success stories for quantitative funds. By 2015 it had claimed its place as one of the world's largest alternative investment firms (follow this link for more information).

Two Sigma

Two Sigma, a hedge fund based in New York city is famous for implementing the scientific method to investing. The "About Us" page on its website reads: "We apply the scientific method to investment management. Some people call us systematic. Here's what that means to us: We combine massive amounts of data, world-class computing power and financial expertise to develop sophisticated trading models. And we use technology in an effort to constantly refine our investment process and our insights into how the markets will behave. Risk is a living thing in our business, and we watch over it carefully" (follow this link for more information).

Man AHL

Man AHL is a diversified quantitative investment manager that has been a pioneer in the application of systematic trading since 1987. The following link is from their website, explaining how they have made use of machine learning algorithms for years.

A section on their website reads: "Man AHL has been conducting research in this area for decades. However, until quite recently the results of the research, which featured buzzwords in the 1990s such as 'genetic algorithms' and 'neural networks', floundered against the requirements for raw computational power and data density. With significant advances in these areas, as well as improvements in underlying theory, we have now been actively applying machine learning, in various forms, within some of our less trend-focused client programs since 2014. This year, a suite of machine learning algorithms, constrained to behave in a predominantly momentum-like manner, was incorporated into Man AHL's flagship trend-following strategies."

D.E. Shaw & Co.

Computer scientist and investor David Elliot Shaw founded D.E. Shaw & Co. He is most famous for being part of the team at Morgan Stanley, 1985, that discovered pairs trading (a form of statistical arbitrage).

As of January 2018, the firm manages more than 47 billion dollars and has offices in North America, Europe and Asia.

Point72

Point72 is a global asset management firm led by Steven Cohen. It uses Discretionary Long/Short, Macro and Systematic strategies to invest in eight offices across the globe (follow this link for more information).

Unit 4: Academic Papers in Review

"AN EMPIRICAL COMPARISON OF MACHINE LEARNING MODELS FOR TIME SERIES FORECASTING"

NESREEN K. AHMED | AMIR F. ATIYA | NEAMAT EL GAYAR | HISHAM EL-SHISHINY

ONLINE PUBLICATION DATE: 15 SEPTEMBER 2010

This paper is publicly available for students, via ResearchGate.

Notes

Every year there is a competition hosted in which models are compared in the context of business time series problems. Different algorithms are used to establish which yields the best performance. Different business-type datasets are also used, each exhibiting certain features. In the M3 Competition there were 3003 time series. Here is a link to the M4 Competition, which includes 100,000 time series.

There are very few large-scale comparison studies for machine learning regression techniques applied to time series. You will recall, from earlier in the module, that the majority of papers related to financial applications of machine learning are fixated on using classification techniques, predicting if stock is going to go up or down.

The two best algorithms reported in "An Empirical Comparison of Machine Learning Models for Time Series Forecasting" are neural networks – the authors refer to these as a multilayer perceptron (MLP) – and Gaussian Processes. They also test different preprocessing methods and how they affect performance. There have been many studies comparing the results of neural networks against traditional econometric models for forecasting and other econometric problems.

Importantly, the results of these studies are somewhat mixed, but the overall consensus is that neural networks tend to outperform classical linear techniques. In this particular case, neural networks perform best, followed by Gaussian processes. Note, however, that Gaussian processes run much faster than neural networks. The table below presents the computation time for every method per time series. What we can see here is that Gaussian processes are relatively fast.

Model	Time (Min/Series)
MLP	1.6076
BNN	2.8251
RBF	0.6917
GRNN	0.0885

KNN	0.0068
CART	0.0803
SVR	0.3099
GP	0.6268

Table 1: The computation time for every method per time series (From Ahmed et al., 2010)

Another important aspect to note is that although SVMs are one of the top models in classification, in regression they do not seem to maintain the top spot. In the second paper reviewed in these notes, we will see that SVMs are coming up again as a top performer.

In this course, we are going to focus on deep learning models (neural networks) rather than SVMs, because deep learning is setting the benchmarks in almost all applications. In finance SVMs, however, Random Forests are popular, and many papers also include neural networks. A neural network is a useful algorithm to learn. Later, we will give this special attention.

Trends and seasonality affect forecasts negatively. It is easier to forecast a stationary time series. The table below reveals the different error rates for trends, no trends, seasonal, and non-seasonal.

Model	**Trend**	**No-Trend**	**Seasonal**	**Non-Seasonal**	**Zero-Hid**	**Non-Zero-Hid**
MLP	0.1366	0.0597	0.1081	0.0811	0.0848	0.0869
BNN	0.1634	0.0716	0.1978	0.0831	0.1078	0.0937
RBF	0.2109	0.0803	0.1511	0.1191	0.1304	0.114
GRNN	0.1729	0.0689	0.1169	0.1015	0.1087	0.0958
KNN	0.1724	0.0683	0.1154	0.1011	0.1084	0.0944
CART	0.2043	0.0776	0.1354	0.1175	0.126	0.1104
SVR	0.1655	0.0658	0.1113	0.0971	0.1029	0.0935
GP	0.1565	0.0631	0.1126	0.0911	0.0981	0.0888

Table 2: The overall summetric MAPE (SMAPE-TOT) by other categories on all the 1045 time series for the LAGGED-VAL preprocessing method (From Ahmed et al., 2010)

Finally, "An Empirical Comparison of Machine Learning Models for of Time Series Forecasting" looks at the typical transformations performed on time series data, as well as three new transformations. The typical transformations performed in the data for these types of problems, performed in this order, are:

1 **Log transformation**: We have to remember that prices are often assumed to be log-normally distributed and thus returns are normally distributed.

2 **Deseasonalization**

3 **Scaling**

The additional methods considered in the paper are as follows:

1. **Lagged-values**, which is no special preprocessing. This is where the input feature vector to your model are the features at $t - 1$, where your target is at t.
2. **Time series differencing**, which is what you are used to in the ARIMA style models – it is the first backward difference and then you apply the model on the differences.
3. **Taking a moving average**, mainly to eradicate noise. Financial time series are infamous for having a very low signal to noise ratio. When using a moving the average you will want to have several moving averages with different window periods, to preserve different levels of detail.

Results

The result of the authors' study is that the MLP / neural net outperforms the other models; however, the "superiority of MLP is genuine and not due to ability to switch to a linear model".

Differencing the time series resulted in a much worse performance. This is very important. Dr Lopéz de Prado brings this up in his book regarding integer differentiation. Here it comes up again and we see that differencing time series resulted in a much worse performance. A possible explanation as to why it did worse is that the absolute level of the time series contributes important information. For example, there may be a mean-reverting behavior in the series that indicates when a series is more likely to change direction or reverse when a high or low level is reached.

While differencing for linear models is a well-understood operation, for which tests exist to notify practitioners of its need, the above is not true for non-linear models like neural networks. Dr Lopéz de Prado suggests an elegant solution that is termed 'fractional differentiated features'. He goes into detail about the problem of memory versus stationarity, which you can read more about in Chapter 5 of his textbook.

Next, we will go over some graphs from the article.

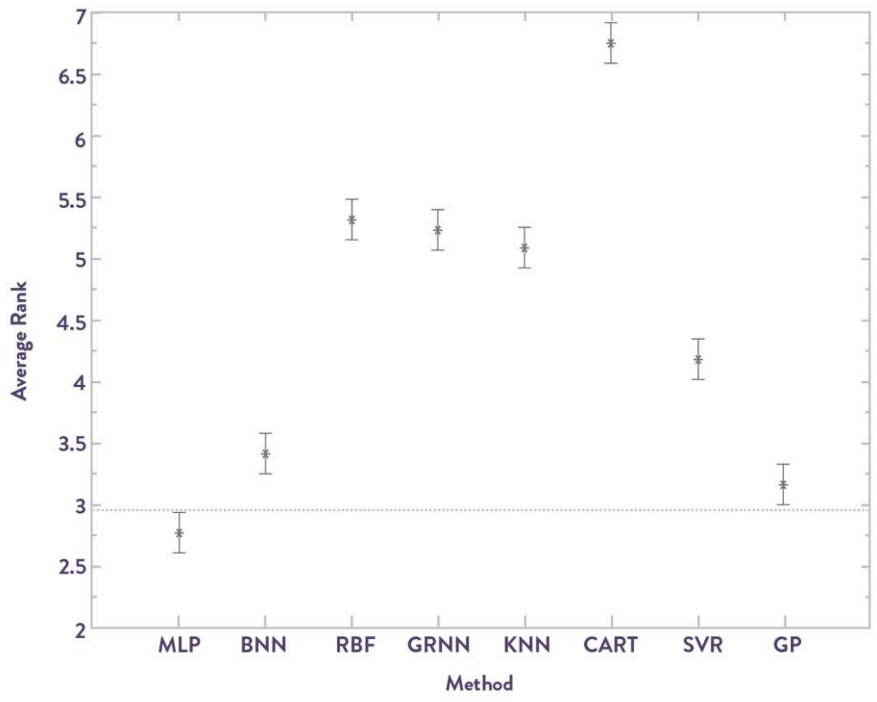


Figure 13: The average ranks with 95% confidence limits for the multiple comparison with the best test for the LAGGED-VAL preprocessing method. The dashed line indicates that any method with confidence interval above this line is significantly worse than best.
(Adapted from Ahmed et al., 2010)

The table below presents their error measure. MLP is the lowest, followed by Gaussian processes, but when one turns to Table 9 for computation time, it is clear to see that MLP is far more computationally expensive than the Gaussian process.

Model	**SMAPE**
MLP	0.0384
BNN	0.0411
RBF	0.0442
GRNN	0.0513
KNN	0.0511
CART	0.0566
SVR	0.0468
GP	0.0397

Table 3: SMAPE for the eight models for the time series example
(From Ahmed et al., 2010)

Note that the error measure that they are using, SMAPE, refers to the symmetric mean absolute percentage error, and because it is a relative measure the values across datasets can be combined into one overall figure.

"EVALUATING MULTIPLE CLASSIFIERS FOR STOCK PRICE DIRECTION PREDICTION"
MICHEL BALLINGS | DIRK VAN DEN POEL | NATHALIE HESPEELS | RUBEN GRYP
ONLINE PUBLICATION DATE: 13 MAY 2015

This paper evaluates multiple classifiers for stock price direction prediction, but in particular ensemble techniques.

Notes

This paper looks ahead, one year into the future. Naturally, a lot can change over this of time. The authors found that, for the purposes of their prediction, the top algorithms were Random Forest and SVMs.

It is worthwhile noting that the authors make a distinction between the linear methods we saw in Econometrics and the nonlinear methods found in machine learning. They particularly note that: "Although EMH is generally accepted, it was initially based on traditional linear statistical algorithms. Many researchers have already rejected the hypothesis by using algorithms that can model more complex dynamics of the financial system" (Ballings et al., 2015).

In the literature review of the papers it is clear that the majority of stock price direction prediction literature has focused on Logistic Regression, Neural Networks, K-Nearest Neighbors and SVMs. Neural Networks are by far the most common followed by SVMs and logistic regression.

	Prediction method						
	LR	**NN**	**KN**	**SVM**	**AB**	**RF**	**KF**
Schönenburg (1990)		X					
Bessembinder and Chan (1995)	X						
Brownstone (1996)	X	X					
Saad, Prokhorov, and Wunsch (1996)		X					
Kim and Chun (1998)		X					
Saad, Prokhorov, and Wunsch (1998)		X					
Kim and Han (2000)		X					
Kuo et al. (2001)		X					
Kim (2003)		X		X			
Kim and Lee (2004)		X					
Rodriguez and Rodriguez (2004)	X	X			X	X	
Huang et al. (2005)		X		X			
Kumar and Thenmozhi (2005)	X	X		X		X	
Lunga and Marwala (2006)					X		
Wu et al. (2006)							

Wang and Chan (2007)	X						
Huang et al. (2008)	X	X	X	X			
Senol and Ozturan (2008)	X	X					
Lai, Fan, and Chang (2009)							
Lee (2009)		X		X			
Ou and Wang (2009)	X	X	X	X			
Kara, Boyaciogly and Baykan (2011)		X		X			
Wei and Cheng (2011)		X					
Subha and Nambi (2012)	X		X				
Lin, Guo, and Hu (2013)				X			
De Oliveira, Nobre, and Zárate (2013)		X					
Chen, Chen, Fan, and Huang (2013)		X					
Rechenthin et al. (2013)			X	X	X		
Ji, Che, and Zong (2014)		X					
Bisoi and Dash (2014)		X					
Zikowski (2015)				X			
Hafexi, Shahrabi, and Hadavandi (2015)		X					
Patel et al. (2015)		X		X		X	
This study	X	X	X	X	X	X	X

Table 4: Algorithms for stock price direction prediction used in literature

(Adapted from Ballings et al., 2015)

Interestingly, SVMs will continually come up as a best-performing algorithm for financial time series. In the figure below we can see the median area under the curve (AUC). What is interesting is that neural nets did not perform as well as the other classifiers, such as Random Forest and SVMs.

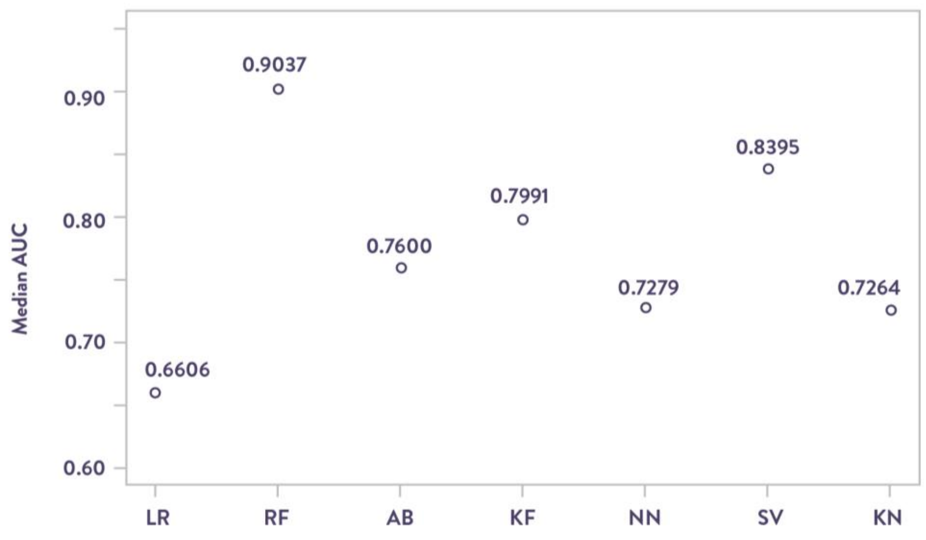


Figure 14: 5x2fc median AUC per algorithm
(Adapted from Ballings et al., 2015)

Bibliography

Recommended Reading

Ahmed, N. K., Atiya, A., El Gayar, N., and El-Shishiny, N. (2010). ‘An Empirical Comparison of Machine Learning Models for Time Series Forecasting’. *Econometrics Reviews*, 29(5-6), p. 594. DOI: 10.1080/07474938.2010.481556

Chincarini, L. B., and Kim, D. (2006). *Quantitative Equity Portfolio Management: An Active Approach to Portfolio Construction and Management*. McGraw-Hill.

Ballings, M., Van den Poel, D., Hespeels, N., and Gryp, R. (2015). ‘Evaluating Multiple Classifiers for Stock Price Direction Prediction’. *Expert Systems with Applications*, 42(20), p. 7046. DOI: 10.1016/j.eswa.2015.05.013

Zhang, W., Cao, Q., and Schniederjans, M. J. (2004). ‘Neural Network Earnings per Share Forecasting Models: A Comparative Analysis of Alternative Methods’. *Decision Sciences,* 35(2), p. 205. DOI: 10.1111/j.00117315.2004.02674.

References

Bengio, Y., Courville, A. and Vincent, P., 2013. ‘Representation learning: A review and new perspectives’. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8), p. 1798.

Brownlee, J. (2016). *Supervised and Unsupervised Machine Learning Algorithms*. [online] Available at: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

Lopéz de Prado, M. (2018). *Advances in Financial Machine Learning*. Wiley Publishers.

McCarthy, J., Minsky, M.L., Rochester, N. and Shannon, C.E. (1955). *A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence*, Dartmouth proposal.

Moravec, H. P. (1988). *Mind Children: The Future of Robot and Human Intelligence*. Harvard University Press.

Scikit-learn. (2017). *Scikit-learn User Guide*. [online], Available at: http://scikit-learn.org/stable/user_guide.html

Searle, J.R. (1980). 'Minds, Brains, and Programs'. *Behavioral and Brain Sciences*, 3(3), p. 417.

Sutton, R.S. and Barto, A.G. (1998). *Reinforcement Learning: An Introduction.* MIT press.

Trefethen, L.N. and Bau, D. (1997). *Numerical Linear Algebra*, SIAM.

Turing A.M. (1950). 'Computing Machinery and Intelligence'. *Mind*, 49, p. 433.

Zhu, M., Philpotts, D. and Stevenson, M. J. (2012). 'The Benefits of Tree-Based Models for Stock Selection'. *Journal of Asset Management*, 13(6), p. 437. DOI: 10.1057/jam.2012.17

Collaborative Review Task

The collaborative review task is designed to test your ability to apply and analyze the knowledge you have learned in the module.

Instructions

- Work through the notebook, answer all questions/problems.
- You are allowed to consult the internet, as well as your peers on the forum.
- Your answers and solutions to the problems should be added to this notebook.
- Submit your final work as an html file.
- Note that Python (version 3.6.4) has been used to calculate the solutions.
- **Hint**: In this assignment, you work with a single linear regression problem where you generate data yourself. This means that you have an idea of what the computed solution should look like.

1 Regression

1.1 We will start with the simplest possible regression problem. Solve the following system of linear equations:

$$\boldsymbol{x} = 1.0$$
$$\boldsymbol{x} = 0.5$$
$$\boldsymbol{x} = 0.7$$

1.1.1 Write the system above as a linear system of equation of the form $Ax = b$.

(3 marks)

1.1.2 Write down the normal equations, $A^{TAx} = A^{Tb}$ for this system

(3 marks)

1.1.3 Solve the system of normal equations for the system above. Is there an intuitive interpretation of the solution?

(3 marks)

1.2 We will now revisit the well-known linear regression problem – first formulating it as the solution of an over-determined system, then as an optimization problem. For solving this example, we will use the computer.

We are given data in the form of (x, y) pairs and we know that, given $x, y = mx + c$. Since one can fit a straight line through two points: we only need two pairs to fit a line. But there is a rub: the y values are contaminated by noise (at this point you may want to look at the example below), and it is not possible to find a single line that will pass through all points. We therefore need to find the line that best fits the given values. We will lead you through the solution.

Given $(\boldsymbol{x_n}, \boldsymbol{y_n}), \boldsymbol{n} = 1, \ldots, \boldsymbol{N}$, we need to solve for $\boldsymbol{m}$ and $\boldsymbol{b}$ from $\boldsymbol{m}\boldsymbol{x_n} + \boldsymbol{b} = \boldsymbol{y_n}, \boldsymbol{n} = 1, \ldots, \boldsymbol{N}$.

1.2.1 Write this as a linear system where the unknowns are $\boldsymbol{m}$ and $\boldsymbol{b}$.

(3 marks)

1.2.2 Write down and solve the normal equations for the unknowns $\boldsymbol{m}$ and $\boldsymbol{b}$.

(3 marks)

1.3 Next, we treat this as an optimization problem. Instead of using the normal equations, we now look for m and b that minimizes the mean squared error,

$$E = \frac{1}{N}\sum_{n=1}^{N}(mx_n + b - y_n)^2$$

1.3.1 Write down the $equations$ $\frac{\partial E}{\partial m} = 0$ and $\frac{\partial E}{\partial b} = 0$.

(3 marks)

1.3.2 Write the equations as a linear system and solve for m and b. How does your answer compare with the answer obtained above?

(3 marks)

It should be clear that these are the same as the normal equations.

2 Solutions: using **numpy**

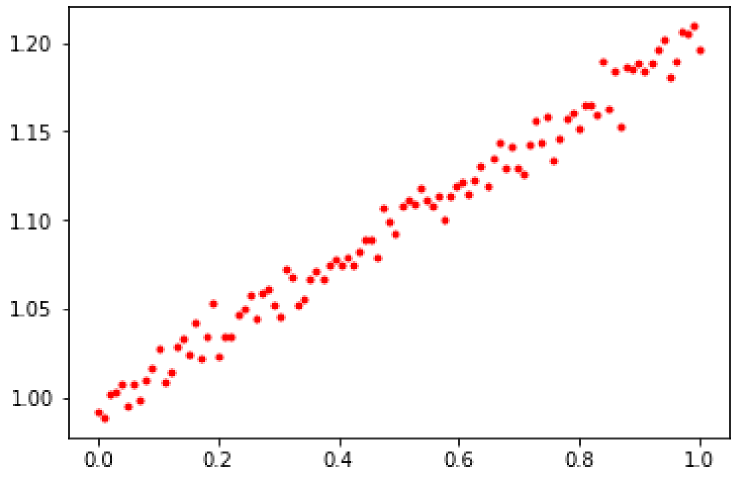
2.1

```
In [1]:# This ensures that any graphs we generate will be part of
       # the notebook, instead of opening in a separate window
       %matplotlib inline

In [2]:import numpy as np # import the numerical linear algebra model
       from matplotlib import pyplot as plt

In [3]:x_train = np.linspace(0,1,100)
       y_train = 0.2*x_train+1+0.01*np.random.randn(x_train.shape[0])

       plt.plot(x_train, y_train, 'r.')
       plt.show()
```



2.1.1 Using numpy, create $A$and b for the system $Ax = b$ (here x is the vector containing the unknown weight w and the offset b).

(5 marks)

2.1.2 Solve the normal equations for $\boldsymbol{m}$ and $\boldsymbol{b}$.

(5 marks)

2.1.3 Why do the numerical values differ from the values used to generate the data?

(3 marks)

2.2 Although it does not matter much for the simple example above, in general it is bad practice to use the normal equations to solve the least-squares problem. Surprisingly enough, the culprit is the covariance matrix A^{TA}. We cannot compute it in a numerically stable fashion. Technically, its condition number is roughly the square of the condition number of A and, in practice, this means that we lose roughly half the precision if we calculate the covariance matrix.

A full discussion is beyond the scope of this course, but the interested reader is encouraged to consult Trefethen & Bau (1997).

A numerically stable way to solve the linear least-squares problem is to first do a $\boldsymbol{QR}$ factorization of $\boldsymbol{A}$. If $\boldsymbol{A}$ is an $\boldsymbol{m} \times \boldsymbol{n}$ dimensional array with $\boldsymbol{m} > \boldsymbol{n}$ (more equations than unknowns), then the $\boldsymbol{QR}$ factorization is given by

$$\boldsymbol{A} = \boldsymbol{QR}$$

where $\boldsymbol{Q}$ is an $\boldsymbol{m} \times \boldsymbol{n}$ matrix with orthogonal columns, i.e. $\boldsymbol{Q}^{\boldsymbol{T}}\boldsymbol{Q} = \boldsymbol{I}$, and $\boldsymbol{R}$ is an $\boldsymbol{n} \times \boldsymbol{n}$ upper triangular matrix. Assuming that $\boldsymbol{A}$ has full rank, i.e. its columns are linearly independent, then it can be shown that $\boldsymbol{R}$ is non-singular. One has to be careful, but there are a number of numerically stable algorithms to compute the $\boldsymbol{QR}$ factorization.

Let's now substitute the $\boldsymbol{A} = \boldsymbol{QR}$ in the normal equations:

$$(\boldsymbol{QR})^{\boldsymbol{T}}\boldsymbol{QRx} = (\boldsymbol{QR})^{\boldsymbol{T}}\boldsymbol{b}.$$

Taking the orthogonality of $\boldsymbol{Q}$ and the non-singularity of $\boldsymbol{R}$ into account, this equation boils down to

$$\boldsymbol{Rx} = \boldsymbol{Q}^{\boldsymbol{T}}\boldsymbol{b}.$$

Since $\boldsymbol{R}$ is non-singular and upper triangular, this system is easily and efficiently solved.

What has happened here? After we calculated the $\boldsymbol{QR}$ algorithm in a stable manner, we did many of the calculations analytically (using $\boldsymbol{Q}^{\boldsymbol{T}}\boldsymbol{Q} = \boldsymbol{I}$ and cancelling $\boldsymbol{R}^{\boldsymbol{T}}$ from both sides), and that is done with infinite precision.

2.2.1 Using numpy, create $\boldsymbol{A}$and $\boldsymbol{b}$ for the system $\boldsymbol{Ax} = \boldsymbol{b}$ (here $\boldsymbol{x}$ is the vector containing the unknown weight $\boldsymbol{w}$ and the offset $\boldsymbol{b}$).

(3 marks)

```
In [4]:  # Your code here
         # End of your code

         print(Q.shape, '\n')
         print('R = ', R, '\n')
         print('Q^TQ = ', np.dot(Q.T,Q))
         print(m, b)
```

3 Solutions: using **scikit-learn** and **Keras**

3.1 First, we need to import the necessary module from scikit-learn:

```
In [7]:    # Import modules from sklearn
           from sklearn.linear_model import LinearRegression
```

3.1.1 Solve the linear least-squares problem using scikit-learn:

(5 marks)

```
In [6]:    # Fit the data to the model

           # Print the weight and offset. The values are the same as
           from # the normal equations
```

3.2 Next, import the necessary modules from Keras:

```
In [8]:    from keras.models import Sequential
           from keras.layers import Dense
           from keras.optimizers import Adam
```

3.2.1 Solve the linear least-squares problem using Keras.

(5 marks)

3.2.2 Use the Adam optimizer and experiment with different learning rates (Note: You should find everything you need on the internet).

(5 marks)

```
In [9]:    # Construct and fit your model here
           # Use a batch_size=20, epoch=300

In [10]:   # Print model weights, might not be exactly the same as
           above.
```