ABSTRACT

ACKNOWLEDGEMENTS

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Learning curves may reflect different underlying learning processes. For example, in insight learning, it may follow slightly fluctuated level and sudden dramatic increase at some points. Because the learning is sometimes the result of experience through personal interactions with the environment. And following the occurrence of insight, abrupt realisation of how to solve the problem can be repeated in future similar situations. This means that the associated experiences with insights can be parameterised, and it can be linked to future behaviours.

Therefore, there are several possible functions which might fit learning curves and represent different learning processes; step function, powerlaw, piecewise powerlaw and etc. Through possible functions and learning curves, people can be grouped into several learning patterns. Moreover, by inspecting people in those groups, 'how they practice' to get the learning curve and 'which features' having an affect upon individuals can be found. Then suggestions for better learning for individuals may be categorised and proposed.

For this, online games are a reasonable instrument. As it involves "rapid perception, decision making, and motor responding", as well as it gives rich details of practice history, in order to find individuals' learning curve and investigate their features (Stafford and Dewar, 2013)

Text...

1.2 Problem Definition

There are three main issues in traditional reinforcement learning curve. Because it has smooth power law with diminishing gain derived from average value, all possible individual learning curve are squashed into one learning curve. Thus, it cannot tell which learning curve will belong to a task, behaviour, and learning process. Furthermore, even though identifying bad or good on learnings are important, but it cannot provide sufficient information about those.

Below is the list of problems on current reinforcement learning curve.

- Not being able to categorise learning process.
- Averaging possible individual learning curves
- Not enough information for identification of learning success

1.3 Aims and Objectives

These are the concepts I think you should cover in your introduction:

Skill acquisition & expertise - factors which are known to influence skill acquisition - practice amount, practice spacing, others

Using games to study skill acquisition

Learning curves - different possible underlying functions - problems with averaging over

The ultimate aim, of course, is to identify individuals who learn faster or for longer, and try and relate this to how they practice The aim of this paper is to fit different functions to the Axon game data. In the data, there are 854,064 individuals data about when, where they did the game and other informations, as well as scores of game according to their attempts. And so identify the underlying learning process. Furthermore, it is to inspect features which may have influences on the learning curves.

- Understand several possible functions; step function, powerlaw, piecewise powerlaw and etc.
- Understand different underlying learning processes; insight learning, associative learning, multi-component/process learning and etc.

- Fit functions to individual Axon game data and compare those to identify.
- Test theories of what makes learning most effective, exploiting unsupervised learning, establish which parameter is important for getting learning curve from the data.
- Design more effective learning practices.

1.4 Project Management

Text...

Chapter 2

Data Curation

A large number of data were acquired through the online game named Axon which is developed by Preload for Welcome 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 toal 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. ??? out of the total number of players attempted the game more than 300 times, and also they were removed. Also discarded players having unrecognisable time information were ???. 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. Though cleaning those is simply mentioned in this session, the specific method how to manage players with discontinuous data will be discussed in chapter ???.

As a result, raw data with 4, 038,802 plays with xxx players were cleaned, and then reduced to ??? plays and ??? individuals decreased by ???.

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, ”$local\ time = UTCtime + (longitude 24 \div 360), modulo 24$” (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 on 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,

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, Fairhurst, & Balsam, 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 predefined 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).

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

and cost function J is

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

where μ is mean of cluster, **D** is (the number of parameters) data-set, **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).

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

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
        $\mathbf{C}_k \leftarrow \arg\min_x ||\mu_k - x_i||^2$ ,
      end for
    end for
    for all $k \in K$ do
      $\mu_k \leftarrow \text{mean}(\mathbf{C}_k)$
    end for
  end while
  return $\mathbf{C}, \mu$
```

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.

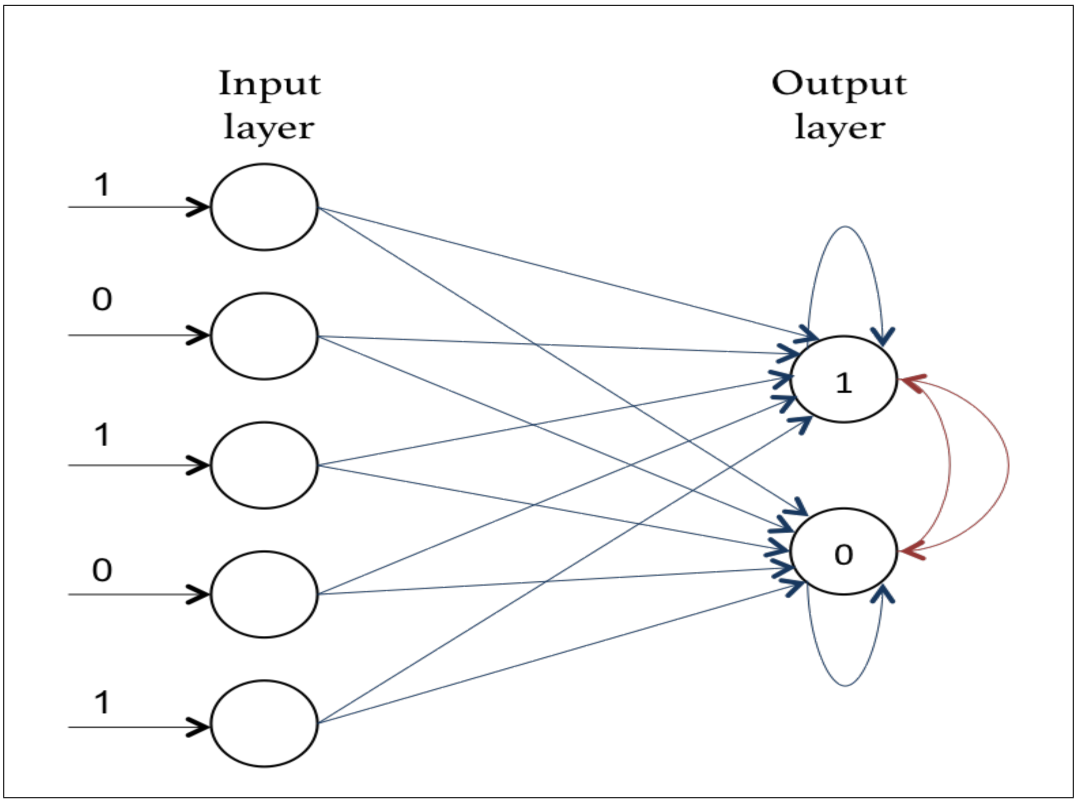


Figure 2.1: Competitive neuron diagram. When an input enters the network, it generates output values according to weights between an input and output neuron (blue arrows). Output neurons compete with each other (red arrows), then only one output fires on an input pattern. Output neurons are considered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

$$O_i = \sum_j w_{ij} x_j = \vec{\mathbf{w}}_{\mathbf{i}} \cdot \vec{\mathbf{x}}, \tag{2.4}$$

for an input vector **x**.

Winner neurons are found by (2.5),

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

where winner neuron is $\vec{\mathbf{w}}_k^\top$, and $\vec{\mathbf{x}}^\mu$ 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, $\vec{\mathbf{w}}_{\mathbf{i}} = 1$, then (2.4) can be redefined as below,

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

(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.

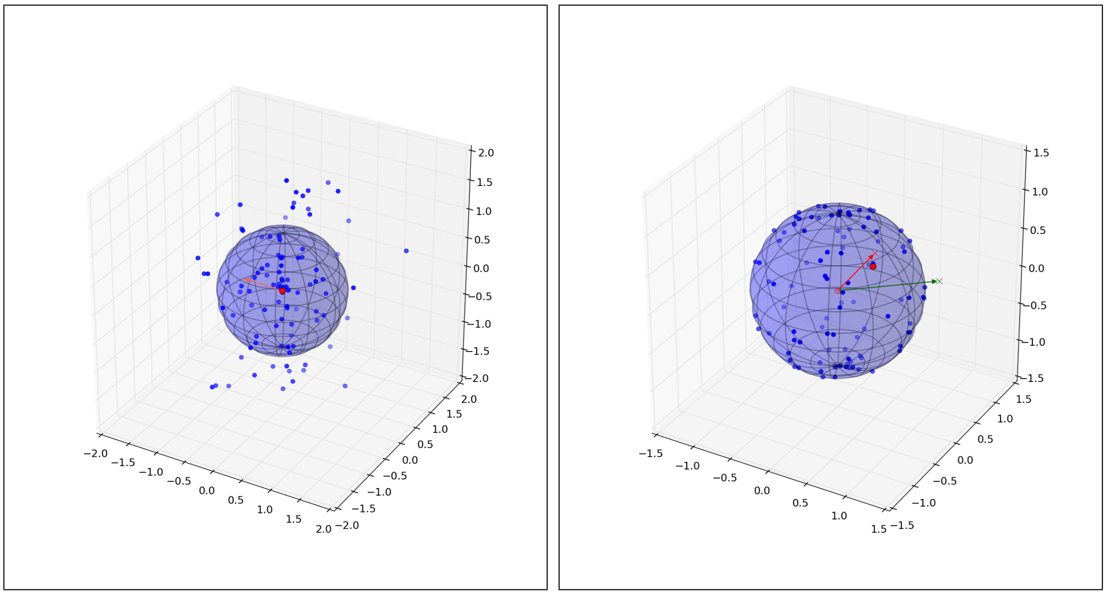
For competitive learning, the thing to note is that weights must be normalised in order for all units to have a unit length.

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

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

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 definded 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,

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

   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 are ??? players in the on-line game data-set, and it has no predefined 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,

$$\begin{aligned} \text{Bayes Information Criteria (BIC)}: &\quad \arg\min_K (J_K + K\log D) \\ \text{Akaike Information Criteria (AIC)}: &\quad \arg\min_K (J_K + 2KD). \end{aligned} \tag{2.10}$$

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. Difference between 13K and 14K is xx and, 14 K and 15 K is yy. 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,

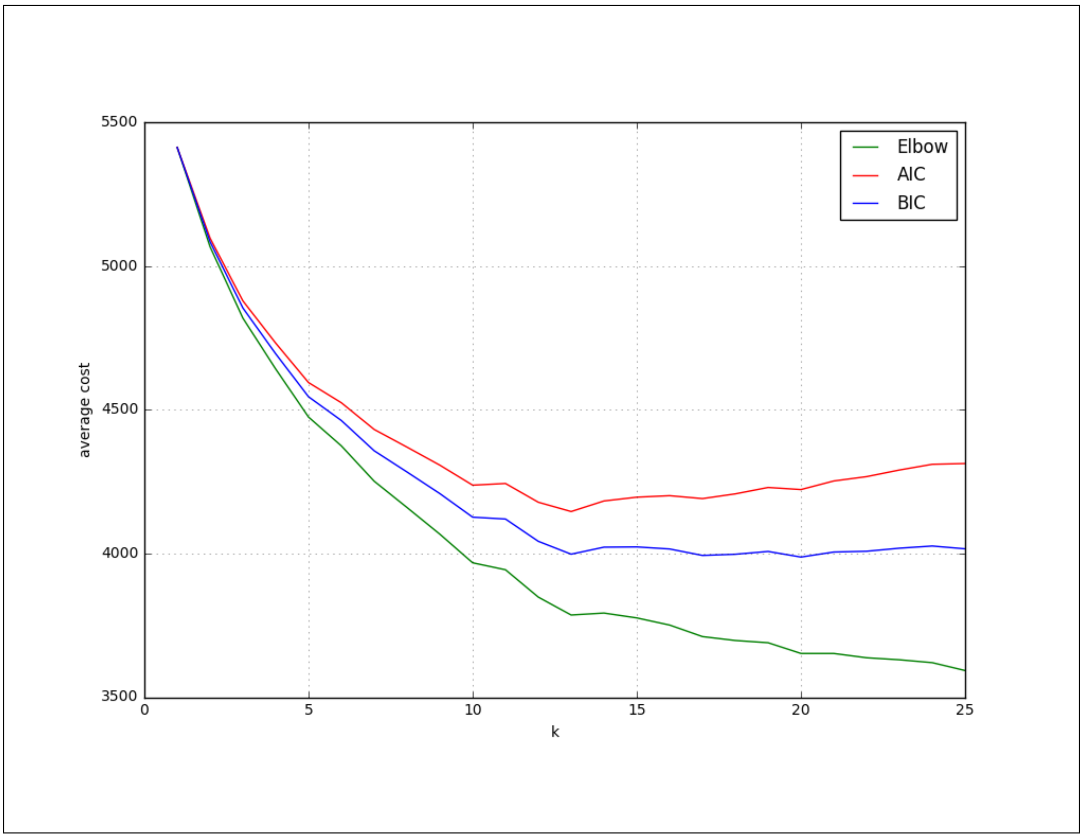


Figure 2.3: Comparison between AIC and BIC according to the different number of k. AIC are more strict method than BIC to identify the number of clusters.

(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.

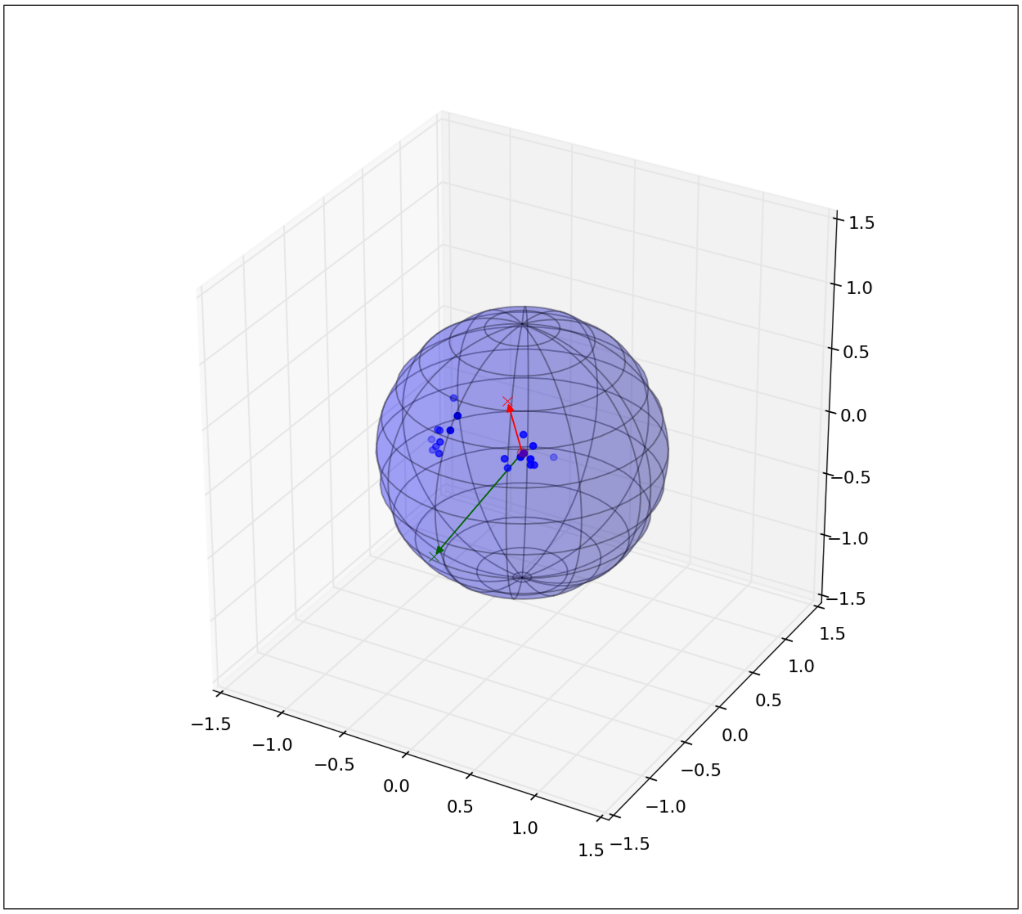


Figure 2.4: An example of dead unit problem. Red and Green arrows are initialised weight before applying into the network. Green arrow is set far from both two input clusters, and it is assumed as a dead unit. Unfortunately, green arrow will never fire through the learning, and which means that input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

In (2.5), ??? never fired and it is the dead unit, even though neuron ?? and ?? won several time compared to other ?? output neurons.

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

1. leaky learning (LL),

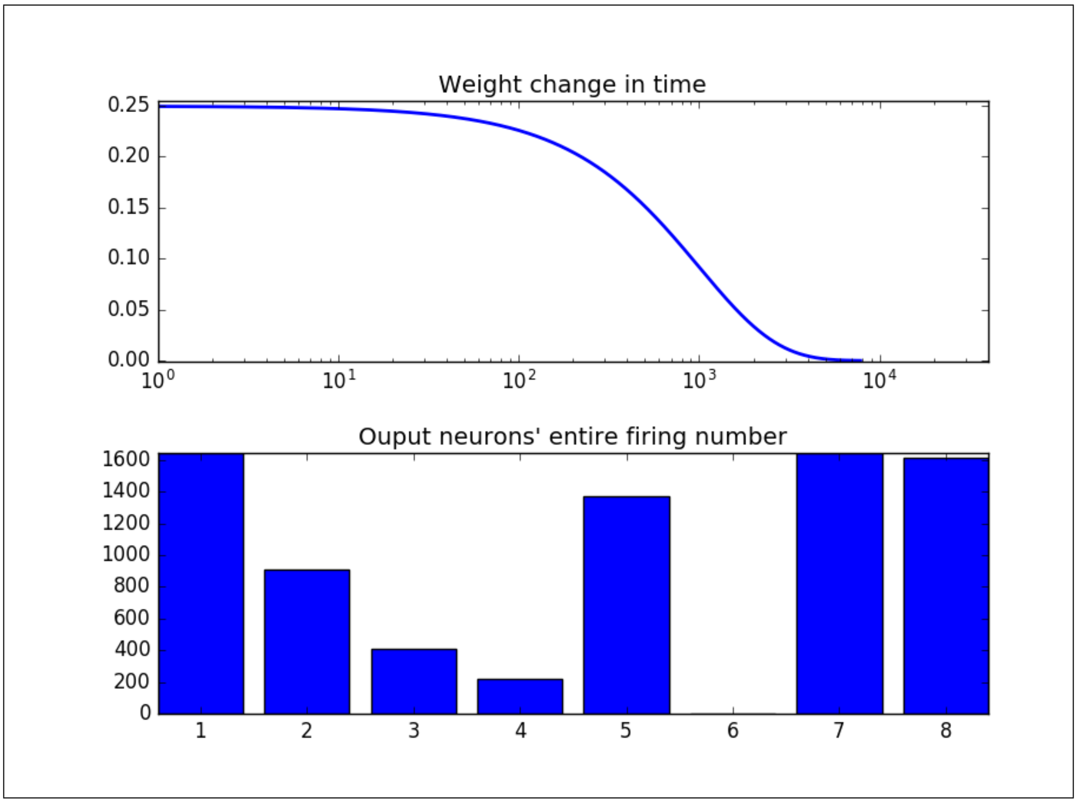


Figure 2.5: Learning rate and existence of dead units

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

will be discussed.

First, leaky learning (LL) is the way of adjusting learning rate for both winner and lower 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,

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

The second way to deal with dead units is adding bias (AB) to winner output unis which repeatedly fire. Then, they become hard to win again and again.

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

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).

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

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.

$$\begin{aligned} w_{ij}^{init} &= w_{ij}^{random} + \varepsilon, \\ \text{where } \varepsilon &\sim N(0, \sigma^2) \end{aligned} \tag{2.14}$$

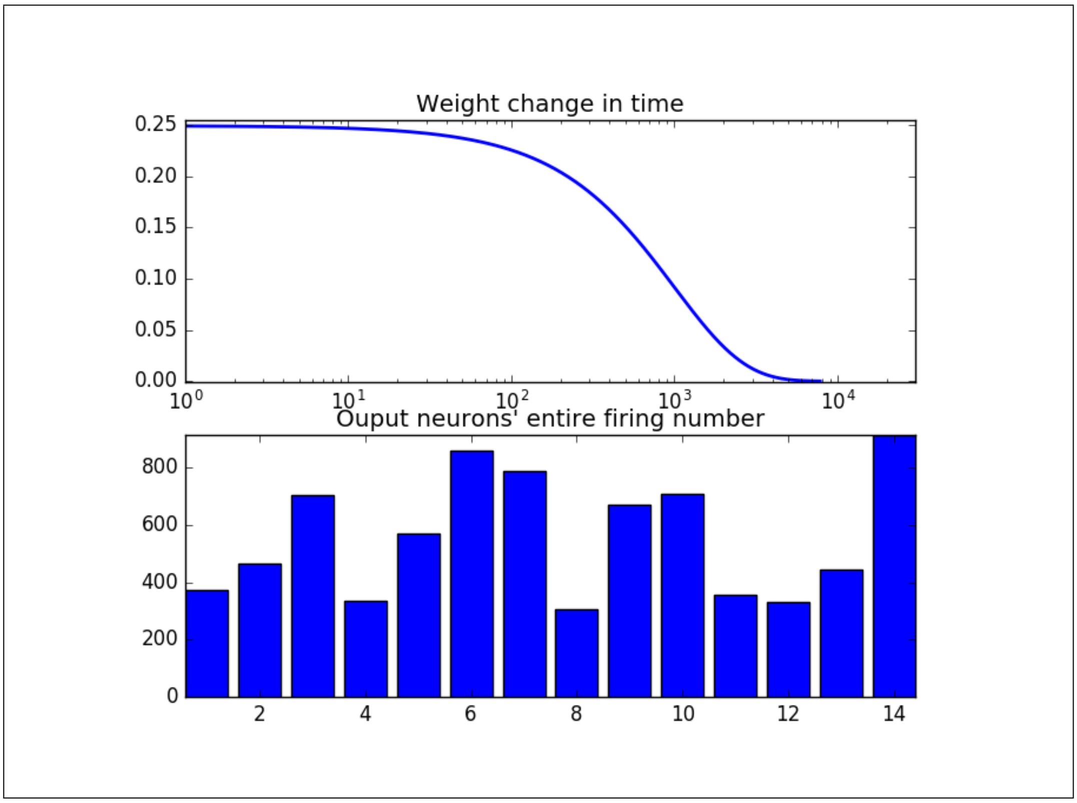


Figure 2.6: Learning rate and and entire firing number of output neurons after using SV approach for dead units

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.

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.

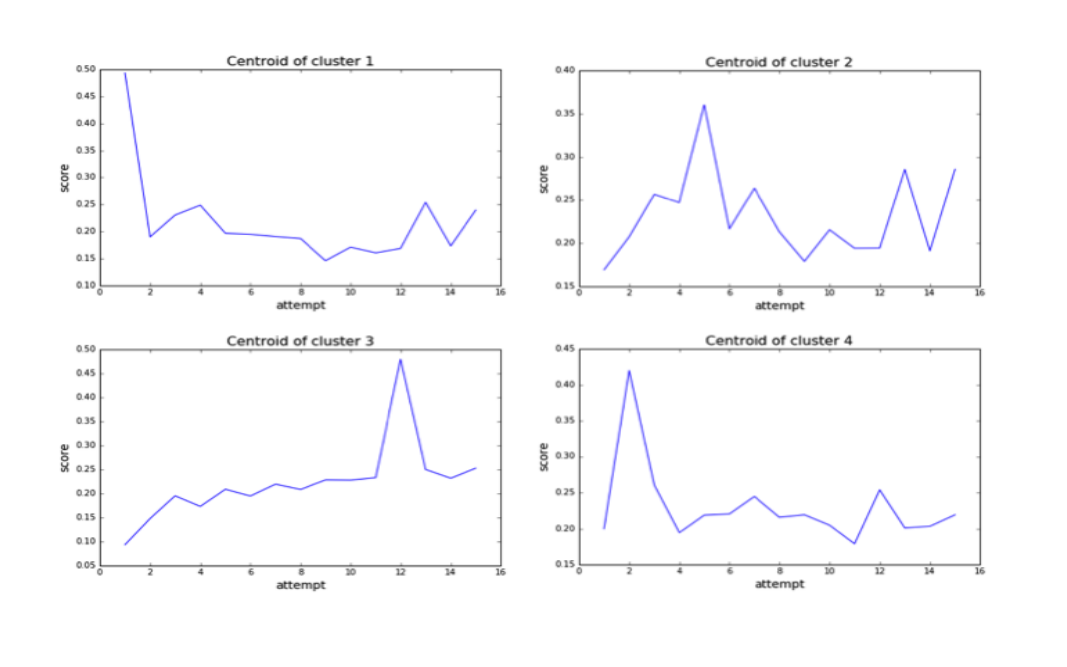


Figure 2.7: Centroids of cluster from 1 to 4

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

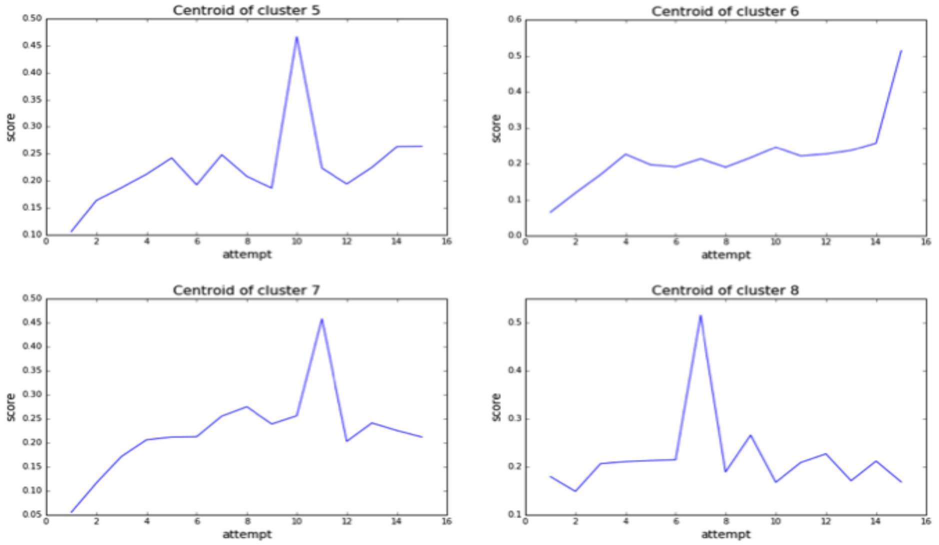


Figure 2.8: Centroids of cluster from 5 to 8

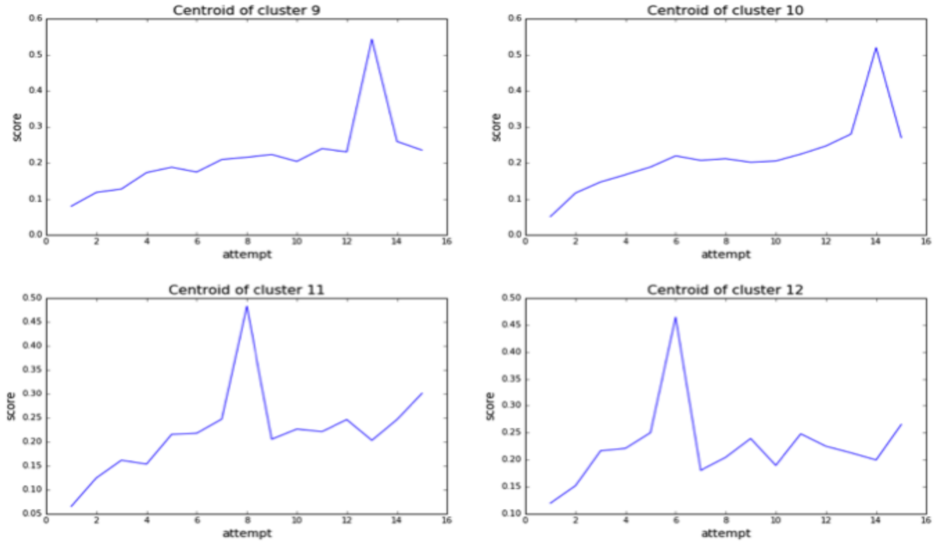


Figure 2.9: Centroids of cluster from 9 to 12

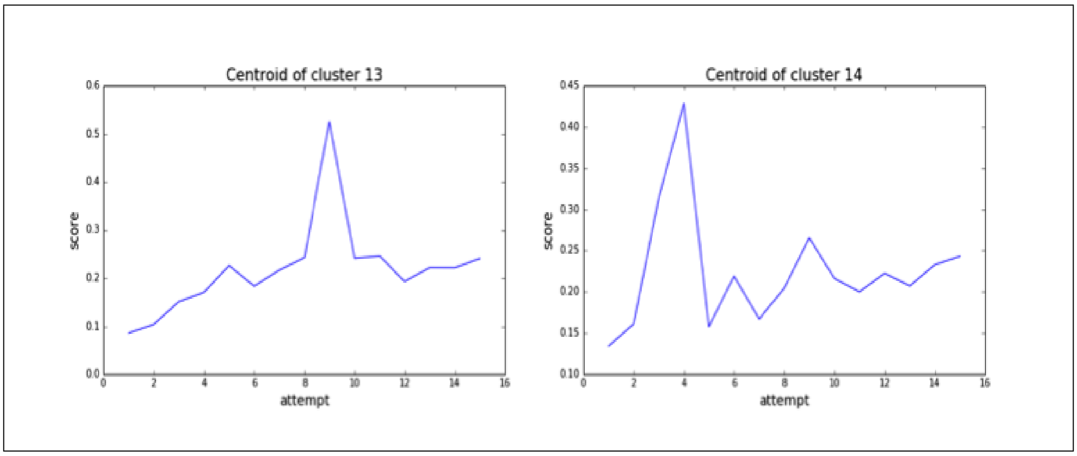


Figure 2.10: Centroids of cluster 13 and 14

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

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

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,

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

where parameters are w_0, w_1 and w_2, x_n represents data points, and $n \in \{1, 2, \cdots, n\}$. 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,

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

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 3.2.3.

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 function.

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,

$$\begin{aligned} \text{3rd order model :} \quad & f(x_n;\, \mathbf{w}) = w_0 + w_1 x_n + w_2 x_n^2 + w_3 x_n^3 \\ \text{4th order model :} \quad & f(x_n;\, \mathbf{w}) = w_0 + w_1 x_n + w_2 x_n^2 + w_3 x_n^3 + w_4 x_n^4 \\ \text{5th order model :} \quad & f(x_n;\, \mathbf{w}) = w_0 + w_1 x_n + w_2 x_n^2 + w_3 x_n^3 + w_4 x_n^4 + w_5 x_n^5. \end{aligned} \tag{3.4}$$

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 3.2.2, 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.

$$
\begin{aligned}
\text{one parameter}: &\quad f(x_n;\, w_0) = w_0 + e^{x_n}, \\
\text{two parameters}: &\quad f(x_n;\, \mathbf{w}) = w_0 + w_1 e^{x_n}, \\
\text{three parameters}: &\quad f(x_n;\, \mathbf{w}) = w_0 + w_1 e^{w_2 x_n}, \\
\text{four parameters}: &\quad f(x_n;\, \mathbf{w}) = w_0 + w_1 e^{w_2 (x_n + w_3)},
\end{aligned}
\tag{3.5}
$$

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, Brown, & Mewhort, 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.

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

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 $-infty$, 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,

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

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,

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

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

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 theses conditions. ??? 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

Through the previous chapter, it is discussed that 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 containing many noises, and they make hard to identify true parameters of the models with such high disturbance. Furthermore, even after feasible parameters are found, some 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 criterion to search for optimal parameters. Squared lost function is one of possible and well known method 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 close as possible the data points.

The squared lost function is defined as below,

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

where $\mathscr{L}_n$ represents cost value of the difference between true value y_n and trained value $f(x_n)$, and $\mathscr{L}_{\text{avg}}$ 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 finding such geometrical turning points with respect to parameters, and it will tells the minima corresponding 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),

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

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,

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

Then, (3.13) is solved with respect to **w**, using the Python command ***numpy.linalg.solve*** or QR decomposition,

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

where **Q** is orthogonal matrix, so that $\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}$, and matrix **R** is upper triangular. QR decomposition may provide a solution with more numerically stability, because $\mathbf{X}\top\mathbf{X}$ is removed, in which all elements are squared and then potentially numerical precision is disappeared. However, QR decomposition will operate on X directly, preserving the model's numerical precision. Finally, second derivatives tells 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 $\eta$ {learning rate prevents overshoot problem}
while $p_{n\_new} - p_{n\_old}$ is not greatly smaller do
  for $i = 0$ to $n$ do
    $p_{n\_new} \leftarrow p_{n\_old} + \eta \frac{\partial \mathscr{L}}{\partial p_n}$
  end for
end while
```

Through these two methods, it can be determined that the optimal parameters which 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 for 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 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, this criteria cannot afford the comparison method between a single function and piecewise model, because separating the data range as smaller as possible means fitting data set in small interval, and will lead definitely and undoubtedly smaller cost (over-fitting), such as the situation that increasing the number of clusters leads smaller mean square errors in session (2.4.5).

Regularisation prevents the model to become 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.

3.2.3 Training Set and Validation Set

Separating the entire data set into two parts; training set and validation set, is another method to deal with the over-fitting problem. Ideally, the best fitting learning curve can generalise predictions close to the true value. Therefore, separating the individual data-set with 300 attempts to two parts can have the same role of the generalisation. Although the model fits well on the training set and have 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.

validation check

At first, assumption for learning processes will be tested on 14 data sets in which groups show similar learning patterns between 15 attempts and scores.

3.2.4 Choice of Breaking Point

Fig. 3.1 shows an example how to find the exact transition points. Firstly, 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). Minimum 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 have remained 150 of attempt range as an initial condition.

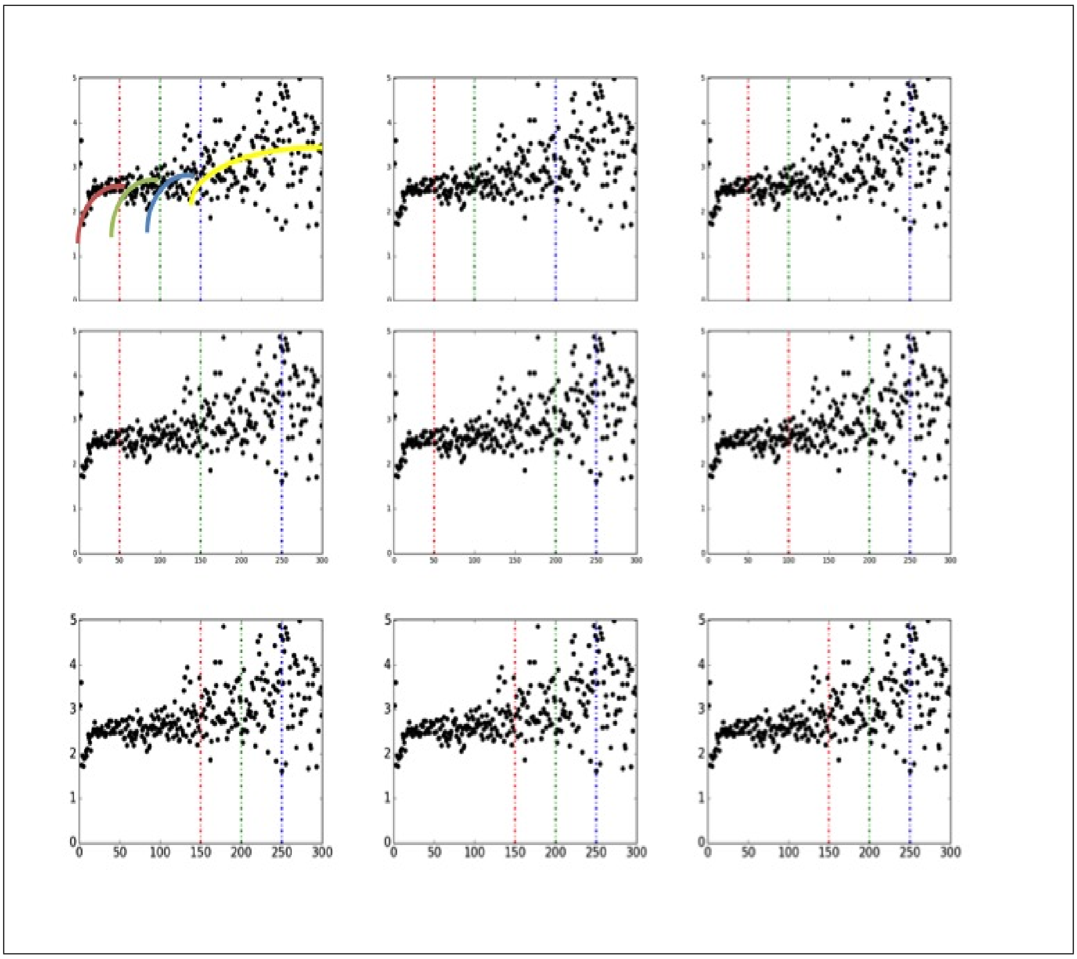


Figure 3.1: Transition point shift in 4 pieces

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

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

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

Because a start point of the first piece does not change, all elements in the first row are 1. How many 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 has a minimum error cost can be identified. Firstly, it is repeated that one element in each row is derived and summed. Then the total value from every iterations 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**
        $l \leftarrow \sum \mathrm{row}_{i,j}$
        **if** $l$ is 300 and element$_{i+1,j}$ > 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 $\min c$
```

3.3 Curve Fittings

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.

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.

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. *Disponıvel 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/)