

ABSTRACT

There are a large amount of data set collected by 854,064 players from the on-line game called Axon. Using the on-line game data set, how individuals learn in practice is identified by fitting possible underlying learning models to the data set. The power law model is fitted to the data set, as well as other several functions; exponential, polynomial, step, and piecewise functions combined with various shape of functions. This analysis showed that many individuals follow the power law rather than other practice law in on-line game practice. Additionally, the trade-off between exploitation and exploration in reinforcement learning as the skill acquisition process in practice is discussed, based on the result of the curve fitting.

ACKNOWLEDGEMENTS

Firstly, thank Dr. Stafford for the kind supervision. He has supervised thoroughly how to appropriately deal with the unexpected problems in this data analysis, and encouraged me a lot when I faced many difficulties. Also, thank Dr. Vasilaki who is a lecturer in Computer Science at University of Sheffield, because the artificial intelligence algorithm, this analysis uses for the clustering task, was discussed through the Adaptive Intelligence model. Finally, the large amount of game data set is given by PRELOAD for the Wellcome Collection (Stuart, 2009).

Contents

1	Introduction	1
1.1	Background and Motivation	1
1.2	Problem Definition	2
1.3	Aims and Objectives	3
1.4	Project Management	4
2	Data Curation	5
2.1	Data Cleaning	6
2.2	Data accommodating	7
2.2.1	Original variables in the data source	7
2.2.2	Hidden variables calculated from the original variables	7
2.3	Data Deformation	8
2.3.1	Pickle file format	9
2.3.2	Data storage and index table	10
2.3.3	Multiprocessing	10
2.4	K-Means Clustering with Competitive Learning Algorithm	11
2.4.1	K-Means clustering	12
2.4.2	Competitive learning	13
2.4.3	Normalisation	15
2.4.4	Procedure of the algorithm	17
2.4.5	Local optima and optimal number of clusters	18
2.4.6	Dealing with dead unit	20
2.4.7	Centroids	24

3	Function Fitting	27
3.1	Functions	28
3.1.1	Parametric Method	28
3.1.2	Functions for Single Curve	30
3.1.3	Piecewise Function	32
3.2	Method	34
3.2.1	Squared Lost Function	34
3.2.2	Training Set and Validation Set	37
3.2.3	Over-fitting and Regularisation	39
3.2.4	Choice of Breaking Point	40
4	Discussion	43
4.1	The Best Learning Curve	44
4.1.1	Power Law and Exponential Law	44
4.1.2	Single and Piecewise Curves	45
4.2	Exploitation and Exploration	49
	References	51

Chapter 1

Introduction

1.1 Background and Motivation

Shape of learning curve may reflect different underlying learning processes. For example, in insight learning, the learning curve may follow slightly fluctuated level and sudden dramatic increase at some points. If this feature between performances and practises is trained to identify the underlying learning curve, it would result in a step-like model. Another example is a power curve. It is well known "law of practice" in Psychology (Newell & Rosenbloom, 1981). Many different skill acquisition tasks can be described well under the power law (Lacroix & Cousineau, 2006). Howard (2014) said that power law is "correct and general" to explain learning processes. If subjectives show a learning performance of power law, their learning performance will be enhanced progressively early in learning, but will become slower and slower, and then will reach to asymptote. Therefore, "curve fitting with benefit of a model" would provide effective understanding of skill acquisition and learning processes (Newell & Rosenbloom, 1981).

Power function has been used as a fundamental and popular practice law in Psychology, such as Ericsson et al. (1993) proved the expert performance theory that "performance increases monotonically" following with the power law, although "the form of the practice law may also seem to be an unsolvable technical issue, rather than an important psychological question" (Heathcote, Brown, & Mewhort, 2000).

Individuals may show different shapes of learning process beyond power function.

Some may follow exponential law, and others may have "multi-step changes in behaviour" if experiences and insight may reinforce each other and can be associated to one another (Gallistel, Fairhurst, & Balsam, 2004). There are several possible functions which might fit to the curve of skill acquisition, and may describe different learning processes; step function, power law, piecewise power function, and so on. Because the power law is often said that it is the outcome of averaging these possible functions, individuals can be categorised into several groups in which they show similar learning patterns on these possible functions.

In contrary with the traditional approaches for investigation of learning curve which usually brought experts into a lab to test skill acquisition, this analysis uses a large amount of data set collected from on-line game. The traditional approaches lack "comprehensive details" of subjectives' training and the exact feature of "the history of their practice which is related to the skilled performance" (Stafford & Dewar, 2014). Also, it has a limited number of subjectives. It might thus result in the power curve for the investigation of skill acquisition.

However, the computer game provides an abundant "record of every action in the history" of practice, as well as a large number of samples of players (Stafford & Dewar, 2014). This large number of data set and records may be able to allow identifying the underlying practice law which are hidden under the power law of averaged learning curves.

1.2 Problem Definition

It has been often argued that the power law is an "artifact of averaging across individual learning curves" (Donner & Hardy, 2015). Power law of skill acquisition has a single monotonic power curve to predict individual performance. However, learning processes might be dynamic, and show various shape of curves. Individual may show a "step-like performance" (Gallistel et al., 2004) represented as "aha! moment" (Kounios & Beeman, 2009), which is involved in the abrupt increases in practice.

Even among monotonic shape of curves, whether the power law can be representative of learning curve or not is controversial. Heathcote et al. (2000) strongly insisted

that the power law should be repealed, and that the exponential function with three parameters can describe the not-averaged data set better than the power functions.

Gaschler et al. (2007) tested the exponential function with various parameters to the data sets of chess game practice, and they found that "the exponential function with three parameters was better than the power function in fitting" (Gaschler, Progscha, Smallbone, Ram, & Bilalic, 2007).

Individuals may follow the learning process showing the shape of exponential curve rather than the power law. If individuals in the exponential law are averaged together, a grouped learning curve will be the shape of power law (Murre & Chessa, 2011). In addition, if all other possible shapes of learning curves including the exponential curves are squashed into one learning curve, the power law has a possibility to be yielded as the outcome from the averaging those learning curves.

Moreover, as explained above, the traditional way to investigate the practice laws in the laboratories may restrict the amount of data, as well as the history of practice for the analysis (Stafford & Dewar, 2014). The problem of this lack of data may cause different results rather than the true information.

To sum up, problems in current practice laws can be summarised as below,

- not being able to categorise the various learning processes,
- averaging possible individual learning curves,
- not enough information for the identification of skill acquisition,

and those problems are dealt with, and discussed in this analysis.

1.3 Aims and Objectives

In order to solve these three main issues which is explained above, this analysis exploits the on-line game data set called Axon. In which, game play data of players are recorded in accordance with their attempts. Although the collected data set has a lot of information including the history of their practice, it also includes wrong information, such as the attempt time is recorded wrongly and the second attempt time is later than the first

trial time. Therefore, this wrong information has to be filtered before the analysis is implemented.

At the same time, it is assumed that individuals may show different learning patterns. Therefore, players will be categorised into the several groups in which they show the similar learning processes.

Because the on-line data set has a large amount of data, the performance of these processes will be very slow, and it may take a lot of time to analyse such the large amount of data set. In order to solve it, several methods will be introduced, such as using multiple cores of CPU.

To understand which are the underlying learning processes under the averaged power law, two theories will be tested that the power law model is the averaged curve according to individual exponential curves (Murre & Chessa, 2011; Gaschler et al., 2007), and that many tasks show step-like performance (Gallistel et al., 2004). Therefore, function fitting for the possible underlying models; polynomial, exponential, power, step, and piecewise functions with various combinations, to the data set will be implemented.

Through these tasks, it can be identified which underlying learning curves disappear under the averaged power law model.

Finally, the trade-off in skill acquisition in practice, reinforcement learning, will be able to be discussed after identifying the individual learning curves.

1.4 Project Management

Two gantt charts are attached in appendix. The first one is the gantt chart for the project time management, and the second chart is the actual time management.

Chapter 2

Data Curation

A large number of data were acquired through the online game named Axon which is developed by Preload for Wellcome trust (Stuart, 2009). In total 1,201,516 machine identities (or players) played the game over 4 million times. The raw data set is fundamentally comprised by the score, date, time of plays and so on in accordance with machine identities which may represent each individual. Undoubtedly, it can be possible to extract information from the data set on how people practised to get a higher score. For example, it shows how much time they played for each score. However, the raw data set of on-line game seems quite noisy to discern those factors which affect player's learning. It contained many 'undefined' or 'unrepresentable values', and some values are not valid, such as the starting play time is later than the time play finished. For this reason, the information on what the analysis needs cannot be identified directly from it.

Thus, the data source needs refining processes until the gemstone of data set could gleam with the evidence of valuable information. In other words, the big data set has to be curated. Stonebraker et al. stated that "data curation is the act of discovering a data source(s) of interest, cleaning and transforming the new data, semantically integrating it with other local data sources, and deduplicating the resulting composite" (Stonebraker et al., 2013).

Below are the tasks in which steps of data curation are largely expected,

1. Cleaning raw data set,
2. Incrementally accommodating new data entities,

3. Normalising scores grouped by the same players,
4. Clustering players, showing the similar patterns.

In particular, data curation tasks followed methods of two previous studies (Stafford & Dewar, 2014; Stafford & Haasnoot, 2016). Accordingly, the curation tasks of cleaning, grouping by machine IDs and accommodating will be explained in the following section. After that, other tasks for data deformation and clustering will be examined.

2.1 Data Cleaning

What has to be noted before data cleaning is that it has to be implemented under the grouped data by the same machine IDs. In total 4,038,802 plays were sorted by players having the same machine IDs, and then 1,201,516 players were detected. The criteria according to the previous two studies (Stafford & Dewar, 2014; Stafford & Haasnoot, 2016) are as follow,

1. players who did not play at least 15 games,
2. players who attempted more than 300 times,
3. data which lack valid longitude or timing information for each attempt,
4. players with discontinuous game play attempts.

The reason of discarding those who played less than 15 attempts is because those cannot show a causal relationship between attempts and scores. In this step, xxx individuals were deleted. The reason for second step is that plays more than 300 attempts did not show many differences after 300 attempts, and they were removed. Next, players having unrecognisable time information were also discarded. Finally, game play data for individuals were recorded discontinuously. In other words, there are non-recorded game play data between the first attempt and the last attempt for each players. The reason why this problem happened is unknown, though it is obvious that individuals with this issue have to be filtered appropriately.

As a result, raw data of 854,064 players with 4, 038,802 plays were cleaned, and then reduced to 22,832 individuals.

2.2 Data accommodating

The next step of data curation is to incrementally accommodate hidden but obvious features from the previous cleaned data set. It is originally constituted by 8 basic categories. After data accommodating task, 7 additional hidden variables would be derived.

2.2.1 Original variables in the data source

When people play the on-line game, 7 basic features as follow is recorded,

- machine identities,
- scores and attempt numbers,
- date, hour and minute,
- latitude and longitude.

When players access the on-line game, the tracking code written by Preloaded records machine identities. Machine IDs may be considered as individuals who actually play the game. Preloaded inserted tracking code to the on-line game, records a machine identity at each time the game was loaded, as well as the result of the game at the end of plays, and play date and time. Attempts and scores were recorded in the order of the time.

The information of locations where the machine ID accessed the game is also collected approximately at the maximum of city-block level.

2.2.2 Hidden variables calculated from the original variables

From the basic variables which are recorded when people play the game, other features that may imply useful information of how players are influenced during learning processes to get higher marks can be derived as below,

- time difference,
- local time,
- gap types (no, long, short and sleep gap),

- a number of total plays.

The basic time information; date, hour and minute is combined together. Then, time differences between successive attempts are calculated by subtracting current game start time from its previous attempt time. Therefore, the first attempts do not have value of time differences.

Next, local times for each play were calculated in seconds, using the formula, " $local\ time = UTCtime + (longitude24 \div 360), modulo24$ " (Stafford & Haasnoot, 2016). Local time would facilitate comparison of players in different time zones, thus indicating lifestyle choices. There were data that local time could not be calculated because of missing track of latitude and longitude, and those were filtered at the data cleaning task.

On the basis of time difference, Stafford et al. (2016) classified individuals into 4 groups, according to "the nature of the timing of their first 15 attempts at the game". First group is 'no gap' for those who had less than 15 minute break between each play. Second group is 'wake' in which players are assumed to be having a rest in their waking hours if they have "a single gap between 7 and 12 hours". Working hours is time between 5 am and 12 pm. Those who rested for between 7 and 12 hours in working hours were categorised as 'sleep'. Finally, all other individuals are classified as 'no category'.

Furthermore, for more detailed analysis, 'rest' which is defined in 'gap types' categorisation was divided with 3 types. Long gap, short gap and sleep gap were determined when players had a rest 'between 7 hours and 11 hours', 'below 15 minutes', and 'not in working hours' respectively.

As the term of hidden variables suggests, there are possibilities that more factors can be mined from the data set and be incrementally accommodated to it if appropriate methods are implemented.

2.3 Data Deformation

CSV format has been used as a basic data file format for previous study.

CSV is an abbreviation of Comma-Separated Values. A data record of CSV format file is a line formed by one or more elements, and commas separate those element values to be distinguishable (plain text format) (Shafranovich, 2008). According to Idris (2014), it is straightforward for those format to load and store data-set from the data storage onto working place; ease to generate, read and edit manually, offer versatility in most programming languages, but not efficient because exploiting plain text format "take a lot of space". This might result in consuming more time to handle such a big data set than using binary format.

Simply, the data set needs to be stored in memory unit of machine(s) where the analysis processes would operate. However, these are true only when data size is manageable. Because fitting all data-set of such a large amount of data in computer memory takes many spaces. In some cases, loading on computer memory cannot be possible depending on computer capability and setup. Thus, CSV format is not a very efficient way to manage such a large amount of data (Idris, 2014). To solve those problems, two approaches of being able to reduce storing spaces and of enabling faster speed for analysing were applied; converting CSV file format into Pickle format, and splitting data set into several pieces.

2.3.1 Pickle file format

Pickle file format can provide "a high level of data compression such as zip, bzip and gzip" and also faster than CSV (Idris, 2014), because it is "binary protocols for serializing and de-serializing a Python object structure" (python on-line document, n.d.-a). Furthermore, it provides an easy approach to deal with converting Python object hierarchy into a binary stream and vice versa, with the notion of 'pickling' and 'unpickling'. Moreover, pickle file format can automatically represent a large number of Python types (python on-line document, n.d.-a).

Therefore, curated data set after cleaning, accommodating, and deformation process is stored as binary file format, and for the binary file format, pickle specified for Python is used.

2.3.2 Data storage and index table

Although the big data set can be converted to binary file, it still takes a large amount of storage. And it also causes RAM memory optimisation problem. When data curation tasks are performed (of course, as well as data analysis tasks which will be discussed in the next session), processing speed became slower and slower over time. This might be because data occupies all spaces of hardware memory, and exploits virtual memory on main memory unit. Basically, RAM memory provides very high speed compared to hard disk, and it is even faster than Solid State Drive (SSD) which is often considered 'fast'. If data take all RAM memory spaces, they transfer addresses of some part of it to virtual memory. This task causes many interruptions to the tasks, and eventually decreases processing performance significantly. And depending on operating systems, Python kernel may crash.

Therefore, loading only necessary data to a specific task will be a solution to keep processing performance fast. Whole data set was separated by each type of variables. To secure and not to lose player information within the divided data, addresses of all data have to be indexed directing player ID. As a result of data deformation task, 15 pickle data files were generated; eventLabel, date, hour, minute, latitude, longitude, eventValue, play_filter, comb_time, diff_time, local_time, long_gap, shor_gap, sleep_gap, gap_type, and total_plays, which are indicated by individual ID information, eventAction.

2.3.3 Multiprocessing

Because the raw data set is a large amount, the process time for data curation would take long time. Commonly, using multithreading is the easiest way to solve this operating time problem. However, unfortunately Python does not allow multiple statements at a kernel (?, ?). Therefore, the multiprocessing method which uses multiple cores in CPU is used.

In using a single core of Intel xeon processor, E5-2698V3, the data curation task took almost about 3 days. However, using multiprocessing under the 32 threads of 16 multiple cores with 2.3 GHz Processor Base Frequency showed the high performance; it only took below 3 hours.

2.4 K-Means Clustering with Competitive Learning Algorithm

In order to identify learning curves of individuals, the first thing to do is to cluster players into several groups in which they show similar learning patterns. Traditional ways of identifying individual learning curves is either averaging all data of test subjects or fitting different possible underlying functions to observed population data-set (Newell & Rosenbloom, 1981; Gallistel et al., 2004; Donner & Hardy, 2015)

A well represented learning curve in psychology might be a "smooth power law of diminishing gains" (Donner & Hardy, 2015). For example, Howard (2014) tested function fitting on chess performance and verified that "power function fit is best".

However, in contrast, Gaschler et al. (2014) stated that "the exponential function" is "better than the power function fitting" for learning curves of chess players. These different results of the same observation of chess game would result from that the power law is an artifact which averages many divergent shapes of exponential curves (Heathcote et al., 2000; Murre & Chessa, 2011) as well as other possible underlying curves.

Therefore, before analysing the on-line game data set, it would be a better assumption that there are many different shapes of learning curves depicting individuals' learning best, rather than only one learning curve.

In contrast with classification tasks that classify labelled data-set, clustering is an unsupervised classification task which automatically groups similar data without pre-defined and labelled classes (Hackeling, 2014), and it generates the same result as classification algorithm does (Harrington, 2012).

Among several clustering methods, K-Means clustering was used with an artificial neural network algorithm, competitive learning. It is because K-Means clustering algorithm in high dimensions is computationally very slow and the key method of K-Means (comparing distance in 2 and 3 dimensions) would not work for satisfying outcome in high dimensions (Daumé III, 2012).

Hence, there is possibility that the data-set may not properly be clustered, as the on-line game data-set is comprised of 15 dimensions. Compensating K-Means clustering with neural network algorithm would provide one of the solutions for the high

dimensions data-set.

2.4.1 K-Means clustering

The name of K-Means is because it discovers k unique clusters, and the mean values of those clusters represent the centres of each cluster. The key idea of the algorithm is to compare only distances between all data and arbitrary number of cluster centres. After centroids of clusters were found, all data can be assigned to their nearest centres. In the same manner, centres of clusters can be computed after data is allocated to clusters.

Again, there are no labels or predefined information of classes in the data set. Thus, it is unknown which approach has to be implemented first. The ways to tackle this are to start with initial random point, to iterate calculation of centre position, and to rearrange elements in clusters. Through iterating these three approaches until stopping criteria is satisfied, k means eventually converge to local optimisation, and comprise k number of clusters.

There are two terminating criteria in K-Means clustering algorithm. The first one is using a threshold for the difference of cost functions (2.2) through subsequent iterations (2.1).

$$J_i - J_{i-1} < \theta \quad , \text{ where } j \in K, \quad (2.1)$$

and cost function J is

$$J(\mu, \mathbf{C}; \mathbf{D}) = \sum_n ||x_n - \mu_{C_k}||^2 = \sum_{k=1}^K \sum_{i \in C_k} ||x_i - \mu_k||^2, \quad (2.2)$$

where μ is mean of cluster, \mathbf{D} is (the number of parameters) data-set, \mathbf{C} is clusters in the data-set, x is data, N is the number of data, and K is the number of clusters.

The other one is to use a threshold for the difference of centre positions between successive iterations (2.3).

$$x_i - x_{i-1} < \theta \quad , \text{ where } i \text{ in } C_k. \quad (2.3)$$

Below is pseudo-code of K-Means clustering algorithm,

for all k such that $0 \leq k < K$ **do**


```

    randomly initialise  $\mu_k$ 
end for
while stopping criteria (2.1) or (2.3) do
    for all  $k \in K$  do
        for all  $i$  such that  $0 \leq i < N$  do
             $C_k \leftarrow \arg \min_x ||\mu_k - x_i||^2,$ 
        end for
    end for
    for all  $k \in K$  do
         $\mu_k \leftarrow \text{mean}(C_k)$ 
    end for
end while
return  $C, \mu$ 

```

At first, the positions of centroids are randomly initialised. Then, the first for loop in the while loop assigns each data to nearest clusters. The second for loop calculates the centres of clusters. By iterating these two tasks, all centroid of clusters move slightly towards the centres of each cluster, and eventually converge to local optima. For on-line game data analysis, stopping criterion (2.3) was used.

Moreover, this K-Means clustering algorithm was improved with artificial neural network, competitive learning algorithm, in order to cluster the on-line game data-set in high dimensions.

2.4.2 Competitive learning

Competitive learning is unsupervised learning approaches in which the output neurons contend against each other to react as the one firing from the given pattern in inputs. Therefore only one unit take the winner upon all output units (2.1). This exclusive phenomenon in synaptic neurons is called 'winner-take-all units', and sometimes they are also called 'grandmother cells'. Which means a neuron fires on a specific stimulus. This neural networks has an advantage of clustering unlabelled data and searching the correlations of the input data. One of the most significant application of competitive learning might be vector quantisation. In which, input data-set becomes a bunch of

vectors in data space, and input vectors are compressed and stored by transmitting into prototype vectors.

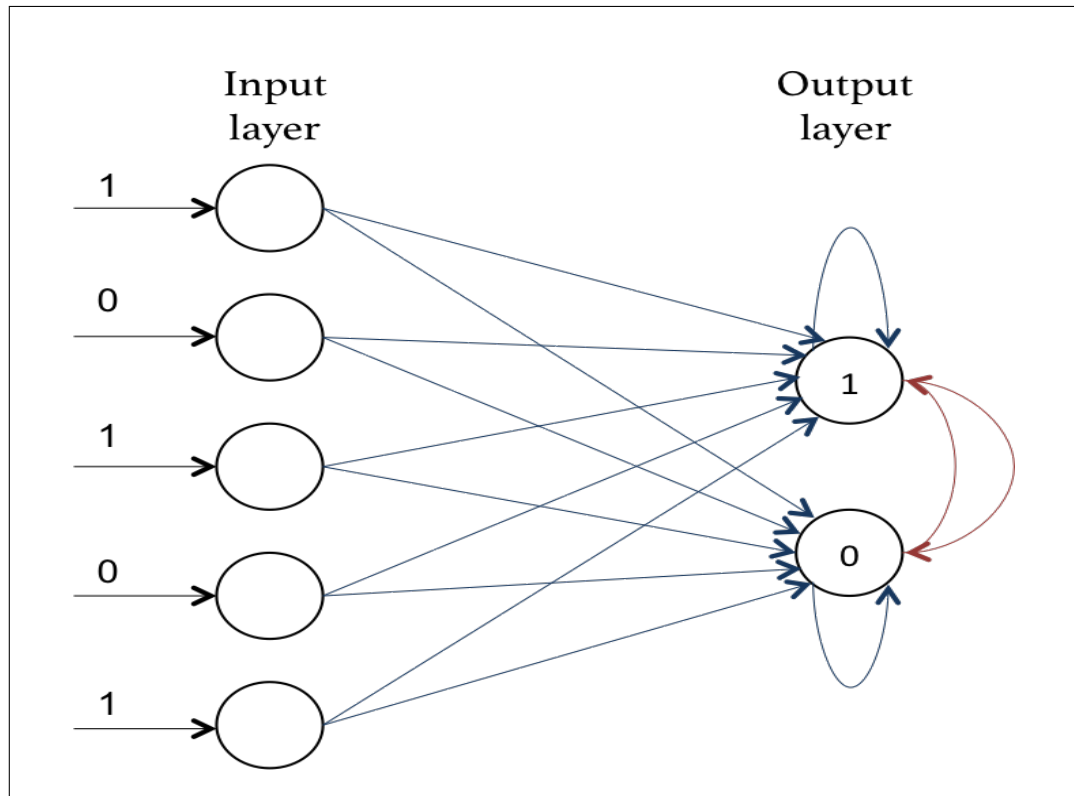


Figure 2.1: Competitive neuron diagram. When an input enters the network, it generates output values according to weights between an input and output neurons (blue arrows). Output neurons compete with each other (red arrows), then only one output fires on an input pattern. Output neurons are considered only having binary 0/1 output.

It has a very similar concept to K-Means clustering algorithm. The prototype vectors can be interpreted as centroids in k-means. Besides, vector quantisation employs Euclidean metric to assign an input vector to the nearest prototype, weight vector. When input data, usually represented as a vector, are applied at the input of the network, the winner neuron indicates the appropriate class.

If all input data are entered to the network, the input space organises a Voronoi tessellation. To be specific, Voronoi tessellation which was used for the analysis is Centroid Voronoi tessellation, in which means are positioned in each Voronoi cell (centre of mass). The weight vectors (or centroid in K-Means) are the vectors of prototypes.

Output neurons O_i in (2.4) are the units having the net input in an pattern at a time,

$$O_i = \sum_j w_{ij}x_j = \vec{w}_i \cdot \vec{x}, \quad (2.4)$$

for an input vector \mathbf{x} .

Winner neurons are found by (2.5),

$$\vec{w}_k^\top \vec{x}^\mu > \vec{w}_i^\top \vec{x}^\mu \quad \text{for all } i, \quad (2.5)$$

where winner neuron is \vec{w}_k^\top , and \vec{x}^μ is an input of pattern μ .

(2.5) tells winning unit is the biggest. If the weight vectors connecting between each input and output are normalized, $\vec{w}_i = 1$, then (2.4) can be redefined as below,

$$|\vec{x}^\mu - \vec{w}_k^\top| \leq |\vec{x}^\mu - \vec{w}_i^\top|, \quad \text{only if } |\vec{w}_i| = 1 \text{ for all } i. \quad (2.6)$$

(2.6) shows similarity to the key idea of K-Means clustering, which only consider distances between all data and centroids.

Competitive learning only consider the output neuron with the maximum net value of inner product between input and weight. Biologically, output neurons are connected with other output neurons with lateral inhibition, as well as excitatory connection itself, and therefore winner takes all units.

For competitive learning, the thing to note is that weights must be normalised in order for all units to have a unit length.

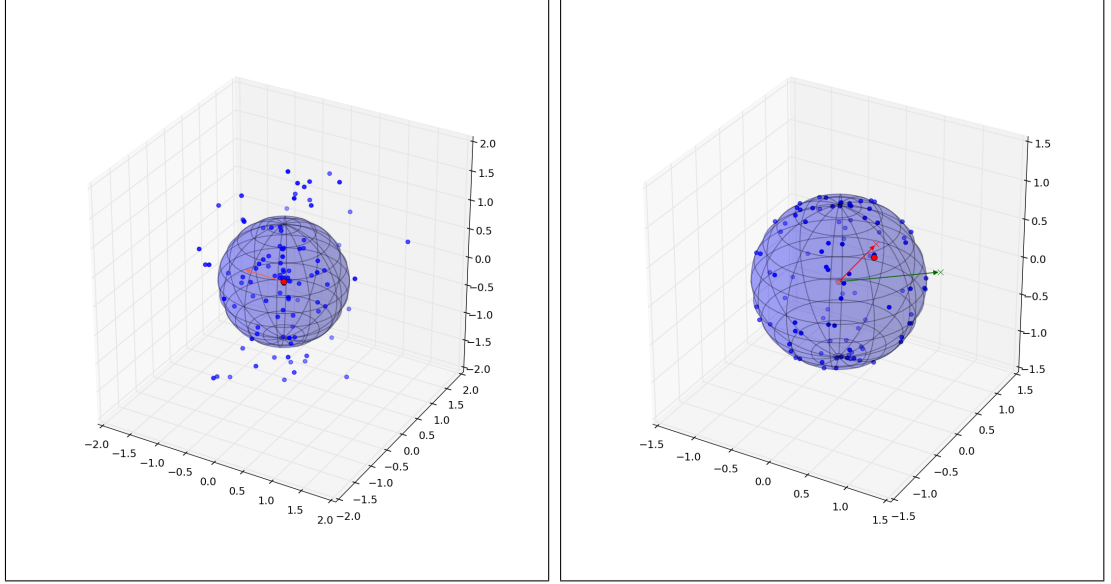
2.4.3 Normalisation

Normalisation is important for analysing data with different scale to be compared and then to derive information in K-Means clustering (Bowles, 2015). To be specific about normalisation for competitive learning, the network exploits the inner product for each input with weight vectors in order to identify winner output unit. If looking inside of (2.5), the equation for inner product is

$$\vec{w}^\top \cdot \vec{x}^\mu = \vec{w}^\top \vec{x}^\mu \cos(\theta), \quad (2.7)$$

with respect to the lengths of both units and a direction between input pattern and weight. Not-normalised one has no way for a criterion for competing output units with

weight vectors. Because the neuron which has the largest value in length always wins. That implies the algorithm will finish without any learning or no informative outcome through iterations, in the both cases of not-normalised data-set and weight vectors.



(a) Not-normalised data with a unit weight (red arrow) (b) Normalised data-set with not-normalised weight vectors (green arrow).

Figure 2.2: Examples of not-normalised data and weight in 3 dimensions. In both left and right figures, sphere represents unit length area.

For the solution of that, normalising input and weight vectors simply transforms (2.7) into the equation computing only a direction between those vectors, which tells how close weight vector is from the input vector.

Eventually, the inner product in (2.6) becomes the criterion which detects the shortest Euclidean distance by comparing angles between input and weight vectors, (2.8).

$$\begin{aligned} \vec{\mathbf{w}}_k^\top \cdot \vec{\mathbf{x}}^\mu &> \vec{\mathbf{w}}_i^\top \cdot \vec{\mathbf{x}}^\mu \\ \cos(\theta_k) &\geq \cos(\theta_i). \end{aligned} \tag{2.8}$$

The firing output neuron still has the largest value among output units when the θ is smaller than others based on the competition with proximity of the input vector.

(2.2) shows not-normalised issue in competitive learning. In the not-normalised data-set and weight vector, (2.7) always compares only the length, not the closest one

in the distance between data and weight vector. It is obvious that the most long vector always win, which cannot be representative for input patters. Thus weights would not move to the centre of Voronoi cell.

2.4.4 Procedure of the algorithm

After normalising data-set and setting weight vectors with unit length, the competitive learning algorithm for pattern classifier exploiting (2.4) and (2.8) follows as below,

1. choose weight vector with random values:

Similar to K-Means clustering, weight vectors which implies center of mass for clusters (or Voronoi cells) are initialised with random values. However, the issue of dead unit would be caused by randomly initialised weight vectors. Dead unit is the unit which is very far from any input patterns, and therefore never fires. An other problem is the optimal number of weight vectors. This problem will be discussed in the next section. The last thing to be noted is that weight vector must have unit length.

2. apply input patterns to the network in turn or in random order:

After setting initial weights, input data vector as a pattern is entered to the network. It does not matter applying it in turn or in random order. The advantage of competitive learning over K-Means is that K-Means comparing all data, then must be slow, but competitive learning can run in random order input data until it approaches the sufficient learning rate. Therefore it is much faster than K-Means, on condition of how learning rate is prescribed.

3. discover a winner output unit for each input:

When an input vector is applied to the network, which output unit is likely to take winner position for each normalised input pattern is found through (2.8). For a given input pattern, the winner unit has the largest value among other output neurons. In other words, winner-take-all unit is the closest vector to the input vector with the smallest angle. Inhibitory and self-exitatory connection are defined biologically, but alternatively using python command, **numpy.argmax**, just maximum value is searched.

4. Update the winner neuron's weight to be close to the input vector:

For each input, winner \mathbf{w}_k is found among the outputs, and then it is updated to be closer by an allowed amount with learning factor η to the current input vector. After continued update of the weight for the winner unit, weight vectors will be positioned at the center of Vornoi cells which shows similar patterns of inputs. Below is on-line and batch update rule,

$$\begin{aligned} \text{on-line rule : } \Delta \vec{w}_k &= \eta (\vec{x}^\mu - \vec{w}_k) \\ \text{batch rule : } \Delta \vec{w}_k &= \eta \sum_{\mu \in C_k} (\vec{w}_k - \vec{x}^\mu) \end{aligned} \tag{2.9}$$

On-line learning rule updates the change of one data point at a time. However, batch learning rule accumulates weights until all inputs in a cluster are passed to the network, and sums all of those constantly. Both converge the same centroids of clusters, but batch rule is slower than on-line learning rule which slightly fluctuates towards local optima when updating.

As mentioned in procedure 1, choose weight vector with random values, the two possible problems; the number of clusters and dead units, of competitive learning have to be handled with proper method. The next section will introduce breakthroughs which can tackle those two issues.

2.4.5 Local optima and optimal number of clusters

Sometimes, a number of mean values might be randomly initialised very nearly to each other. And centroids would converge a same point in a same cluster. In this case, clusters would not describe data-set informatively. This issue is called local optima problem. Unfortunately, there is "no way of knowing what the right answer is" for the outcome of clustering algorithm, but "repeating the algorithm dozens or even more times" can increase reliability of clustering result (Hackeling, 2014). By repeating clustering algorithm several times, the starting random data points will be set differently and converge different or the same local optima as subsequent outcomes. Clustering results are compared relatively with repeated results, and the most frequent similar results can be presumed as reliable result.

Another crux of K-Means clustering is that the optimal number of clusters is unknown. There 854,064 players in the on-line game data-set, and it has no predefined

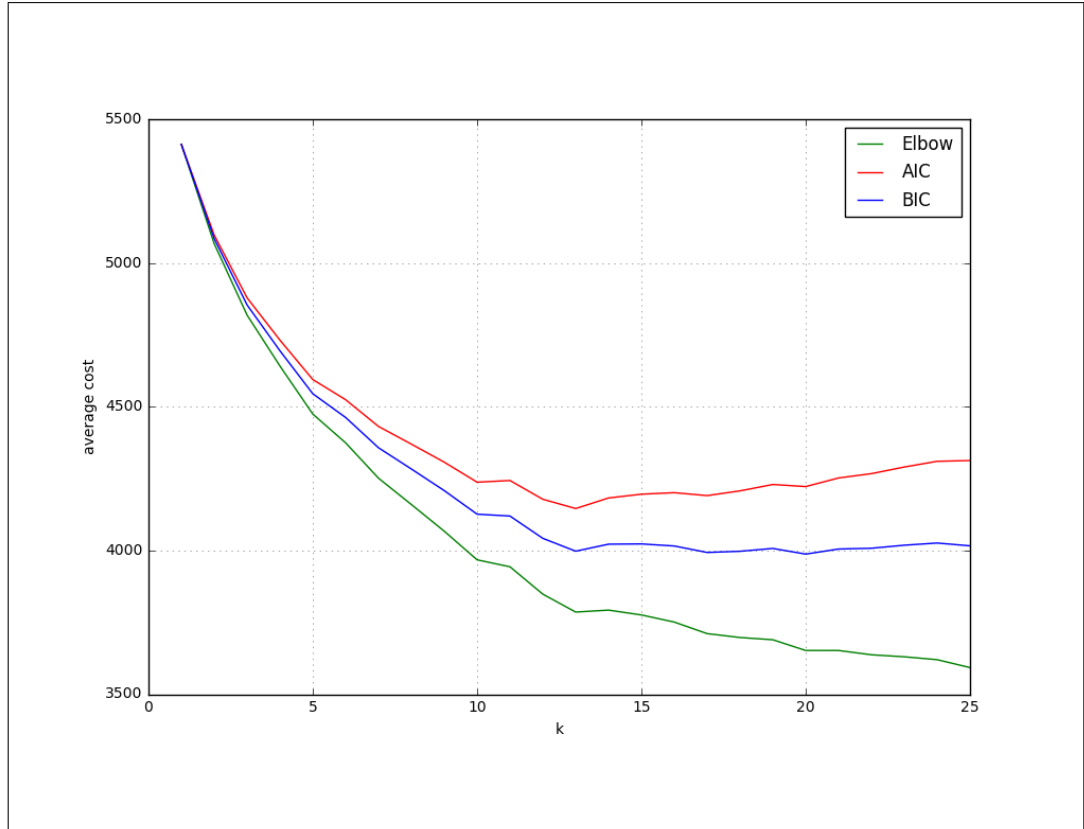


Figure 2.3: Comparison between AIC and BIC according to the different number of k . AIC are more strict method than BIC to identify the number of clusters.

information of how many groups showing what the similar learning patterns are. To estimate optimal number of clusters, Elbow method with Bayes and Akaike Information Criteria (2.10) was used. The criteria which can determine the number of cluster is as follow,

$$\text{Bayes Information Criteria (BIC)} : \arg \min_K (J_K + K \log D) \quad (2.10)$$

$$\text{Akaike Information Criteria (AIC)} : \arg \min_K (J_K + 2KD).$$

The more a number of clusters, the less the value of the cost function, J_K . Because, increasing K can divide the data-set into smaller pieces of clusters than the actual number of how many clusters necessarily exists in it. Usually, using the Elbow method shows changes of the value of cost function. It goes down significantly at first some K , and suddenly slows down the change. By inspecting the point showing from dramatic to gradual change, generally the optimal number of K is detected.

However, when only Elbow method for the on-line data-set is plotted, this inflection

point is rarely found. Because all costs seem decreasing gradually, Fig. 2.3. Therefore, BIC and AIC was exploited with Elbow method. If plotting AIC values in different number of K , it will decrease by the certain amount cost, and then start to increase. It is because $2KD$ or $K \log D$ proportionally compensates decreasing the value of cost function for various number of K . In high dimensions, AIC criteria gives "high penalty for many clusters" (Daumé III, 2012). As a result, K value which minimizes AIC or BIC (2.10) could be considered as the optimal number of clusters.

Fig. 2.3 shows the information of elbow method with AIC and BIC to define the number of clusters for the on-line game data-set. By these three graphs, it seems that the data-set can be clustered into n clusters. If the number of 14 is set as a threshold for these differences, increasing K beyond 14 would not improve the result of clustering.

2.4.6 Dealing with dead unit

A common problem in competitive learning algorithm is dead unit issue. This is the case when weight vectors are initialised far distant from all input vectors. As they cannot fire among those which are closer to input vectors. Geometrically speaking, (2.4) would show precise understanding. Two arrows with different colours represent randomly initialised weight vectors. Red arrow is closer to two input groups than green weight vector. When they are threw into the network, red weight vector will move closer and closer to the centre between two groups at every iterations. Which means red weight vector always takes winner, and green weight vector will never be updated.

For example of test data set for 7 clusters in (2.5), neuron 6 was never fired and it is the dead unit, even though neuron 1 and 2 won over 1600 times.

There are several approaches which can tackle the dead units problem, and 4 of those;

1. leaky learning (LL),
2. add bias (AB),
3. add noise (AN),
4. initialisation within input space (SV),

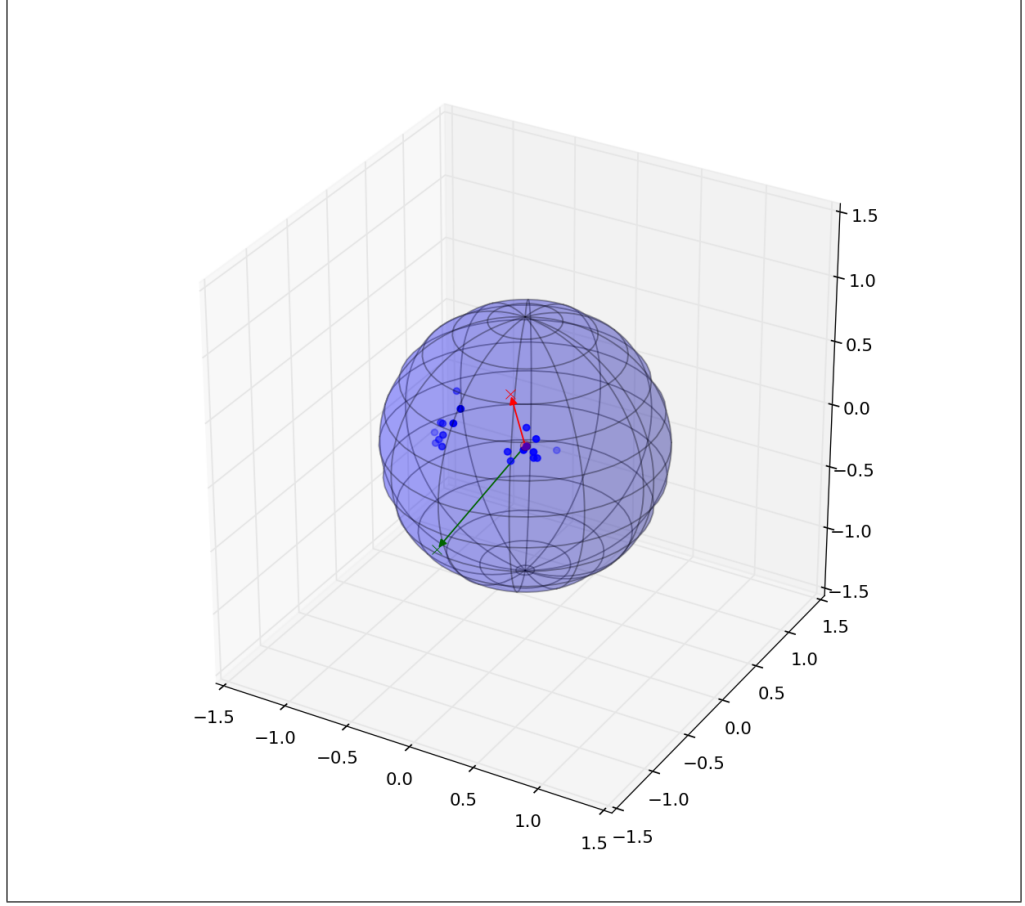


Figure 2.4: An example of dead unit problem. Red and Green arrows are initialised weight before applying into the network. Green arrow is set far from both two input clusters, and it is assumed as a dead unit. Unfortunately, green arrow will never fire through the learning, and which means that input vectors are not properly clustered.

will be discussed.

First, leaky learning (LL) is the way of adjusting learning rate for both winner and loser output units. It gives loser units some chances to fire in the future, and they move towards more dense clusters. Equation of LL is as follow,

$$\begin{aligned} w_{ij}^{new} &= w_{ij}^{old} + \eta_{winner}(x_i - w_{ij}^{old}), \\ w_{ij}^{new} &= w_{ij}^{old} + \eta_{loser}(x_i - w_{ij}^{old}), \quad \text{where } \eta_{winner} \gg \eta_{loser}. \end{aligned} \quad (2.11)$$

The second way to deal with dead units is adding bias (AB) to winner output units

which repeatedly fire. Then, they become hard to win again and again.

$$\begin{aligned}
f_j^{new} &= f_j^{old} + \beta(z_j - f_j^{old}), \\
b_j &= \gamma(1/n - f_j), \\
w_{ij}^{new} &= w_{ij}^{old} + \alpha(x_i - w_{ij}^{old})z_j, \\
y_j &= \sum_i w_{ij}x_i + b_j,
\end{aligned} \tag{2.12}$$

where z_j is a selector with a value of 0 for loser or 1 for winner, γ, β are constant factors, n is total nodes, f is firing rate of output neuron, and b is bias. By varying these parameters in (2.12), learning performance can be adjusted.

Next is to add noise (AN) to output neurons while they are being updated. For the on-line game data analysis, Gaussian distribution are added to all output neurons. Then, among the output units with Gaussian noises, the largest output neuron is taken as a winner unit. Because of randomness, it is possible that frequently winner neurons may not fire even in the situations when the winner and loser output units are obvious from the equation (2.8).

$$\begin{aligned}
y_j &= \sum_i w_{ij}x_i + \varepsilon, \\
\text{where } \varepsilon &\sim N(0, \sigma^2)
\end{aligned} \tag{2.13}$$

This is the simplest method to escape dead units problem, by just adding noises to outputs, one of loser units may take the place to win, instead of actual winner output neuron.

The last approach is to initialise weight vectors with randomly sampled vectors from input space. Three approaches which are discussed above do not solve the fundamental problem of dead units, and they still have the risk that some weight vectors are set far from any cluster. Because they randomly initialise weight vectors.

However, SV set initial weights in the actual input space. Therefore, they are always positioned inside the data set. Although they may manage dead units, there is other problem that some weight vectors are initialised near to each other. If some weight vectors are set nearly, they will converge the same optima, and data set will not properly clustered.

Thus, large (LSD) or small(SSD) values randomly sampled with Gaussian distribution are added to the initial weight vectors, so that they have enough distance from each other.

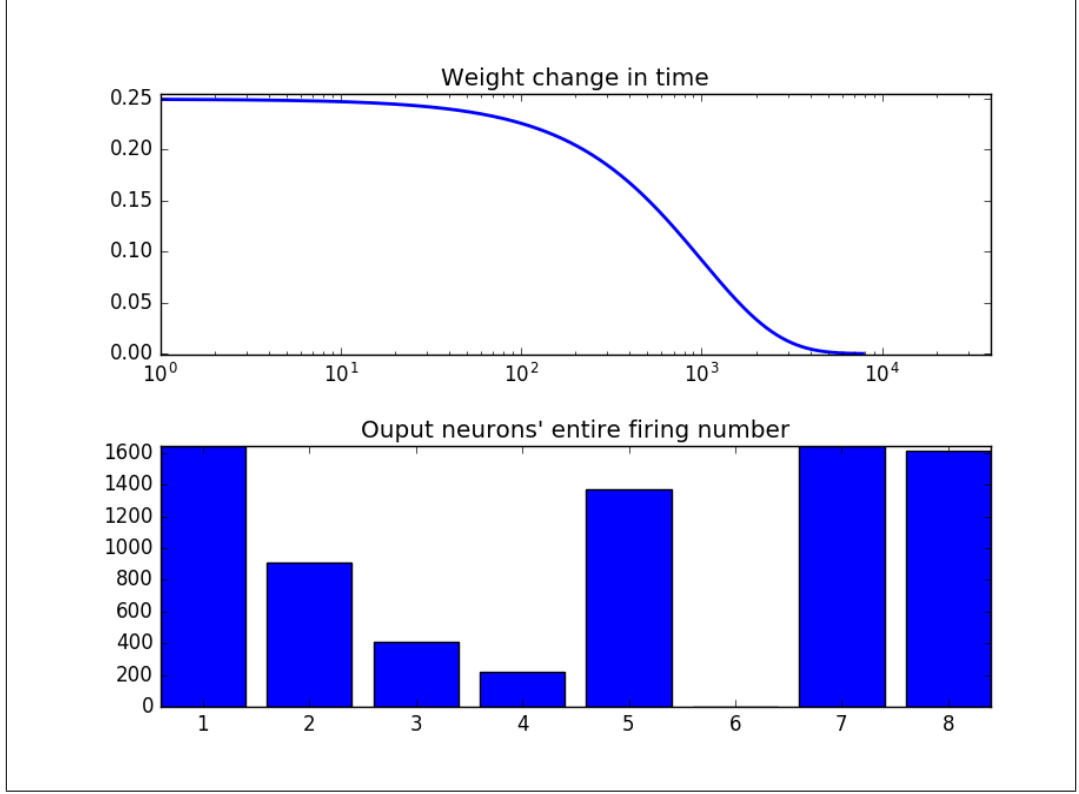


Figure 2.5: Learning rate and existence of dead units

$$w_{ij}^{init} = w_{ij}^{random} + \epsilon, \quad (2.14)$$

where $\epsilon \sim N(0, \sigma^2)$

In (2.6), it is shown that dead units are not exist. All output neurons fired above about 300 times, though some clusters are dominant in learning. This also give assumptions of how many individuals are belong to each cluster. It has to be noted that output firing state only give a rough estimate, as competitive algorithm exploit random point choice to compare with centroids. In addition, because there are no labelled and predefined features in the data set, there is no way to know how correct a result of clustering is. Thus, the data set is considered as unbiased, which means average from 10 iterations may represent the true value of the population.

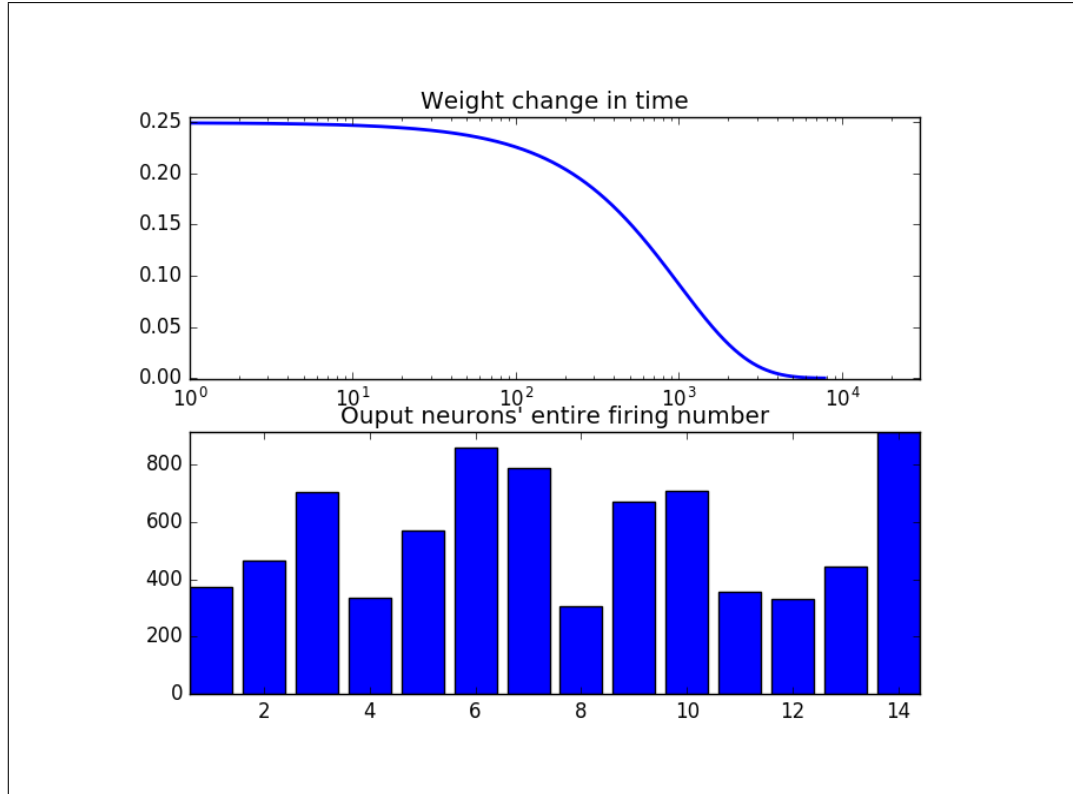


Figure 2.6: Learning rate and entire firing number of output neurons after using SV approach for dead units

2.4.7 Centroids

In conclusion of data curation, 1,201,516 individuals are grouped into 14 clusters as Fig.2.7, 2.8, 2.9. Note that these are the results of truncated 15 attempts.

In the next chapter, various curves are fitted to these centroids, and which curve may properly represent learning process for each group.

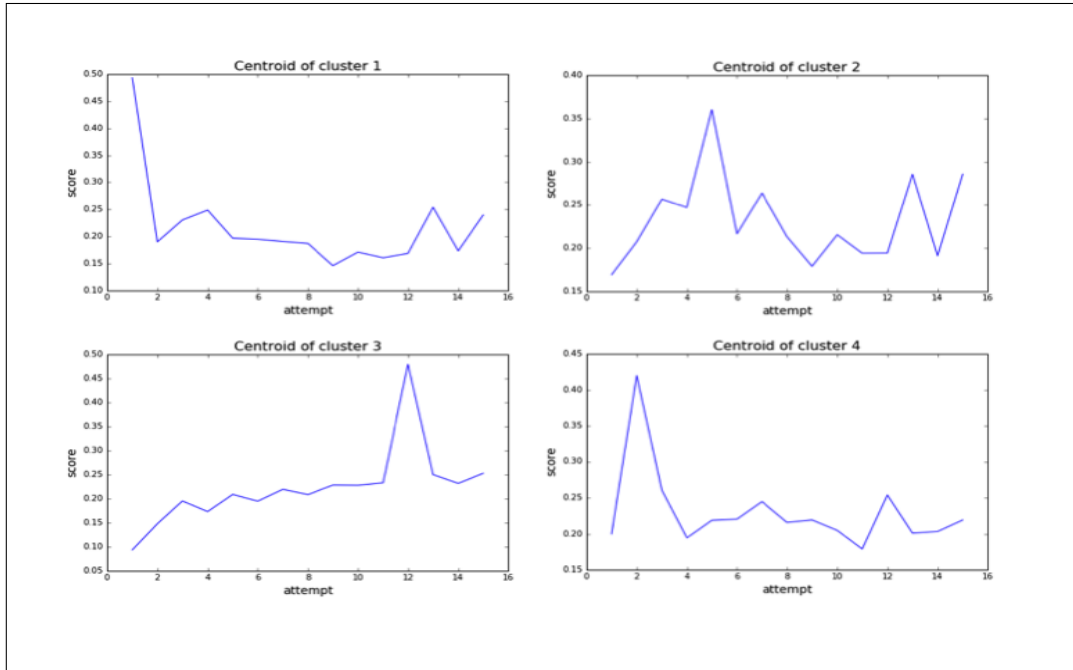


Figure 2.7: Centroids of cluster from 1 to 4

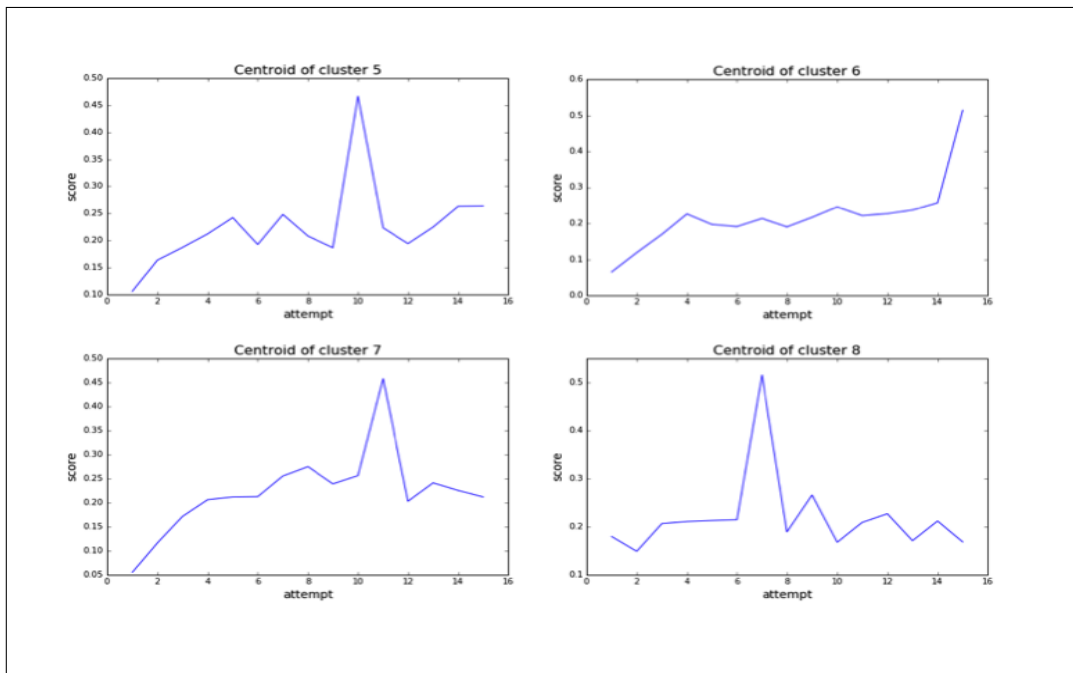


Figure 2.8: Centroids of cluster from 5 to 8

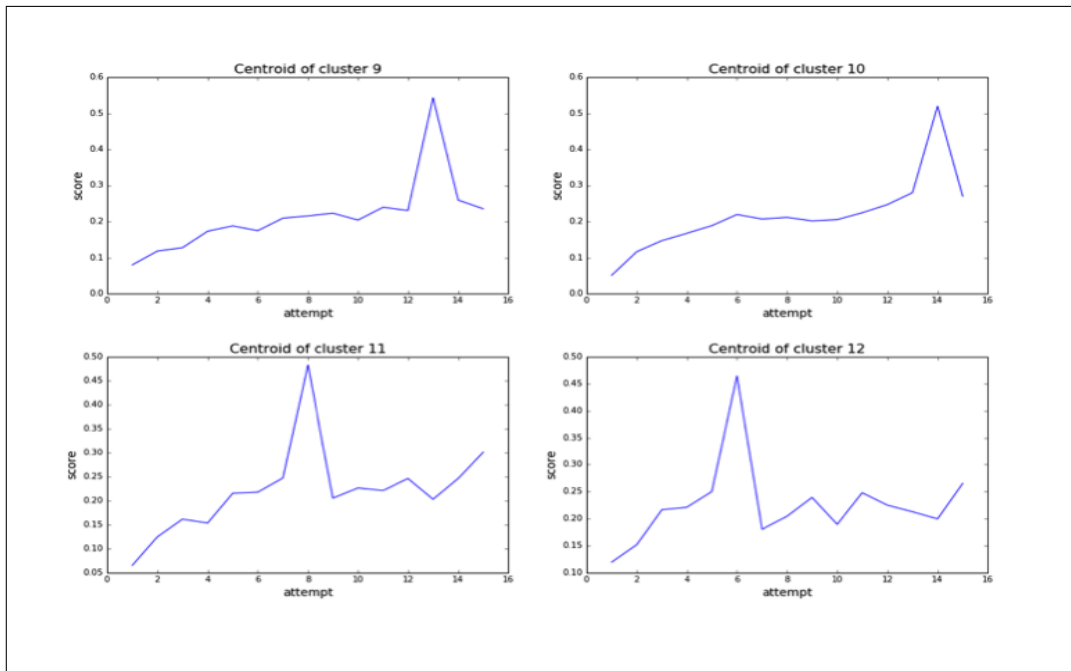


Figure 2.9: Centroids of cluster from 9 to 12

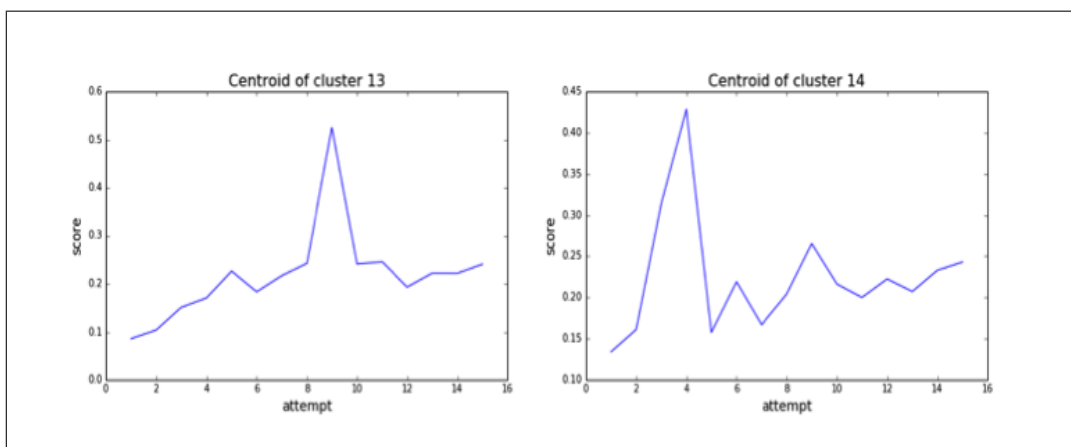


Figure 2.10: Centroids of cluster 13 and 14

Chapter 3

Function Fitting

This chapter examines the type of relationships (learning processes) between variables (attempts) and the corresponding responses (scores) players follow. The relationships can be modelled to some functions, and these may give a clue interrelating the amount of practises and performance of the on-line game data set.

This process can be called 'inference' in statistical learning. Inference is different from 'prediction'. To make sure of concise understanding, both methods estimate function f which may show the relationship between inputs and outputs. For an unbiased model, it can be described as

$$\hat{Y} = \hat{f}(X), \quad (3.1)$$

where average of errors goes to zero, and hat symbols represent estimates. However, prediction is used for the situations that predictors (inputs) are easily obtained but responses (outputs) are not readily available. Thus \hat{f} is treated as an estimator to predict currently unknown responses. On the other hand, inference has a different feature to "develop an accurate model" to "understand the relationship" between inputs and outputs (James, Witten, Hastie, & Tibshirani, 2013). In other words, it reveals a functional relationship between a set of independent variables and target variables. Thus, to find a best curve for the data set would be the task of inference.

Possible functions are introduced in the next session, and then the method of how to identify a best fitting model will be explained. Finally, models as the results of the inference task will be briefly discussed.

3.1 Functions

There are many functions which can map input variables to target responses. The easiest and most straightforward function in machine learning might be a linear model (Rogers & Girolami, 2015). The linear model has historically been taken for a model f , but can the method also be applied to the on-line game data set as a function of individual learning pattern? In general, the true relationship might be more complex, and a linear function may not inference an accurate model.

In the same manner, a well known function in Psychology is a power function (Howard, 2014). It has been controversial but considered as 'describing skill acquisition process well', whereby performance rapidly improves in early learning, decreases gradually, and reaches asymptote. Newell and Rosenbloom (1981) argued that "a single law, the power law of practice, adequately describes all of the practice data". Nevertheless, it is also argued that single exponential function may fit better for individuals than a power function, as it is produced by many averaged individual exponential functions (Murre & Chessa, 2011).

Others say that individual learning curve may show an abrupt or 'aha! moment'. The untrained subjects show often "step-like increase" (Gallistel et al., 2004), and the insight solutions are "correct more often" than "analytic solutions" for different types of problems; even in the analytic problems, (compound remote associates, anagrams, rebus puzzles, and visual aha problems). Thus, there is possibility that learning curve of the on-line game data set might show step-like appearance.

Therefore, it would be better to fit various possible functions and to compare each other. After that, a curve which shows the best result can be assumed the function representing performance of learning best.

3.1.1 Parametric Method

Fitting models to the data set are involved with finding appropriate parameters of the functions. Once parameters are determined, those are substituted to the model. Then the best fitting curve can be drawn on the input variables. This model-based approach in statistical inferring needs three steps of assumption.

Firstly, specific shape of model has to be predefined before a learning process. For example, the on-line game data might have relationship as polynomial function in the first order,

$$\begin{aligned} \text{first order model : } f(x_n; w_0, w_1) &= w_0 + w_1 x_n, \\ \text{second order model : } f(x_n; w_0, w_1, w_2) &= w_0 + w_1 x_n + w_2 x_n^2 \end{aligned} \quad (3.2)$$

where parameters are w_0, w_1 and w_2 , x_n represents data points, and $n \in \{1, 2, \dots, n\}$. Likewise, once the model is defined, a number of model parameters are automatically determined, and set as factors which have to be mined from statistical inferring process.

What has to be noted next is that linearity of the model. Most models in machine learning would be practically linear function such as a first order polynomial function above. If a model is linear itself or in the parameters, the predefined function such as equation (3.2) can be alternatively simplified into vector and matrix formation,

$$\begin{aligned} \text{vector : } f(x_n; \mathbf{w}) &= \mathbf{w}^\top \mathbf{x}_n, \\ \text{matrix : } f(\mathbf{x}; \mathbf{w}) &= \mathbf{X}\mathbf{w}, \end{aligned} \quad (3.3)$$

where \mathbf{w} is a vector containing parameters, and \mathbf{X} is a matrix consisted with data vector. For instance, a second order polynomial model has a data vector as $\mathbf{x}_n = [1, x_n, x_n^2]^\top$, and in matrix form,

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_N^\top \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & & \\ 1 & x_N & x_N^2 \end{bmatrix}.$$

Checking the linearity of the model is important, because function fitting process can be particularly simpler on linear functions than non-linear functions. In linear function, just taking partial derivatives of the loss function with respect to each parameter will result in the optimum values of parameters. The loss function will be discussed in session 3.2.1, squared lost function.

Finally, parametric methods need training data to fit the model. Having collected and analysed on on-line game data, there are curated 300 data sets for each individuals, as well as centroids of 13 clusters. Theses 300 data also can be separated into two parts;

training set and validation set. Training data are used to fit the model in the same way as before, and validation data are for testing the fitted curve, so as to check they are well representative of the data set. For more specific information of training and validation set, these will be explained later in the session 3.2.2.

3.1.2 Functions for Single Curve

In total 4 functions which are briefly explained above are examined as possible candidates for curve fitting,

1. polynomial functions in different orders
2. exponential functions with different parameters,
3. power functions with various parameters,
4. step-like function.

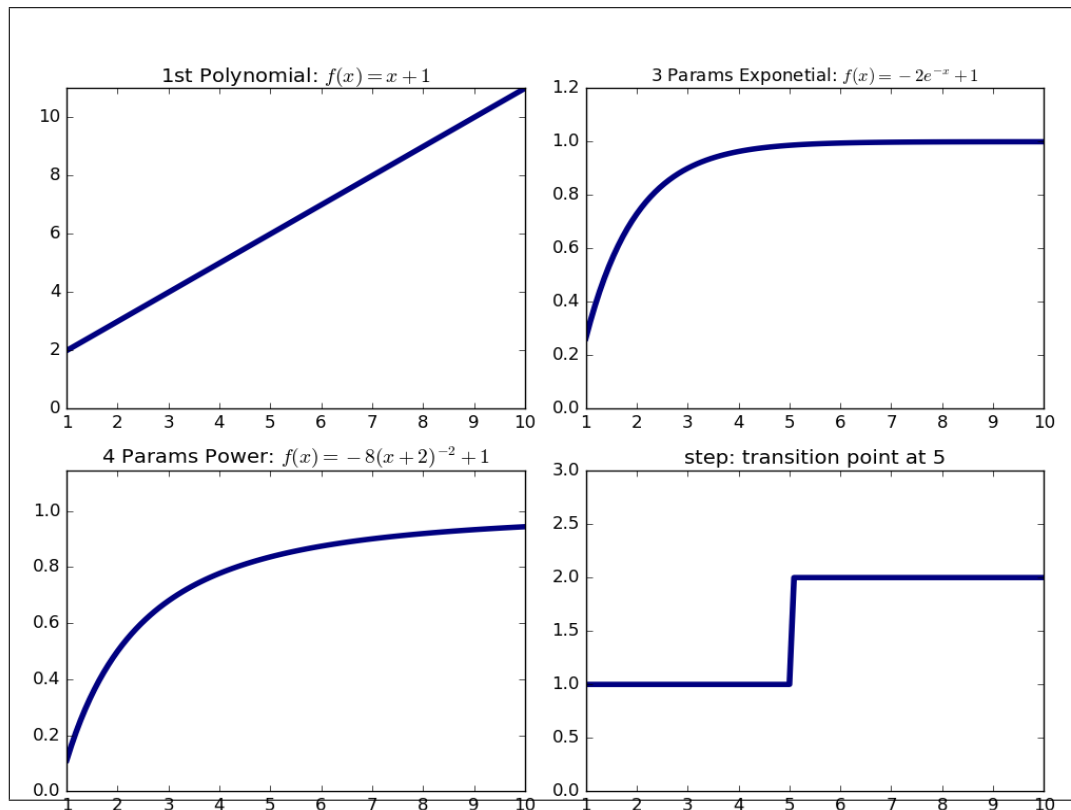


Figure 3.1: Possible 4 models; first order polynomial, 3 parameters exponential function, 4 parameters power function, step-like function with change point at 5

Polynomial functions from first up to five orders are trained for on-line game learning curve. First and second order were introduced in equation (3.2). Other remaining functions are as below,

$$\begin{aligned}
\text{3rd order model : } f(x_n; \mathbf{w}) &= w_0 + w_1x_n + w_2x_n^2 + w_3x_n^3 \\
\text{4th order model : } f(x_n; \mathbf{w}) &= w_0 + w_1x_n + w_2x_n^2 + w_3x_n^3 + w_4x_n^4 \\
\text{5th order model : } f(x_n; \mathbf{w}) &= w_0 + w_1x_n + w_2x_n^2 + w_3x_n^3 + w_4x_n^4 + w_5x_n^5.
\end{aligned} \tag{3.4}$$

Although high order polynomial functions are non-linear on function itself, they are still linear in parameters. In polynomial function, the increasing number of order, the lower cost they return. This situation is known as 'over fitting' problem in machine learning. Over fitting problem is common and must be solved in function fitting. This will be discussed in session 3.2.3, overfitting.

Next function is exponential functions which are often considered describing individual learning curves well. Function fitting for exponential function is examined from one to four parameters.

$$\begin{aligned}
\text{one parameter : } f(x_n; w_0) &= w_0 + e^{x_n}, \\
\text{two parameters : } f(x_n; \mathbf{w}) &= w_0 + w_1e^{x_n}, \\
\text{three parameters : } f(x_n; \mathbf{w}) &= w_0 + w_1e^{w_2x_n}, \\
\text{four parameters : } f(x_n; \mathbf{w}) &= w_0 + w_1e^{w_2(x_n+w_3)},
\end{aligned} \tag{3.5}$$

where w_0 is asymptote, w_1 is slope, w_2 is power, and w_3 is delay of the exponential function. It might be thought that exponential function with four parameters will fit best among various number of parameters and training model with parameters less than four will be time consuming, because the more parameters there are, the more complexity the function can have. However, Heathcote et al. showed that exponential functions with three parameters (asymptote, slope and power) described learning result better than the three parameter power function (Heathcote et al., 2000). Therefore, in order to find the best curve fitting best to the on-line game data set, the same model with different number of parameters has to be trained and compared. Interestingly, the exponential function fitting may result in step-like shape within the range of input variables, when inputs or w_2 goes to $-\infty$. This case will be discussed later.

The model of power law is also tested, varying the number of parameters.

$$\begin{aligned}
\text{two parameters : } f(x_n; \mathbf{w}) &= w_0 + x^{w_1}, \\
\text{three parameters : } f(x_n; \mathbf{w}) &= w_0 + w_2 x^{w_1}, \\
\text{four parameters : } f(x_n; \mathbf{w}) &= w_0 + w_2 (x + w_3)^{w_1},
\end{aligned} \tag{3.6}$$

where w_0 is asymptote, w_1 is power, w_2 is slope, and w_3 is delay. Power function with one parameter is omitted. Because power function with only w_0 parameter is first order polynomial function $f(x) = w_0 + x$. Likewise exponential function, if w_1 goes *infy*, it will show step-like shape, but if w_1 converges to 1, the power function will become first order polynomial function.

Finally, players might show step-like learning curve,

$$f(x_n; \mathbf{w}) = \begin{cases} w_0 & \text{if } x_n \geq \text{break point,} \\ w_1 & \text{if } x_n < \text{break point.} \end{cases} \tag{3.7}$$

While above polynomial, exponential and power functions are continuous over a range of input variables, step-like function shows 'discrete' appearance which has a separate unconnected 'break point'. Thus, step-like function shows an abrupt change between a left side and right side at the break point. Step-like function is largely related to the feature of piecewise. The other other three functions are also trained on several pieces of train set.

3.1.3 Piecewise Function

Piecewise function may be continuous within pieces, but the continuity will break at the point between pieces. Pieces defines at least two equations on different parts of the input domain. Individual learning curves performed on the on-line game may contain this feature, as Gallistel (2004) argued that smooth power function (or Weibull function) cannot provide reasonable evidences of individual learning curves, and discontinuity occurs at change-points. Therefore, individual learning curves of the on-line game might be explained better through piecewise defined functions than the single curve functions.

For the analysis, the above four functions are combined themselves. For instance, multiple power functions constitutes a piecewise power function. In the same manner,

piecewise exponential and step functions can be extended to multiple exponential and step functions respectively. Below is an example of piecewise power function having 3 parameters with 1 transition point,

$$f(x_n; \mathbf{w}) = \begin{cases} w_0 + w_2 x_n^{w_1} & \text{if } x_n \leq \text{transition point,} \\ w_3 + w_5 x_n^{w_4} & \text{if } x_n > \text{transition point.} \end{cases} \quad (3.8)$$

Similarly, all different functions can be combined together, such as the power function which comprises earlier part of learning and step-like or polynomial functions following transition point,

$$f(x_n; \mathbf{w}) = \begin{cases} w_0 + w_2 x_n^{w_1} & \text{if } x_n \leq \text{transition point,} \\ w_3 + 2_4 x_n & \text{if } x_n > \text{transition point.} \end{cases} \quad (3.9)$$

Every player may show a large variety of learning patterns, but exact learning curves are ambiguous. Thus, functions are fitted as many and various as possible, and compared with each other under a criterion, squared lost function. This criterion will be discussed in session 3.2.1

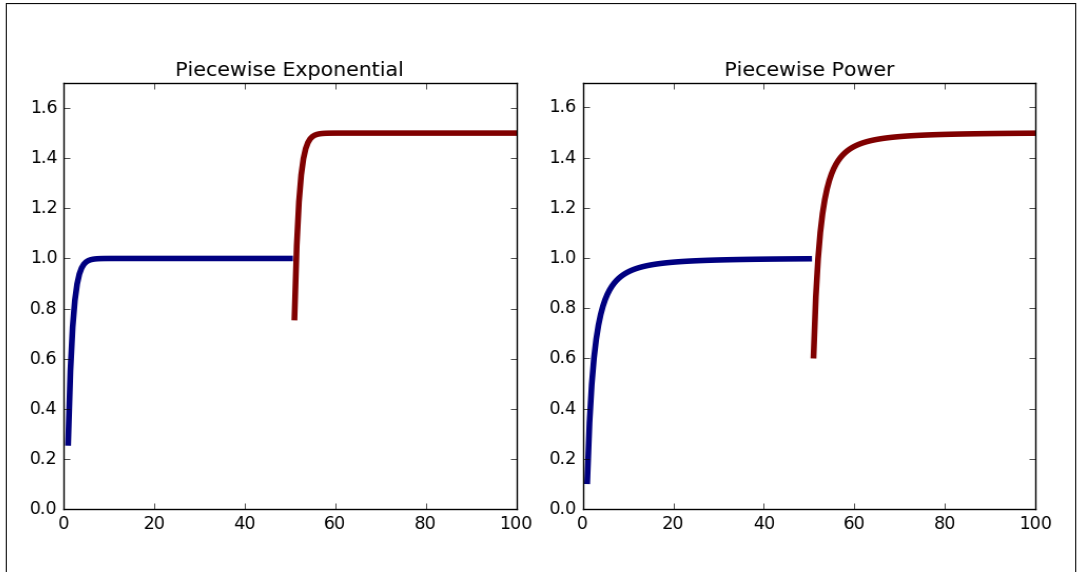


Figure 3.2: Example of piecewise functions; exponential and power functions

Another clue about the best curve fitting in piecewise functions is identifying such transition points (or change-points) in the data set (Gallistel et al., 2004). To identify the true discontinuities, Donner and Hardy (2015) suggested two conditions. First, a large data set is required with many individual learning curves. At the same time,

those learning curves should have long enough data sequence including "potentially infrequent discontinuities". Fortunately, the on-line game data set complies with these conditions. 854,064 individuals comprise the train set and almost all players have discontinuous data sequence because of unknown reasons.

In addition, in this on-line game data analysis, transition points would not only represent discontinuities of data sequences but also provide the position where squared errors in a piece wise function are minimum. The minimum cost can tell which points will result in the best curve fitting on the data set. How to find these transition points will be discussed in session 3.2.4.

To sum up, individual learning curve may be represented as a single curve or multiple curves. Four functions; polynomial, exponential, power, and step-like function, are examined for the clustered on-line game data set. How to fit those possible models to the data set will be dealt with in the next session.

3.2 Method

It was discussed throughout the previous chapter what kind of models might be contained in the on-line data set as the individual learning curves. However, the on-line data set seems to be containing many noises, and they make it hard to identify true parameters of the models with such high disturbance. Furthermore, even after feasible parameters are found, criteria which measure how the trained learning curves fit well on the predefined models are necessary. If not, there is no way to distinguish the best individual learning curve among various outcomes.

Therefore, some appropriate methods have to be operated through the curve fitting process. This session introduces the curve fitting method and relevant criteria.

3.2.1 Squared Lost Function

There are several optimality criteria to search for optimal parameters. Squared lost function is one of the possible and well known methods to fit the linear model (James et al., 2013) which is linear itself or in parameters. The best learning curve may consist of the parameters producing a line that passes as many and as close as possible to the data points.

The squared lost function is defined as below,

$$\begin{aligned}\mathcal{L}_n(y_n, f(x_n; \mathbf{w})) &= (y_n - f(x_n; \mathbf{w}))^2, \\ \mathcal{L}_{\text{avg}}(\mathbf{y}, f(\mathbf{x}, \mathbf{w})) &= \frac{1}{N} \sum_{n=1}^N \mathcal{L}_n(y_n, f(x_n; \mathbf{w}))\end{aligned}\tag{3.10}$$

where \mathcal{L}_n represents cost value of the difference between true value y_n and trained value $f(x_n)$, and \mathcal{L}_{avg} is average cost of errors loss. If the value of cost is smaller and smaller, the model with trained parameter becomes closer to the true variables at the input position n . In other words, the average errors loss will decrease when the trained function increases with accurate parameters to the true values.

The arg min of the average loss is involved in finding such geometrical turning points with respect to parameters, and it will tell the minima corresponding to the turning points,

$$\mathbf{w}_{\text{opt}} = \arg \min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^N (y_n - f(x_n; \mathbf{w}))^2.\tag{3.11}$$

One way to identify those turning points with respect to parameters is using first and second partial derivatives, using vector and matrix formation. Substituting (3.3) to the loss function will lead to (3.10),

$$\begin{aligned}\frac{\partial \mathcal{L}_{\text{avg}}}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}} \left[\frac{1}{N} \sum_{n=1}^N \mathcal{L}_n(y_n - \mathbf{w}^\top \mathbf{x}_n)^2 \right] \\ &= \frac{\partial}{\partial \mathbf{w}} \left[\frac{1}{N} (\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) \right] \\ &= \frac{\partial}{\partial \mathbf{w}} \left[\frac{1}{N} \mathbf{w}^\top \mathbf{X}^\top \mathbf{X} \mathbf{w} - \frac{2}{N} \mathbf{w}^\top \mathbf{X}^\top \mathbf{y} + \frac{1}{N} \mathbf{y}^\top \frac{1}{N} \mathbf{y} \right]\end{aligned}\tag{3.12}$$

First derivatives of the model (or gradient of the function) must be zero with respect to the turning points in a graph in parameters. Thus (3.12) results in the solution,

$$\begin{aligned}\frac{\partial \mathcal{L}_{\text{avg}}}{\partial \mathbf{w}} &= \frac{2}{N} \mathbf{X}^\top \mathbf{X} \mathbf{w} - \frac{2}{N} \mathbf{X}^\top \mathbf{y} = 0 \\ \mathbf{X}^\top \mathbf{X} \mathbf{w} &= \mathbf{X}^\top \mathbf{y}\end{aligned}\tag{3.13}$$

Then, (3.13) is solved with respect to \mathbf{w} , using the Python command *numpy.linalg.solve*

or QR decomposition,

$$\begin{aligned}
\mathbf{X}^\top \mathbf{X} \mathbf{w} &= \mathbf{X}^\top \mathbf{y} \\
(\mathbf{QR})^\top (\mathbf{QR}) \mathbf{w} &= (\mathbf{QR})^\top \mathbf{y} \\
\mathbf{R}^\top (\mathbf{Q}^\top \mathbf{Q}) \mathbf{R} \mathbf{w} &= \mathbf{R}^\top \mathbf{Q}^\top \mathbf{y} \\
\mathbf{R}^\top \mathbf{R} \mathbf{w} &= \mathbf{R}^\top \mathbf{Q}^\top \mathbf{y} \\
\mathbf{R} \mathbf{w} &= \mathbf{Q}^\top \mathbf{y},
\end{aligned} \tag{3.14}$$

where \mathbf{Q} is orthogonal matrix, so that $\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}$, and matrix \mathbf{R} is upper triangular. QR decomposition may provide a solution with more numerical stability, because $\mathbf{X}^\top \mathbf{X}$ is removed, in which all elements are squared and then potentially numerical precision disappears. However, QR decomposition will operate on \mathbf{X} directly, preserving the model's numerical precision. Finally, second derivatives show that whether the turning points correspond to minimum, maximum or saddle point.

For non-linear function, it is not tractable to find the optimal parameters by equating partial derivatives to zero, and trans-positioning matrices in order to remain parameter matrix. In addition, to set the parameter and variable matrices is not impossible in non-linear functions with respect to parameters. In this case, 'gradient descent' method is used, which minimises the squared lost function.

Initialise p_n {there are n number of parameters}

Set η {learning rate prevents overshoot problem}

while $p_{n_new} - p_{n_old}$ is not greatly smaller **do**

for $i = 0$ to n **do**

$$p_{n_new} \leftarrow p_{n_old} + \eta \frac{\partial \mathcal{L}}{\partial p_n}$$

end for

end while

Through these two methods, it can be determined that the optimal parameters results in the best learning curve. For convenience of these methods, Python command `scipy.optimize.least_squares` is utilised, which "solve a nonlinear least-squares problem with bounds on the variables" (python on-line document, n.d.-b), and returns costs and other useful arguments.

Squared lost function is used not only to find the optimal parameters, but also for the criteria to compare the fitted learning curves. For this analysis, it is assumed that the best fitting learning curve among different models will have a minimum cost.

3.2.2 Training Set and Validation Set

Separating the entire data set into two parts; training set and validation set, is the method to deal with the over-fitting problem. Ideally, the best fitting learning curve can generalise predictions close to the true value (Rogers & Girolami, 2015). Therefore, separating the individual data-set with 300 attempts to two parts would have the same role of the generalisation. Although the model fits well on the training set and has very small cost, the generalised predictions from the model for an unseen data may have great disparity with big errors loss.

Therefore, to choose the best function among a set of models, each model is trained on the reduced training set, and then the corresponding costs are computed on the validation set.

Particularly, training and validation set is used for identifying the best fitting single curve to the data set among four functions; polynomial, exponential, power and step-like functions.

At first, assumption for learning processes will be tested on early in learning of 14 data set, in which it is assumed that individuals with similar performances early in learning may show similar learning patterns after 15 attempts. Next, the entire 300 attempts data set is reduced to the data set containing from 1 to 50 practice, which is supposed that the players practised enough.

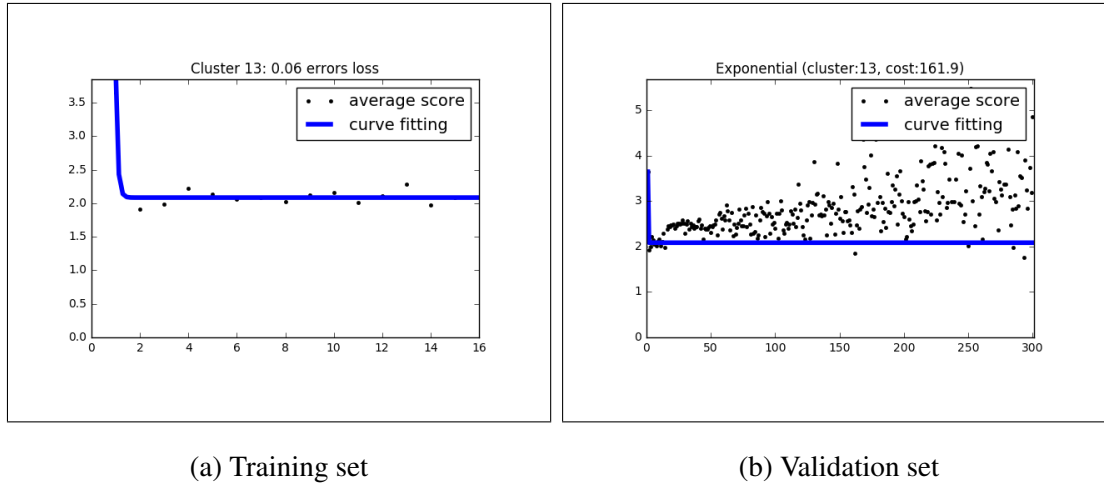


Figure 3.3: Minimum cost in training set, but large cost in validation set. Exponential function fitting with 3 parameters to 15 training set of cluster 13, and validation check

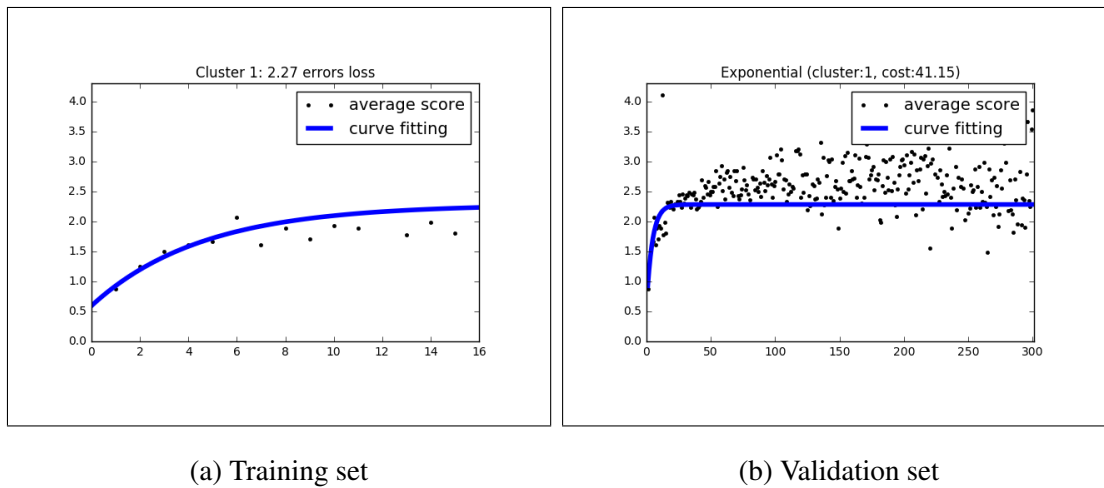


Figure 3.4: The learning curve which has the minimum cost in validation set. Exponential function fitting with 3 parameters to 15 training set of cluster 1, and validation check

3.2.3 Over-fitting and Regularisation

The term of over-fitting in machine learning is the case that "a model is too complex and is using its surplus complexity to fit to noise" (Rogers & Girolami, 2015). For example, in polynomial model, the higher order the function is, the lower errors loss the model will have. It makes sense that the best learning curve will have the least cost in different single functions. However, these criteria cannot afford the comparison method between a single function and piecewise model, because separating the data range as small as possible means fitting the data set into small intervals, and will lead definitely and undoubtedly to smaller cost (over-fitting), such as the situation that increasing the number of clusters leads to smaller mean square errors in session (2.4.5).

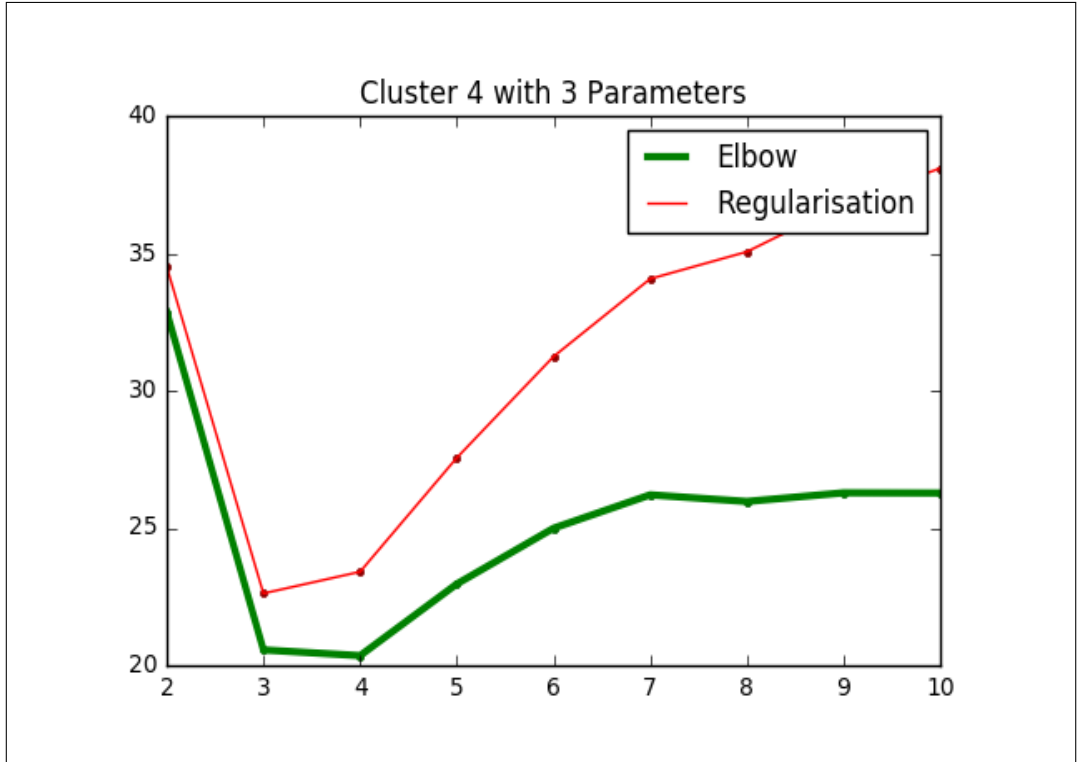


Figure 3.5: Regularisation example of cluster 4 in the piecewise power function with 3 parameters.

Regularisation prevents the model becoming too complex, by penalising over complexity (Rogers & Girolami, 2015). Likewise the solution of k-mean clustering with competitive learning algorithm, 'Bayes information criteria' (BIC) and 'Akaike Information Criteria' (AIC), equation (2.10), are used as the regularisation criteria.

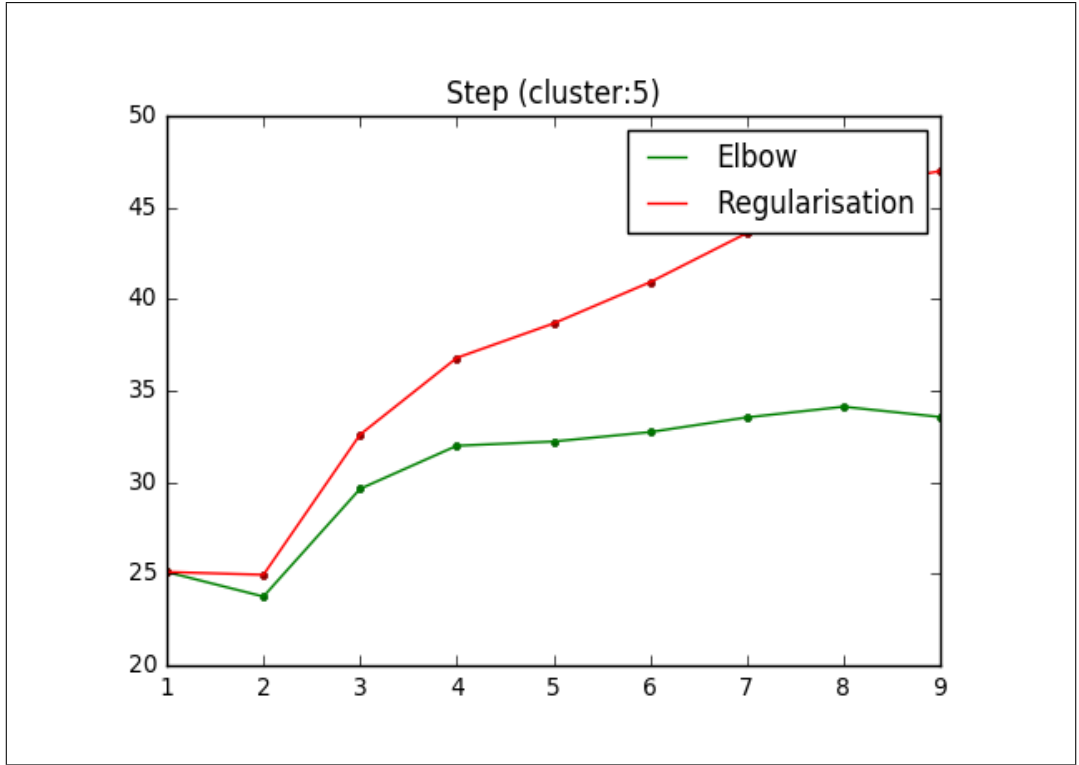


Figure 3.6: Regularisation example of cluster 5 in the piecewise step-like functions.

3.2.4 Choice of Breaking Point

Fig. 3.6 shows an example of how to find the exact transition points. Firstly, the entire data range is separated to $n - 1$ transition points in n pieces. Thus, there are three lines as the transition points with different colours (red, green, blue). A minimum of 50 window in each piece is used for the example, though actual window is 10. Accordingly, transition point 1 is on 50, point 2 is on 100, and the last point is on 150, and the last piece has remained 150 of attempt range as an initial condition. Fig. (3.6) shows this example of the transition point shifts.

In each of the pieces, a single function's cost is saved into a matrix. At the same time, length and transition points of each piece is also saved into another matrix. The shape for both matrices are $[\text{number of pieces} \times \text{number of shift of a last transition point}]$. (3.15) and (3.16) are examples of the cost matrix and length matrix for 3 transition points with 4 pieces.

$$\mathbf{M} = \begin{bmatrix} 1 & 1 & 1 \\ 51 & 101 & 151 \\ 101 & 151 & 201 \\ 151 & 201 & 251 \end{bmatrix}. \quad (3.15)$$

$$\mathbf{L} = \begin{bmatrix} 150 & 100 & 50 \\ 150 & 100 & 50 \\ 150 & 100 & 50 \\ 150 & 100 & 50 \end{bmatrix}. \quad (3.16)$$

Because a start point of the first piece does not change, all elements in the first row are 1. How much iteration is needed is calculated based on the size of the last row. In this case, the last piece has initially 150 range and minimum window 50, then a number of shift in each breaking-point becomes $150 \div 50 = 3$. Therefore, cost matrix has 4×3 shape.

Using those two matrices, the optimal points which allow the piecewise function have a minimum error cost that can be identified. Firstly, it is repeated that one element in each row is derived and summed. Then the total value from every iteration is checked whether is 300. 300 means the entire range of each individual data set. Then matrix L is mapped to the cost matrix, and tested that transition points are plausible. If a start point in a later row is smaller than earlier rows, the total cost value is discarded, and the costs in which each start point are greater than earlier rows from themselves are compared with each other. Finally, a minimum cost with corresponding starting points is found. Below is an pseudo-code of the algorithm to find starting point with a minimum cost.

Initialise l, c, s

for i in rows of length matrix **do**

for j in rows of index matrix **do**

$l \leftarrow \sum \text{row}_{i,j}$

if l is 300 and $\text{element}_{i+1,j} > \text{element}_{i,j}$ **then**

$c \leftarrow \sum \text{cost}_{i,j}$

end if

```

    end for
end for
 $s \leftarrow$  Map  $\arg \min c$  into starting point matrix
return  $s$  and  $\min c$ 

```

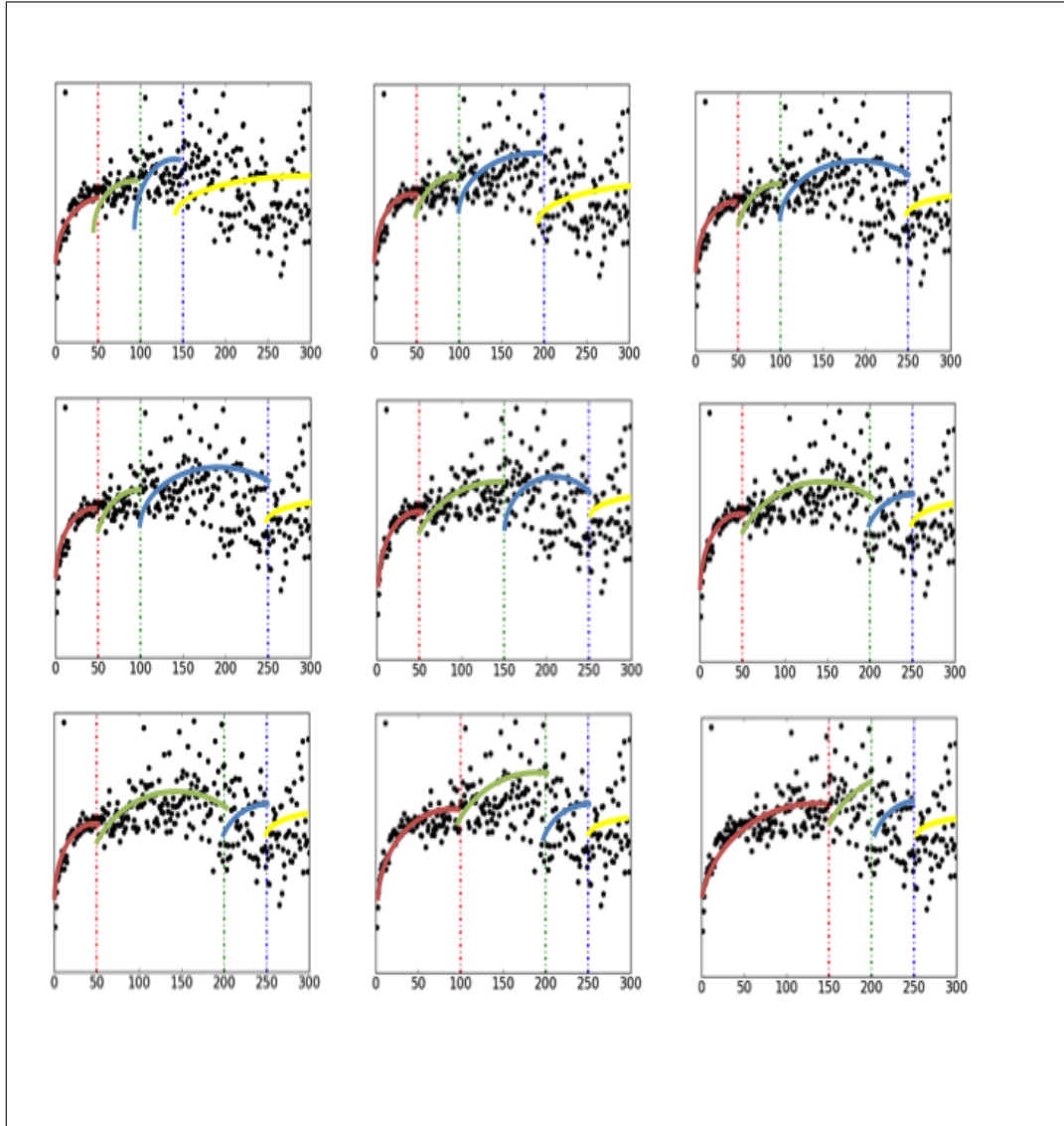


Figure 3.7: Transition point shift in 4 pieces

Chapter 4

Discussion

The traditional power law model (single smooth power function) is successfully fitted to the entirely averaged on-line game data set than the exponential law model. It became obvious that the power law model is the artificial learning curve comprised of several practice law models.

After the entire data set is categorised into the 13 groups through the data curation task, in which individuals show the similar learning patterns early in learning, each group provides different shape of learning curves. The most-frequent function is piece wise model, in which individuals follow the power law model with 4 parameters combined with step-like or first order polynomial models.

It seems that power law model is still dominant in those 13 clusters rather than exponential or other models. The reason of that may be that those groups are still averaged data set over 1000 individuals having similar early learning patterns. Because the raw data of the on-line game for almost all individuals have discontinuous data sequences; for example individual A has the data ranging from 40 to 90 practice, or the data of individual B starts attempt 1 with practice range 20 and the second interval starts 130 attempt with practice range 40, the function fitting for each individual could not be implemented, and thus players' data are clustered on the basis of the early 15 attempts, and are averaged instead.

4.1 The Best Learning Curve

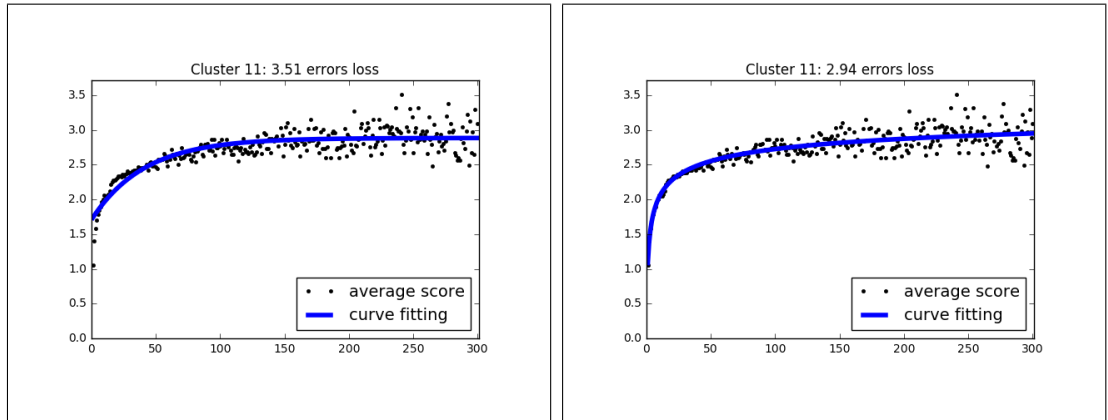
As discussed above, those 13 groups have different shape of learning curves. The most of them are piece wise function, while cluster 3 and 8 have a single first order polynomial function and power function with 4 parameters, respectively.

Firstly, the argument insisted by Murr and Chessa (2011) is tested in session (4.1.1), which is that the exponential learning curve with 3 parameters describes the individual learning process better than power law model, and the power law model is the representative of averaging those individuals.

Next, the result of differences between the single and multiple curves for each cluster are discussed in the following session. Those clusters are the reduced data set from the entirely averaged data set.

4.1.1 Power Law and Exponential Law

In single learning curves, the power law model with 4 parameters are the best fitting model to the entirely averaged on-line game data set with minimum cost, 2.94. Among exponential functions with different number of parameter, 3 parameter exponential model has the minimum cost, 3.51.



(a) Exponential model with 3 parameters

(b) Power law model with 4 parameters

Figure 4.1: Comparison between exponential function and power function. Both are the best fitting learning curve to the entirely averaged data set among different number of parameters.

Therefore, it is assumed that when all performances of players are averaged together,

learning dynamics disappears, and the global learning curve follows the power law.

4.1.2 Single and Piecewise Curves

For clustered individuals, four functions; polynomial, exponential, power, and step functions, are tested through the possible all situations. As well as single curve fitting, piecewise cases in different number of pieces with window 30 are examined, by multiplying a function or combining different functions together. After that, all squared errors loss of those situations are compared with each other, and the functions which has the least cost value are founded. Though this, the tasks for function fitting become a variety.

Three multiple functions comprised by a single model are iterated 10 times in order to get the costs under the different number of pieces. These functions are as follow,

- piecewise exponential model,
- piecewise power model,
- piecewise step-like model.

Next, combinations of those functions are fitted to the averaged cluster data sets.

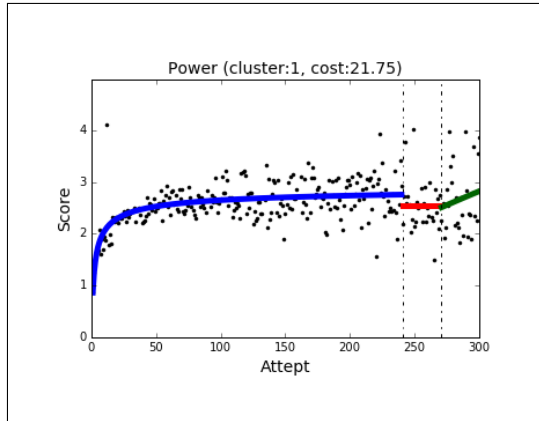
- piecewise exponential and step-like model,
- piecewise exponential and polynomial model,
- piecewise power and step-like model,
- piecewise power and polynomial model.

Fig. 4.8 shows the result of the function fitting. 11 clusters have piecewise functions, while two clusters; cluster 3 and cluster 8, have a continuous single curve; first order polynomial model and power model with 4 parameters. Interestingly, first order polynomial function is fitted best to the individuals in cluster 3. Their performances improves gradually, but the rapid improvement at the early phase in learning with similar to other clusters does not exist. Therefore, they can be assumed as a group who learns

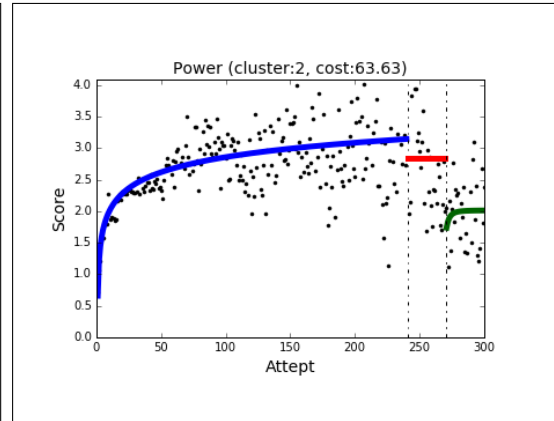
slowly but steadily. Although their learning process is slow, they might be expert after sufficient time, because the learning curve increase without limitation, while others approaches to the asymptotes.

The most dominant model is piecewise power law. In piecewise functions, power law models always appear before the first transition point. 7 clusters have piecewise power curves, not only in the early part of learning before the first transition point, but also in the later trials. 3 groups have a power curve in early attempts, followed by the polynomial functions. Only cluster 7 have the combination of piece wise functions for power and step-like models.

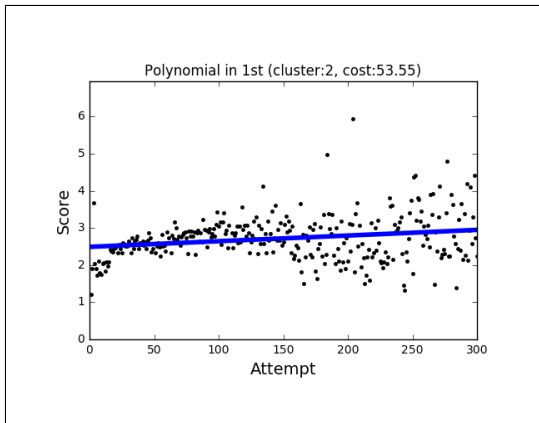
In additions, most first transition points appear after about 200 practice when they experience the stable learning process, which means the learning curves approaches asymptotes. Thus, through these two results that power model exists in almost all piecewise learning processes, and the first transition point appears when learning speed become significantly slow, the hypothesis can be established that individuals may follow the power law to learn any kind of new skills in practice, then they attempt different learning methods in order to exceed the currently slow learning process.



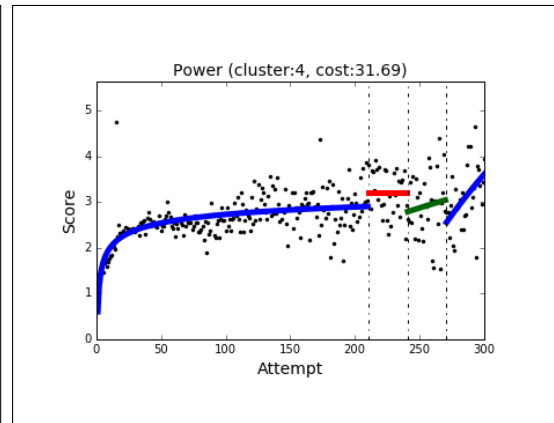
(a) Cluster 1: Power and polynomial functions



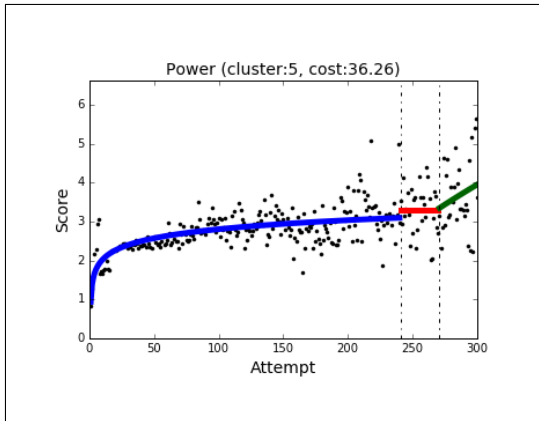
(b) Cluster 2: Power functions



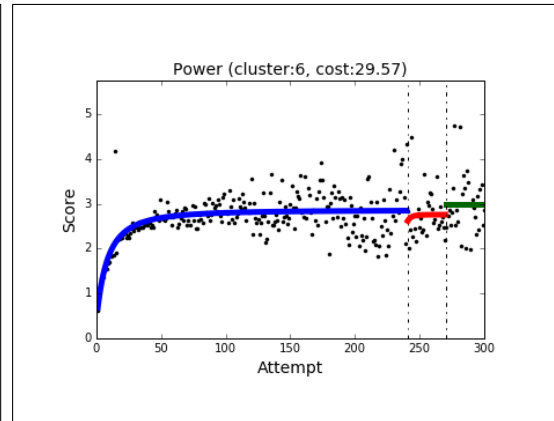
(a) Cluster 3: Single polynomial function



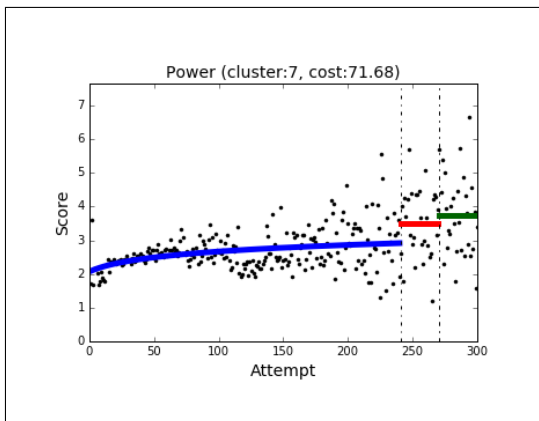
(b) Cluster 4: Power functions



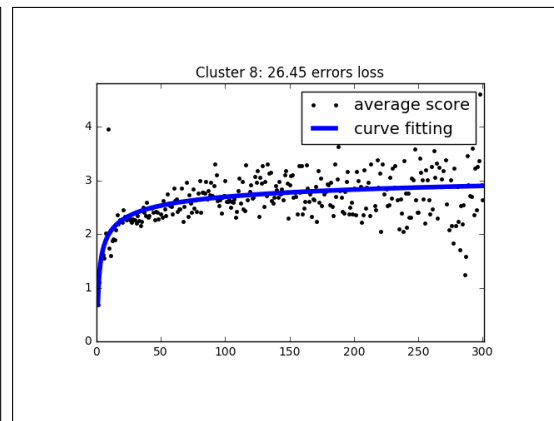
(a) Cluster 5: Power and polynomial functions



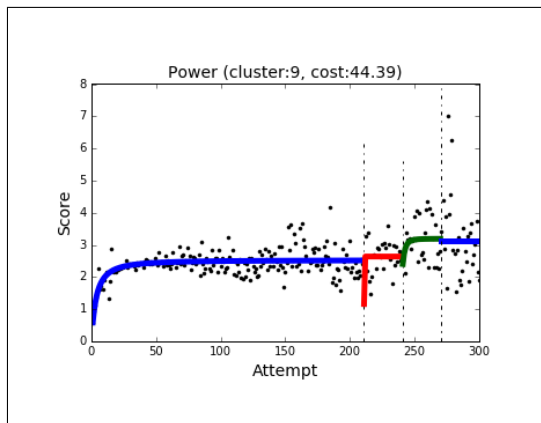
(b) Cluster 6: Power functions



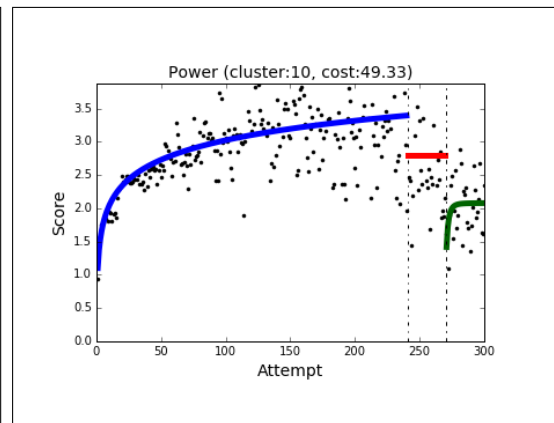
(a) Cluster 7: Power and step-like functions



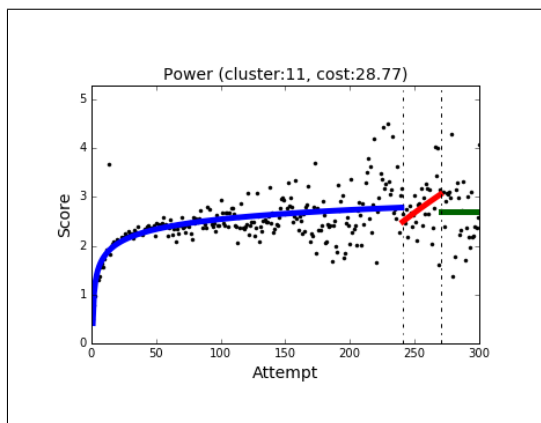
(b) Cluster 8: Single power function



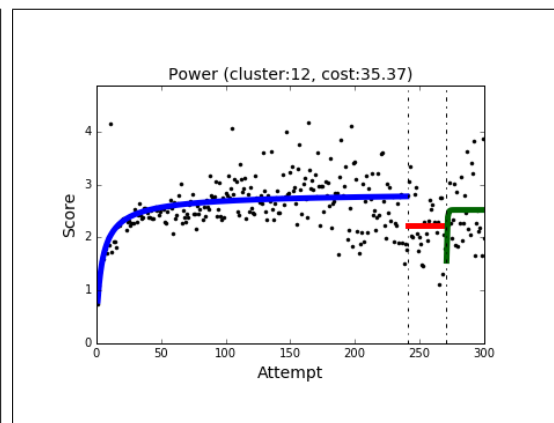
(a) Cluster 9: Power functions



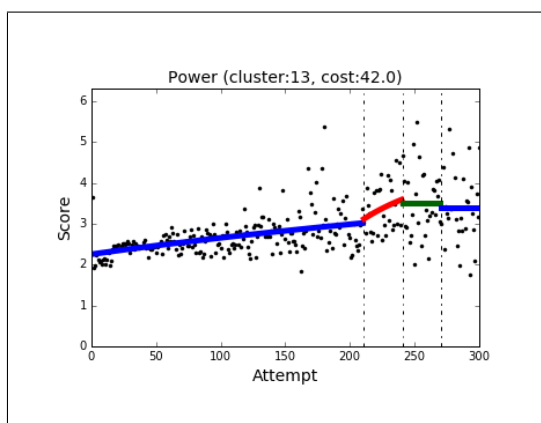
(b) Cluster 10: Power functions



(a) Cluster 7: Power and polynomial functions



(b) Cluster 7: Power functions



(a) Cluster 7: Power functions

Figure 4.8: The result of the best function fitting curves for each clusters.

4.2 Exploitation and Exploration

The process of the skill acquisition in practice can be regarded as the theoretical framework of reinforcement learning (Stafford & Dewar, 2014). The framework of reinforcement learning is based on the situation in which the current decision is taken after considering the future rewards and states. A policy in the reinforcement learning affects the way of decisions at a given time. In other words, "a policy is a mapping from perceived states of the environment to actions to be taken when in those states" (Sutton & Barto, 1998). Depending on the policy, agents may take an action greedily, exploiting, or non-greedily, exploring. The exploration in reinforcement learning represents the case that agents take an action under the currently given or known value of the actions. However, there would be better results beyond the current knowledge and experiences. In this case, the exploration can provide the possibility of the improvement in the estimate of expected rewards, while the outcome after taking the action for the exploration would result in not superior or even worse rewards. Thus, agents are controlled by the policy of which action between exploitation and exploration will take, in order to maximise the future result.

The same as the problem of the trade-off in reinforcement learning, individuals who play the on-line game decide an action whether they will exploit the current knowledge (otherwise, follow the instruction of the game) or explore the game under several plays, in order to get the higher scores. Of course, the on-line game play is involved in not only "decision making", but also "motor responding", and "rapid perception" (Stafford & Dewar, 2014). However, the results of the function fitting to the on-line game data set may imply the framework how players solve the trade-off.

The practice before the first transition point for almost all clusters shows the monotonic improvement in the early phases, getting slower and slower. Finally it approaches the asymptote. All of them, except cluster 3 which has a polynomial learning curve, follow this power law learning process. At the end of the power curve, which is near to asymptote, scores become fluctuated until it reaches to the first transition, then they show different shapes of learning curves with significantly fluctuated scores; but higher than the value of asymptotes.

Therefore, individuals tend to take the exploitation actions at the beginning of the

game, and they does not choose the exploration actions at the beginning. Then, when they feel that their learning process became significantly slow, they try and explore different methods, in order to improve their performance.

References

- Bowles, M. (2015). *Machine learning in python: essential techniques for predictive analysis*. John Wiley & Sons.
- Daumé III, H. (2012). A course in machine learning. *chapter, 5*, 69.
- Donner, Y., & Hardy, J. L. (2015). Piecewise power laws in individual learning curves. *Psychonomic bulletin & review*, 22(5), 1308–1319.
- Gallistel, C. R., Fairhurst, S., & Balsam, P. (2004). The learning curve: implications of a quantitative analysis. *Proceedings of the national academy of Sciences of the united States of america*, 101(36), 13124–13131.
- Gaschler, R., Progscha, J., Smallbone, K., Ram, N., & Bilalic, M. (2007). Playing off the curve-testing quantitative predictions of skill acquisition theories in development of chess performance. *Psychological perspectives on expertise*, 20(40), 137.
- Hackeling, G. (2014). *Mastering machine learning with scikit-learn*. Packt Publishing Ltd.
- Harrington, P. (2012). *Machine learning in action* (Vol. 5). Manning Greenwich, CT.
- Heathcote, A., Brown, S., & Mewhort, D. (2000). The power law repealed: The case for an exponential law of practice. *Psychonomic bulletin & review*, 7(2), 185–207.
- Howard, R. W. (2014). Learning curves in highly skilled chess players: a test of the generality of the power law of practice. *Acta psychologica*, 151, 16–23.
- Idris, I. (2014). *Python data analysis*. Packt Publishing Ltd.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (Vol. 6). Springer.
- Kounios, J., & Beeman, M. (2009). The aha! moment the cognitive neuroscience of insight. *Current directions in psychological science*, 18(4), 210–216.
- Lacroix, G. L., & Cousineau, D. (2006). The introduction to the special issue on ?rt

- $(n) = a + bn - c$?: The power law of learning 25 years later. *Tutorials in Quantitative Methods for Psychology*, 2(2), 38–42.
- Murre, J. M., & Chessa, A. G. (2011). Power laws from individual differences in learning and forgetting: mathematical analyses. *Psychonomic bulletin & review*, 18(3), 592–597.
- Newell, A., & Rosenbloom, P. S. (1981). Mechanisms of skill acquisition and the law of practice. *Cognitive skills and their acquisition*, 1, 1–55.
- python on-line document. (n.d.-a). 12.1. pickle ? python object serialization [Computer software manual]. (Retrieved August 16, 2016, from <https://docs.python.org/3/library/pickle.html>)
- python on-line document. (n.d.-b). Scipy.org - scipy.optimize.least_squares [Computer software manual]. (Retrieved Aug 11, 2016, http://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html#jjmore)
- Rogers, S., & Girolami, M. (2015). *A first course in machine learning*. CRC Press.
- Shafranovich, Y. (2008). Common format and mime type for comma-separated values (csv) files.[sl], 2005. *Disponivel em*:; <http://www.ietf.org/rfc/rfc4180.txt>.
- Stafford, T., & Dewar, M. (2014). Tracing the trajectory of skill learning with a very large sample of online game players. *Psychological Science*, 25(2), 511–518.
- Stafford, T., & Haasnoot, E. (2016). Testing sleep consolidation in skill learning: a field study using an online game. *Topics in Cognitive Science*.
- Stonebraker, M., Bruckner, D., Ilyas, I. F., Beskales, G., Cherniack, M., Zdonik, S. B., ... Xu, S. (2013). Data curation at scale: The data tamer system. In *Cidr*.
- Stuart, P. (2009). Axon - a game for science [Computer software manual]. (Retrieved February 3, 2014, from <http://preloaded.com/axon-game-science/>)
- Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction* (Vol. 1) (No. 1). MIT press Cambridge.

APPENDIX

Gantt charts are attached as bellow.