University of Glasgow | School of Computing Science

Predict the Answer

Konstantinos Psychogyiopoulos

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at the University of Glasgow

September 7, 2018

Abstract

Predictive modeling and analysis is essential nowadays, due to the rapid growth of Big Data. Companies and organizations store and process large volumes of data. Being able to predict the response to a given query without accessing the storage infrastructure is vital and thus companies strive to explore new methods and techniques to extend further the quality and speed of services provided to their customers.

We implement a predictive model which does not require frequent communication with any storing systems, decreasing the cost of query prediction and increasing the query prediction throughput since the increase in database size does not affect query execution.

Our model consists of 2 phases. The prediction phase is based on the Rival Penalizing Competitive Learning algorithm which is used to cluster the dataset that is generated by the training phase and produce accurate responses to any queries fed into our model.

Lastly, we evaluate the suggested model by calculating the predictive error between the true and predicted answers over specific queries that were generated for this purpose. A number of query generation techniques is used to calculate the respective evaluation results. Lastly, we comment on the findings and suggest future work.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: Konstantinos Psychogyiopoulos **Signature:** [illegible]

Acknowledgements

I would like to thank Dr. Christos Anagnostopoulos for his valuable guidance and assistance as my supervisor throughout the whole process of this project, whose contribution was most significant for the completion of my thesis.

Furthermore, I would like to express my sincere gratitude to the Data Lab Msc for funding my postgraduate studies in the University of Glasgow.

Last but not least, I would like to thank my family for their encouragement and support throughout my studies.

Contents

1 Introduction **1**

2 Analysis and Requirements **2**

2.1 Background and Previous work. 2

2.2 Tools and Techniques. 3

2.3 Analysis. 4

2.4 Aims and Objectives. 8

3 Design and Implementation **9**

3.1 The RPCL Algorithm. 9

3.2 System architecture major design decisions. 11

3.2.1 Training the Model. 11

3.2.2 Predicting the Answer. 13

4 Evaluation **15**

4.1 Evaluation Metrics. 15

4.2 Evaluation Strategy. 16

4.3 Evaluation Results. 17

5 Conclusion **20**

5.1 Discussion of Evaluation Results.. 20

5.2 Occam's Razor and Over-fitting. 20

5.3 Summary. 22

5.4 Future Work. 23

A Appendix **24**

A.1 Flowchart presenting the steps followed throughout our Predictive Model's process. 24

A.2 Snippet of code, which shows the calculate() function. It is the algorithm which calculates the true response to a given query. 25

A.3 Snippet of code, which shows the RPCL() class which uses the fit() function.. 26

A.4 Predictive Error – Complexity Curves for the "Airfoil Self-Noise" Dataset using 5 query generation scenarios.. 27

Bibliography **29**

List of Figures

2

2.1 Implemented K-Means by Scikit-learn for the “Airfoil Self-Noise” Data set.. 5

2.2 Batch K-Means algorithm for the “Airfoil Self-Noise” Data set. 6

2.3 Implemented K-Means by Scikit-learn for the “Airfoil Self-Noise” Data set with k=6 number of clusters.. 7

2.4 RPCL version of online K-Means for the “Airfoil Self-Noise” Data set with k=6 number of clusters... 7

3

3.1 The RPCL algorithm Source: L. Xu and A. Krzyzak. *Rival Penalized Competitive Learning for Clustering Analysis, RBF Net, and Curve Detection.* IEEE Transactions on Neural Networks, 1993. 9

3.2 The RPCL algorithm – a different scenario. 10

3.3 RPCL clustering with Principal Component Analysis... 12

3.4 RPCL clustering with t-Distributed Sochastic Neighbor Embedding... 13

4

4.1 Predictive error (RMSE) for different number of clusters using the “Individual household electric power consumption” dataset.. 18

4.2 Predictive error (RMSE) for different number of clusters using the “Airfoil Self-Noise” dataset (100 iterations)... 19

5

5.1 The Elbow Method (SSE – k number of clusters).. 21

5.2 Predictive error (RMSE) for different number of clusters using the “Airfoil Self-Noise” dataset (100 iterations) and the Gaussian distribution with moving z.. 22

List of Tables

4

4.1 The plot shows the clustered data points for the Airfoil Self-Noise Data set, using the Implemented K-Means by Scikit-learn deploying 2 attributes: Free-stream velocity and suction side displacement thickness 17

4.2 The plot shows the clustered data points for the Airfoil Self-Noise Data set, using the batch K-Means algorithm (https://www.youtube.com/watch?v=HRoeYblYhkg) deploying 2 attributes: Free-stream velocity and suction side displacement thickness. 18

Chapter 1

Introduction

With the introduction of Big Data in 2005 [1] enterprises handling large volumes of data, seek cost efficient ways to access and process their data. In our dissertation, we focus on predicting queries issued over a data set. Our technique does not require access to DBMS engines, thus our predictions are insensitive to data sizes. It can be considered as a Predictive Modeling and Analytics technique, based on the previous work by C. Anagnostopoulos et al [2].

To achieve query prediction, we first familiarize with the widely used K-Means algorithm. We customize the algorithm and create a competitive version of the online K-Means algorithm, called Rival Penalizing Competitive Learning (RPCL) [3]. Using downloaded data sets from the UCI repository, we create a set of queries for which we calculate their true answers y. Given our new query-answer generated data set, we create a smaller set of queries which will be issued over the query-answer data set to predict their answers. Our reasoning behind this decision, is that once we have trained our model, that is we have the generated query-answer data set, we will not have to access the original data set for any future queries issued to the system. The response-answer prediction to any given queries is achieved through our implemented version of the RPCL algorithm.

We generate random queries using different distributions. In particular, we apply Uniform, Gaussian and Multimodal Gaussian distributions, observing the results and commenting on the findings. Moreover, the RPCL algorithm that is used for prediction, aims to detect the optimal number of K clusters for a given data set. This is the K value that we will pass as a hyperparameter to our predictive algorithm, in order to classify a query to a particular cluster. We provide measurements over a range of K values to demonstrate the behavior of our model.

The report consists of 5 chapters in total, Introduction, Analysis and Requirements, Design and Implementation, Testing and Evaluation details, and Conclusion.

In Analysis and Requirements chapter we discuss any previous work that is correlated with our topic. We describe the tools and techniques that were used from the start to the completion of our work, provide analysis by describing the data collection techniques and the hypotheses made. Lastly, we present our thesis aims and objectives that we followed throughout the whole process.

In Design and Implementation chapter we present our own version of the Rival Penalizing Competitive Learning algorithm and we run through the algorithms used and major design decisions made, to describe the system’s architecture.

In the Testing and Evaluation details chapter we describe the software evaluation strategy and provide measurements over the testing experiments conducted, demonstrating the effectiveness of our model. We discuss issues surfaced during the evaluation process and interesting findings.

Lastly, in the Conclusion chapter we state our overall achievements and limitations that we faced throughout our work and set new targets for future work.

Chapter 2

Analysis and Requirements

2.1 Background and Previous work

Our current work is based on “Efficient Scalable Accurate Regression Queries in In-DBMS Analytics” by C. Anagnostopoulos and P. Triantafillou. Living in the Big Data era, being able to acquire the exact answers to issued queries from large data sets, requires a lot of resources in terms of efficiency and scalability, thus it can be costly and time inefficient.

C. Anagnostopoulos et. al [2], suggested a novel query processing algorithm that monitors queries and responses in between users and the DBMS. In their work, they focus on two important queries: mean-value and linear regression queries. Using previously executed queries and their answers, they train a model which is then used to predict the answer for any unseen, new queries and predict the coefficients of linear models that belong to different sub-spaces. These separate linear models form a ‘piece-wise’ linear regression, approximating more accurately the linearity of a feature u, which can be a plane in $2 - dim$ space or a hyperplane in dimension space with $d \geq 3$. They achieve high query processing efficiency and accurate approximations of the underlying correlations among the attributes of a given data set.

Clustering is the partitioning of a dataset into sub-spaces according to their characteristics/attributes. Clustering is applied when our dataset is not pre-labeled, thus it is an unsupervised learning technique with no ground truth available.

K-Means is a popular clustering algorithm, on which our work is based, regarding the answer prediction to any unseen/new queries. It is a method of vector quantization firstly introduced in signal processing and is widely used nowadays in data clustering. It aims to partition n observations (datapoints) into k clusters, where k << n. The algorithm has $O(n^2)$ time complexity, where n is the input data size. The algorithm is easy to implement and performs well for a large number of variables/attributes. On the contrary, defining the K value can be a difficult task. Also, assigning the initial seeds (centroids) as well as the ordering in which the datapoints are ‘fed’ into the algorithm may affect the algorithm’s performance [4], [5].

Online K-Means algorithm which is also known as sequential K-Means, allows us to update our model as new data is given as input to the algorithm [6].

An adaptive version of online K-Means, Competitive Learning was introduced and widely applied in statistical data analysis, unsupervised pattern recognition and vector quantization. Winner-Take-All rule is the essence of simple competitive learning algorithms. Amongst the initial seeds (centroids), the seed that is closer to a given datapoint becomes the winner and moves towards the datapoint with a step $\Delta\bar{\omega}$. Rumelhart et al. [7], in 1986 addressed the problem of the under-utilized or dead-unit, according to which an initial seed is not moving during the process. This depends heavily on the random assignment of the initial seeds [3].

Frequency Sensitive Competitive Learning (FSCL) [8], was one of the attempts made to address the dead-unit problem. It utilizes the simple Competitive Learning algorithms adding a new term, the conscience. In practice, conscience is a strategy to reduce the winning rate of frequent winners. Given 2 initial centroids A and B, if centroid A achieves a number of consecutive wins at i-th iteration of the algorithm, then it stops moving and centroid B takes one step towards the datapoint that is fed into the algorithm at $(i+1)$-th iteration.

Although conscience strategy addresses the dead unit problem, it causes a new one [3]. When the k number of initial seeds/centroids is larger than the actual clusters in the input data set, all the k vectors will be moved to some places in the data set and some vectors will not longer be located at the center of the clusters. Therefore, FSCL works well when the number of clusters in the input data set in known in advance.

Rival Penalized Competitive Learning (RPCL) was suggested by L. Xu et. al [3], tackling the problem of selecting the initial K value. The algorithm adds a new term into FSCL, the second winner. For each datapoint given as input into the algorithm, the closest, in terms of Euclidean distance, initial seed is assigned as the winning seed and the second closest initial seed, is assigned as the 2nd winning seed – the rival. The algorithm 'awards' the winner-centroid by modifying its weight vector w_c to adapt to the input datapoint and de-learns the 2nd winning centroid's weight vector w_r by a learning rate smaller than that used by w_c.

We predict the answers to our queries using Rival Penalized Competitive Learning, which can be thought of, as a Penalizing Competitive version of online K-Means algorithm.

2.2 Tools and Techniques

We perform query prediction and evaluation of the results over 2 datasets downloaded from the UCI repository. The datasets used are:

i) Individual household electric power consumption Data Set:
https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption
[9]

The dataset contains 6 attributes (date, time, global active power, global reactive power, voltage and global intensity) and 3 responses (sub-metering 1,2 and 3).

ii) Airfoil Self-Noise Data Set:
https://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise [10]

The dataset contains 5 attributes (frequency in Hertz, Angle of attack in degrees, Chord length in meters, Free-stream velocity in meters per second and Suction side displacement thickness in meters) and 1 response (Scaled sound pressure level in decibels).

We implement our predictive model in Python 2.7. The environment used is Jupyter Notebook, an open-source web application and Google Colab a free cloud service provided by Google. The libraries used are: Pandas, NumPy, Matplotlib, Scikit-learn and Plotly.

As mentioned in Chapter 2.1, we will be using Rival Penalized Competitive Learning [3] to predict the corresponding answers to the input-queries fed into our algorithm. Since our algorithm will be based on K-Means, we initially create a custom version of online K-Means algorithm, using as benchmarking measurement the K-Means algorithm of Scikit-learn package, to compare and verify the validity of our results. Then, we proceed in implementing the RPCL algorithm.

The training phase of our algorithm, prior to the prediction phase, will be described in detail in Chapter 3.

2.3 Analysis

In this section we will present the data collection techniques used and the hypotheses made. We also present the first part of our work, which includes familiarization with the K-Means algorithm, results and findings.

To give our uploaded data a structured, hierarchical form we use Pandas data frame. In this way we can mold our data into a 2-dimensional labeled data structure with columns of potentially different types (integers, float numbers, strings, etc.) The structure includes index (row labels) and column (column labels) arguments, which will help us with data processing. [3: https://pandas.pydata.org/pandas-docs/stable/dsintro.html].

The hypotheses we made for our work are as follows:

i) The X attributes of each dataset used are independent to each other, thus we use randomly selected attributes from each data set. The attributes used for our testing and prediction algorithms may vary from 2 to 3.

ii) The Y responses of each data set, given the data set includes more than one responses, are independent to each other. We use one response to train our model.

iii) According to D. Bora et. al [11], Euclidean distance may not be the best option, neither in terms of computational efficiency, nor in terms of better interpretation of the clustered data. R. Loohach et. al [12] suggest that Euclidean distance should be preferred when we there are $k = 4,5,7$ clusters created, while A. Singh et. al [13] state that the Euclidean distance, as a distance metric, outperforms Manhattan distance metric when used for K-Means algorithm. For our clustering algorithms, we use pairwise Euclidean distance to calculate the distance between vectors (centroid – data point).

iv) For our implemented Rival Penalizing Competitive Learning algorithm, which we will use for the prediction phase of our model, we define the optimal k number of clusters, for a given dataset, according to the counters of the clusters' population. That is, the number of data points assigned in one cluster due to similar characteristics. We set a threshold according to which, clusters with count representing the 10% of the total data points will not be considered as a potentially good cluster. For example, in a dataset of 100 query-answer pairs, if we run the RPCL algorithm for $k = 6$ and we get a list of counters $c = \{11, 8, 56, 100, 2, 70\}$ then the optimal number of clusters will be $k = 4$.

Our implemented version of the Rival Penalizing Competitive Learning Algorithm [3], is based on the well-known K-Mean algorithm. As a first step we run the already-implemented K-Means algorithm by Scikit-learn package on one of our data sets from UCI repository. In Figure 2.1 we show the clustering results for the Airfoil Self-Noise Data Set from UCI repository.

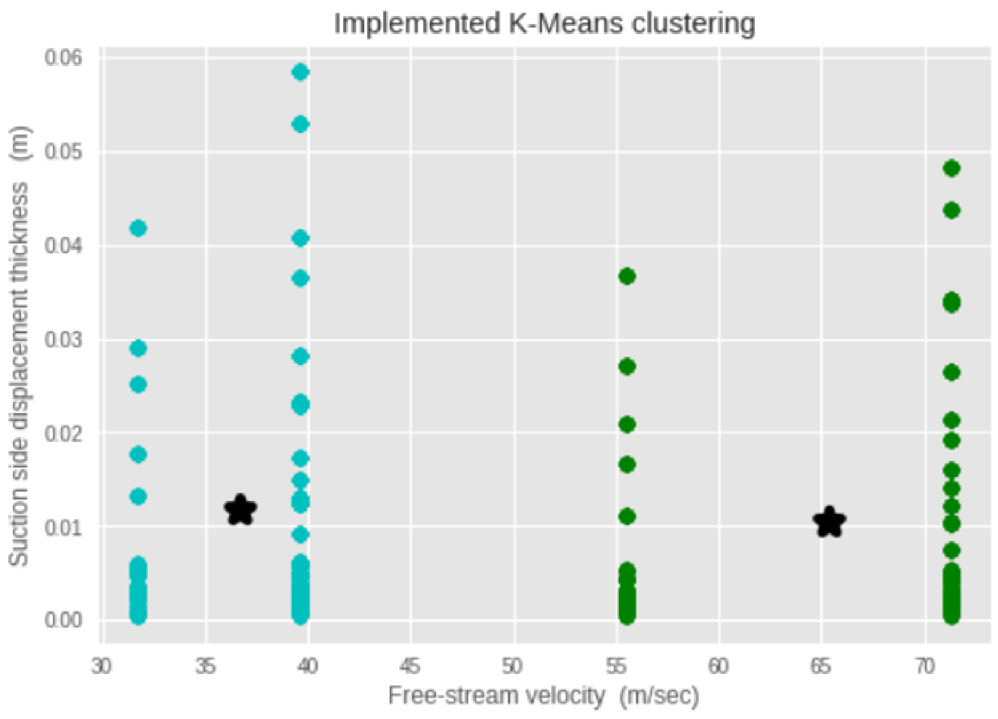
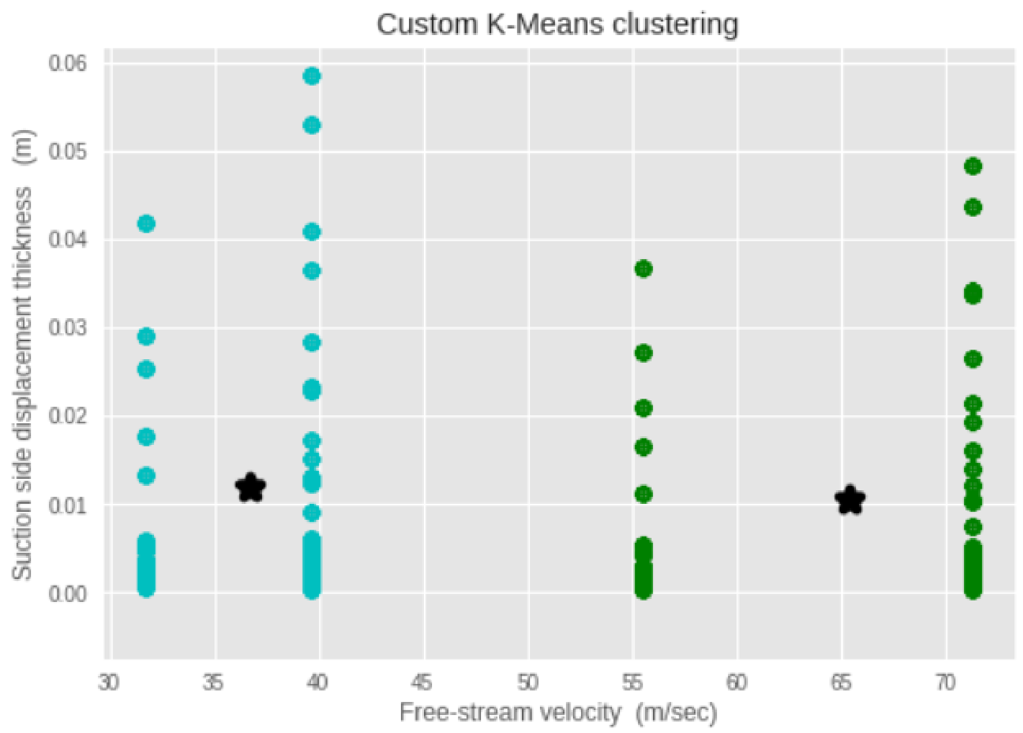


Figure 2.1 – The plot shows the clustered data points for the Airfoil Self-Noise Data set, using the Implemented K-Means by Scikit-learn deploying 2 attributes: Free-stream velocity and suction side displacement thickness.

For the same data set, we use custom batch K-Means. In Figure 2. we show the clustering results for the same data set.

The custom algorithm for batch K-Means, receives 3 hyper-parameters: (a) the number of desired clusters k, (b) the tolerance (tol) which shows us how much a specific centroid moves in every iteration of the algorithm and (c) the number of max. iterations of the algorithm (max_iter).



```
Cluster 1 has: 742 data points
Cluster 2 has: 761 data points
```

Figure 2.2 –The plot shows the clustered data points for the Airfoil Self-Noise Data set, using the batch K-Means algorithm (https://www.youtube.com/watch?v=HRoeYblYhkg) deploying 2 attributes: Free-stream velocity and suction side displacement thickness).

The algorithm terminates, when one of the following conditions is first met:

i) The distance between centroid A in $(i-1)$-th iteration and the same centroid A in i-th iteration is smaller than the tolerance threshold, meaning that our algorithm converges.

ii) The maximum limit of iterations is met. This case is rare, since K-means algorithm converges fast [7].

As a next step, we implement the RPCL version of online K-Means algorithm, which receives only 1 hyper-parameter, the number of desired clusters k. The algorithm works well, removing any redundant initial seeds/centroids. We notice, that the algorithm's clustering results depend on the choice of the initial centroids.

In Figure 2.3 we present the clustered data with Scikit-learn's K-Means for $k=6$.

In Figure 2.4 we present the same data, clustered with the RPCL version of online K-Means for $k=6$.

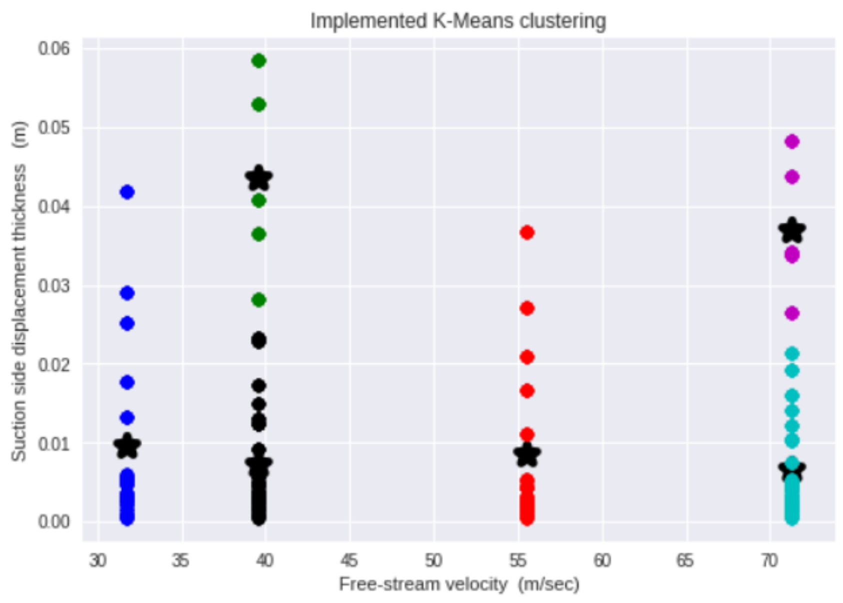
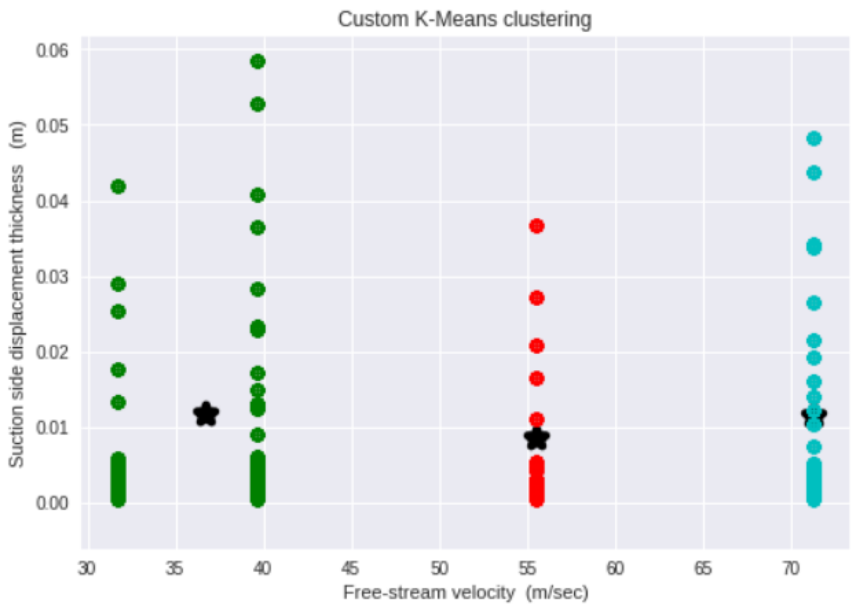


Figure 2.3 – The plot shows the clustered data points for the Airfoil Self-Noise Data set, using the Implemented K-Means by Scikit-learn deploying 2 attributes: Free-stream velocity and suction side displacement thickness. Number of clusters chosen: k=6)



```
Cluster 1 has: 761 data points
Cluster 2 has: 465 data points
Cluster 3 has: 277 data points
Cluster 4 has: 0 data points
Cluster 5 has: 0 data points
Cluster 6 has: 0 data points
```

Figure 2.4 – The plot shows the clustered data points for the Airfoil Self-Noise Data set, using the RPCL version of online K-Means deploying 2 attributes: Free-stream velocity and suction side displacement thickness. Number of clusters chosen: k=6.

We notice that while the simple K-Means algorithm clusters the data set according to the k number of initial centroids, the RPCL version of online K-Means penalizes the redundant initial centroids moving them away from the data set. Thus, RPCL algorithm considers as optimal k value, $k = 3$.

2.4 Aims and Objectives

Before we proceed further into the implementation stage of our work, here we state our thesis aims and objectives.

Predicting accurately the answer to queries over large data sets is nowadays essential. With the introduction of Big Data in the market, companies strive to find ways of processing big amounts of data in cost and time efficient ways. Query prediction comes in hand with the above.

Our thesis aims to predict accurately queries issued over a data set and calculate the true answer to the given query, to measure the accuracy of our prediction. Our predictive model will not require access to the original dataset to locate the answer to a given query.

Below, we state the steps we followed in our thesis to achieve our goal:

1. Load and pre-process the data set from UCI repository.
2. Train our model:
   - i) We generate a set of random queries for a given data set and calculating their true answers. Thus, we create a new data set with query-answer pairs.
   - ii) We use the RPCL algorithm to cluster the new query-answer data set, by giving as input one query-answer pair at a time and updating the winner and 2nd winner centroids.
3. Test our model:
   - i) Here, we generate a smaller number of random queries and calculate their true answers, which will be used to evaluate the predicted results.
   - ii) Predict their answers. This is achieved by using the Rival Penalizing Competitive Learning version of online K-Means algorithm. Here, we cluster the issued queries based on the Euclidean distance of the queries and the query part of the query-answer pairs that we created in Step 2.
4. Evaluate our model by calculating the RMSE, NRMSE and MAE values between the true and predicted answers for our query data set.

We train and test our model using the 2 data sets from UCI repository, which we mentioned in section 2.2. For each data set we use different, random query generation models, including Uniform, Normal and Multimodal Gaussian distributions, to compare and comment on the findings.

In Chapters 3 and 4, we will analyze and describe every step of our model providing screenshots, graphs and tables with results, when needed.

Chapter 3

Design and Implementation

3.1 The RPCL Algorithm

Rival Penalizing Competitive Learning algorithm is the clustering algorithm which we will be using for the training and prediction phase of our model. Here we present the implementation details of the algorithm.

Using Python 2.7, we created the Class RPCL(), which receives as hyper-parameter the number of desired clusters, k.

The Class includes 2 functions: i) *fit(data)*, ii) *predict(data)*

The *fit()* function receives as input the data set that we would like to cluster. Given k desired clusters, the k initial seeds/centroids are then chosen randomly from the data points of the input data set. The reasoning behind our decision in this case, is that the algorithm depends heavily on the choice of the initial centroids. We run the algorithm multiple times. Each time, we chose different initial centroids. If the randomly chosen initial centroids did not match a query-answer pair of the dataset, the algorithm could behave in an unexpected way, not giving the desired clustering results.

According to L. Xu et.al [3], the RPCL algorithm can not only select automatically the appropriate number of units/seeds to represent an input data set, but in some cases it can also speed-up the learning process, pushing the third centroid $\bar{w_3}$ towards the correct cluster, as we can see in Figure 3.1.

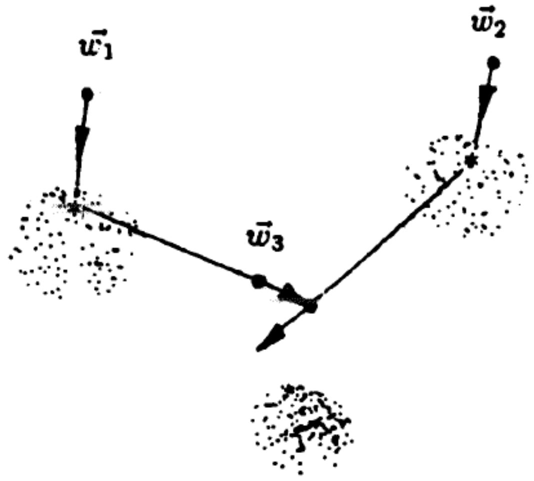


Figure 3.1 – RPCL algorithm: Shifting of the initial centroids ($\bar{w_1}, \bar{w_2}, \bar{w_3}$) towards their respective clusters. Source: L. Xu and A. Krzyzak. *Rival Penalized Competitive Learning for Clustering Analysis, RBF Net, and Curve Detection.* IEEE Transactions on Neural Networks, 1993.

Yet, there are specific cases where the initial seeds will not be assigned to the desired clusters. In Figure 2., the third initial seed/centroid is moving away from the clusters, instead of moving towards the desired cluster.

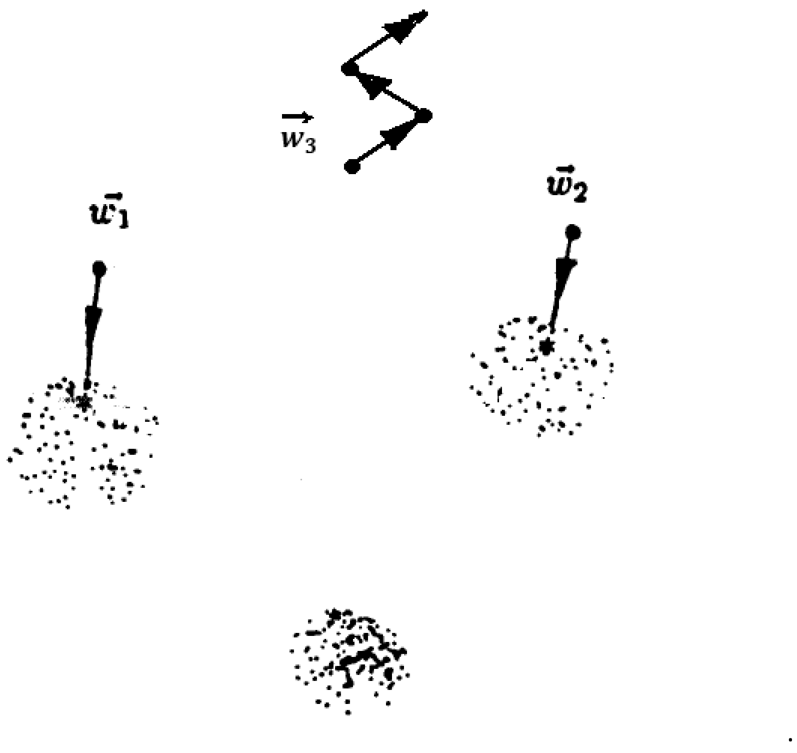


Figure 3.2 – RPCL algorithm: Shifting of the initial centroids ($\bar{w_1}, \bar{w_2}, \bar{w_3}$) towards their respective clusters. As we can see the initial assignment of centroids may affect the clustering results. In particular $\bar{w_3}$ is pushed away from the clusters.

Another parameter affecting the algorithm is the sequence in which each data point is fed into the algorithm – it can be sequential, for example in incremental order, or random. In our case, we chose to feed data points into the *fit()* function of our algorithm sequentially, in incremental order.

Next, in the *fit()* function for each data point given as input, we calculate the Euclidean distances between the data point and each one of the k centroids and find the 1st winner-centroid (smallest distance) and 2nd winner-centroid (second smallest distance). The 1st winner is then shifted towards the data point with a step which is constant for every iteration and the 2nd winner is shifted away from the data point with a smaller constant step in such a way that penalizes the competitor-centroid. After every data point of the data set has been classified, we choose to re-calculate the final centroid for the m-th cluster, where $1 \leq m \leq k$, if the m-th cluster consists of more than 10 data points. We decided to use this feature, as a way of awarding the valid clusters formed after the *fit()* function is executed.

The *predict()* function of our algorithm receives as input data points and assigns them to the corresponding cluster. This function was used as part of our experiment trials while testing the functionality of our algorithm. It is not used in our model, but is included for completeness.

3.2 System architecture major design decisions

3.2.1 Training the Model

Given a data set with $\{i1, i2, \ldots, in\}$ attributes and $\{j1, j2, \ldots, jk\}$ responses, we select a subset of 2 attributes and 1 response. To process our dataset, we perform sample selection from 2 distributions for each of the selected attributes that we will use to generate the set of random queries.

To be able to predict the real answer y for our randomly generated queries, we introduce a third attribute z, the radius of the circle, within which we will select and calculate the average value of the y response-values from the data points belonging to our true data set. That is, given a randomly generated query:

$Q = [q1, q2, z]$,

where: $q1$ is the $1st$ input dimension (attribute),

$q2$ is the $2nd$ input dimension (attribute),

and z is the circle radius

the query will have a center $C(q1, q2)$ and radius z. The datapoints D, satisfying the condition:

$$\left\| D - Q \right\|_2 \leq |z| \qquad (1)$$

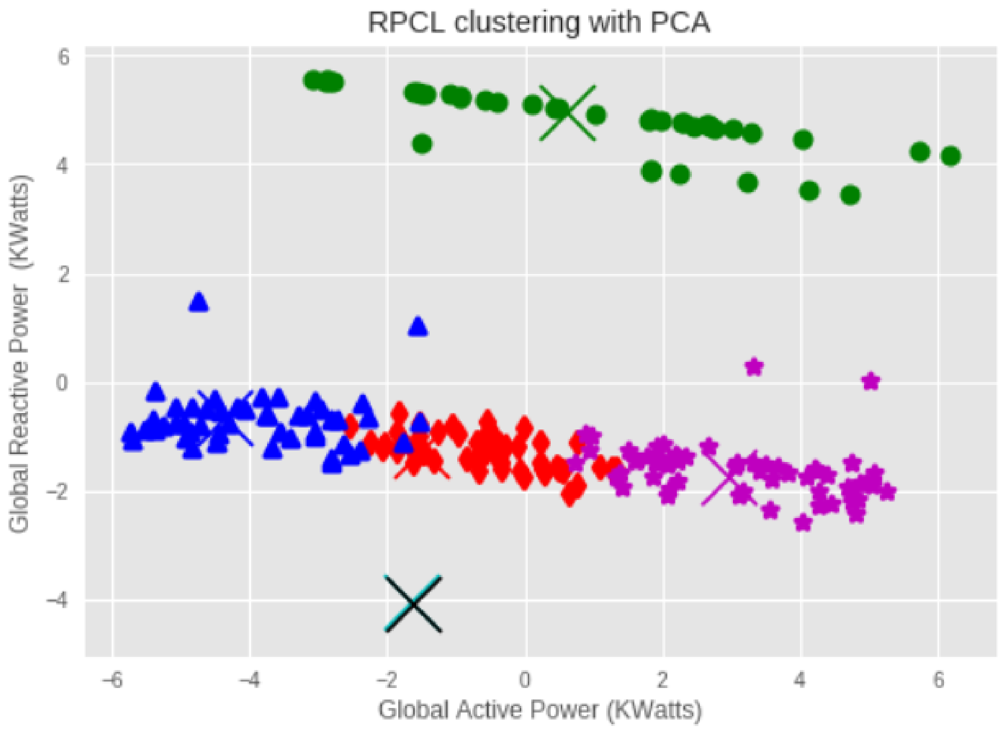
belong within the circle with center $Q(q1, q2)$ and radius z. In fact, we measure the similarity between 2 vectors, the query and the incoming data point from the downloaded UCI data set. Their in between distance is the Euclidean distance. Next, we calculate the average response value of all the incoming data points satisfying condition (1), thus finding the real response value y for our query Q.

Now, we have a new data set of N generated queries with their real answers (responses) y. The query-answer data set $[q1, q2, z, y]$ will then be given as an input to the Rival Penalizing Competitive Learning Algorithm, to cluster the data points and observe the results.

The clusters formed by the RPCL algorithm belong to a $4 - dim$ space. For visualization we use 2 techniques:

1. Principal Component Analysis (PCA)
2. t-Distributed Stochastic Neighbor Embedding (t-SNE)

Principal Component Analysis uses eigenvalues and eigenvectors, the powerhouse of vectorization, to perform linear dimensionality reduction [8]. It is a way of identifying patterns in data and expressing the data in such a way as to highlight their similarities and differences. [15] We use Skikit-learn's implemented PCA and set the parameter n_components = 2, to project our $4-dim$ data points into a $2-dim$ space. Then, we cluster the new $2-dim$ formed dataset using the RPCL algorithm. According to Figure 1, the optimal number of clusters for the generated query-answer data set is $k=4$.

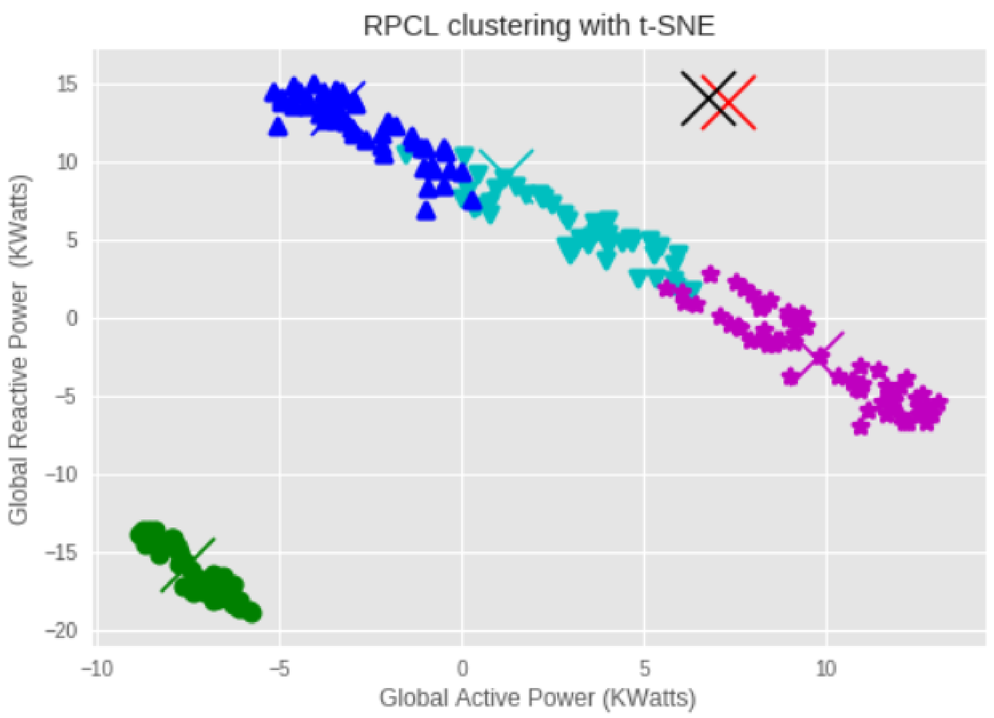


```
Cluster 1 has: 40 data points
Cluster 2 has: 0 data points
Cluster 3 has: 50 data points
Cluster 4 has: 49 data points
Cluster 5 has: 0 data points
Cluster 6 has: 58 data points
```

Figure 3.3 – RPCL clustering for a dataset with $n > 2$ attributes on which we have performed Principal Component Analysis. Number of initial centroids assigned $k=6$. We notice that algorithm has pushed away the unnecessary centroids, creating 4 clusters.

Another technique of visualizing in a high-dimensional space is t-SNE. The technique is a variation of Stochastic Neighbor Embedding and is better than the existing techniques at creating a single map that reveals structure at many, different scales. [16] Similarly, we use Skikit-learn's implemented t-SNE and set the parameters n components = 2 and random_state = 0, which is the seed used by the random number generator [17]. In Figure 2, we observe the shaped clusters for the same query-answer dataset, with initial $k=6$.

In both Figure 1 and Figure 2 we notice that any redundant initial seeds (centroids) are moved away from the clusters. Thus, the optimal number of clusters for the given data set is $k=4$.



```
Cluster 1 has: 40 data points
Cluster 2 has: 46 data points
Cluster 3 has: 0 data points
Cluster 4 has: 47 data points
Cluster 5 has: 0 data points
Cluster 6 has: 64 data points
```

Figure 3.4 – RPCL clustering for a dataset with $n > 2$ attributes on which we have performed t-distributed stochastic neighbor embedding. Number of initial centroids assigned $k = 6$. We notice that algorithm has pushed away the unnecessary centroids, creating 4 clusters.)

3.2.2 Predicting the Answer

For the prediction phase of our model, we first generate a smaller number of queries $Q = [q1, q2, z]$, similarly to the training phase. Now, we have a set of k queries Q. First, we find the true answer to each one of the k queries, by calculating the Euclidean distances between the query and the data point vectors, using equation (1). To calculate the true response values (answers) y for our new queries, we use as data points the query-answer pairs from the dataset that we generated in the training phase. In other words, we calculate the true answer of a query using as ground truth, the randomly generated data set of query-answer pairs as described in Section 3.1.2.

Having calculated the real answer y for our new queries, we now have a dataset of query-answer pairs. We cluster the data points from the dataset with N answer-pairs that was described in section 3.1.2, using the RPCL algorithm. After the data points have been clustered, we predict the cluster to which a given query belongs. That is, we calculate the Euclidean distances between the query-vector and the query part $[q1, q2, z]$ of the centroid-vectors of each cluster. The query is then assigned to the cluster, for which their in between Euclidean distance is the smallest.

The predicted answer y' to our query is the response value y for the centroid of the cluster in which the query is assigned to.

Having extracted both the true answer y and the predicted answer y', we can use this information to calculate the error and see how accurate our model is. This will be described thoroughly in the next Chapter.

Chapter 4

Evaluation

In this Chapter we present the metrics that we used to evaluate our model and the strategy that we followed to conduct software testing and evaluation. Lastly, we provide the evaluation measurements – tables and graphs are included, to show how our model predicts the answer to a given query in different cases, which we will discuss in detail.

4.1 Evaluation Metrics

To evaluate our model, we used three popular metrics applied in predictive modeling evaluation:

i) *RMSE* (Root Means Squared Error) – A quadratic scoring rule that also measures the average magnitude of the error. It calculates the average of squared differences between predicted values and actual observations (true values). RMSE is described by the following mathematic equation:

$$RMSE = \frac{1}{n}\sqrt{\sum_{j=1}^{n}(y_j - y'_j)^2} \qquad (2)$$

where y_j are the actual observations, y'_j are the predicted values and n is the number of observations.

ii) *NRMSE (Normalized Root Means Squared Error)* – Normalized RMSE is used to compare datasets or models with different scales. It expresses a percentage, with lower values expressing less residual variance. In our case the denominator of the following equation is the Variance of our output values. NRMSE is described by the following mathematic equation:

$$NRMSE = \frac{1}{n}\sqrt{\frac{\sum_{k=0}^{n}(y_j - y'_j)^2}{\sum_{k=0}^{n}(y_j - \bar{y}_j)^2}} \qquad (3)$$

where y_j are the actual observations, y'_j are the predicted values, $\bar{y}_j$ is the average value of the actual observations and n is the number of observations.

iii) *MAE (Mean Absolute Error)* – A measure of difference between 2 continuous variables. It measures the average error over a test sample of the absolute differences between predicted values and actual observations (true values). MAE is described by the following mathematic equation:

$$MAE = \frac{1}{n}\sum_{j=1}^{n} |y_j - y'_j| \qquad (4)$$

where y_j are the actual observations, y'_j are the predicted values and n is the number of observations.

RMSE is preferred in measuring a model's prediction error when large errors are undesirable. According to equation (2) taking the square root of the averaged squared errors, results in assigning a relatively high weight to large errors. Also, RMSE can be problematic when comparing results on test samples with different sizes [18]. In our work, the test samples for each dataset have equal size. We use both RMSE-NRMSE and MAE measurements for completeness.

4.2 Evaluation Strategy

To evaluate the predictive model, we implemented 5 different query generation strategies, which are used both in the training and testing phase and will help us reason about the impact of the query space onto the prediction accuracy of our model. The strategies are as follows:

1. We generate random queries using the Uniform distribution. In particular, we estimate the X_1 and X_2 attributes of a query using 2 Uniform distributions, that is $X_1 \sim$ U(min_1, max_1) and $X_2 \sim$ U(min_2, max_2), where min_1, max_1 are the minimum and maximum values for X_1 attribute and min_2, max_2 are the minimum and maximum values for X_2 attribute respectively, according to the downloaded dataset. To calculate the true answer y for a given query we use the radius of a circle. That is, a relatively small fixed value z, with $z = 5\%$ of the domain range.

2. We generate random queries using the Uniform distribution identically to scenario 1. To calculate the true answer y for a given query we use the radius of a circle, which belongs to a Uniform distribution, that is $z \sim$ U(min_3, max_3), where min_3 is the 1% of the domain range and max_3 is the 5% of the domain range. That is, all 3 query dimensions are now random variables.

3. We generate random queries using the Gaussian (Normal) distribution. In particular, we produce the X_1 and X_2 attributes of a query using 2 Gaussian distributions, that is $X_1 \sim$ N($\mu_1, \sigma_1{}^2$) and $X_2 \sim$ N($\mu_2, \sigma_2{}^2$) where μ_1, μ_2 are the mean values and $\sigma_1{}^2$, $\sigma_2{}^2$ are the standard deviation values. We set $\mu_1 = \mu_2 = 0$, and $\sigma_1 = (max_1 - min_1)/2$ and $\sigma_2 = (max_2 - min_2)/2$ respectively. Radius z, is a relatively small fixed value, with $z = 5\%$ of the domain range.

4. We generate random queries using the Gaussian distribution identically to scenario 3. To calculate the true answer y for a given query we use the radius of a circle, which belongs to a Gaussian distribution, that is $z \sim$ N($\mu_3, \sigma_3{}^2$) , where μ_3 is the 5%

of the domain range and σ_3 is the 2% of the domain range. That is, all 3 query dimensions are now random variables.

5. We generate random queries using the multi-modal Gaussian distribution. That is, we generate the X_1 attribute from 2 different Gaussian distributions N(μ_{11}, σ_1^2) and N(μ_{12}, σ_1^2) and the X_2 attribute from 2 other Gaussian distributions N(μ_{21}, σ_2^2) and N(μ_{22}, σ_2^2), where the means and standard deviation values are randomly chosen from a uniform distribution similar to how we generated queries in scenarios 1 and 2. To calculate the true answer y for a given query we use the radius of a circle. That is, a relatively small fixed value z, with $z = 5\%$ of the domain range.

We conduct our model testing and evaluation over 2 datasets from the UCI repository, which were described in Chapter 2 [9], [10]. We apply the query generation strategies for each of the datasets.
For the "Individual household electric power consumption" dataset, we selected 2 out of the 6 available attributes and 1 out of the 3 responses. The selected attributes are 'Global active Power' and 'Global reactive Power'. Since, the selected attributes have the same unit of measurement and their span is relatively similar ($min_1 \approx min_2$, $max_1 \approx max_2$) we applied the query generation techniques directly to our dataset.

For the second case, that is the "Airfoil Self-Noise" dataset, we selected 2 out of the 5 available attributes and the 1 response that was available. The attributes which we selected, are 'frequency' in Hertz and 'Angle of attack' in degrees. Hence, we selected attributes with different unit of measurement and different span, as we defined it previously. In this case, after applying the query generation techniques, we normalized the generated set of queries to span between 0 and 1.

4.3 Evaluation Results

Having completed the training and testing phase for the "Individual household electric power consumption" dataset, in Table 1. we provide the RMSE, NRMSE and MAE values.

	Uniform with fixed z	Uniform with moving z	Gaussian with fixed z	Gaussian with moving z	Multi-modal Gaussian with fixed z
RMSE	0.594437	0.844352	1.683784	1.449071	0.471477
NRMSE	2.010766	1.353671	1.230532	9.324725	0.787330
MAE	1.274541	2.173862	5.180275	4.103660	0.883175

Table 4.1 – RMSE, NRMSE, MAE values for 5 query generation strategies for "Individual household electric power consumption" dataset.

At this point, we should note that the metrics reflect the error between the true and the predicted responses. The latter were calculated for k number of clusters, where k was the optimal number of clusters selected by the RPCL algorithm during the Training phase of our model. The values in Table 1. change every time we run the testing part of our model, since the queries that are generated for testing are random, therefore giving different results every time. We should iterate the model many times to see how the RMSE, NRMSE and MAE values converge, which we do in the next dataset.

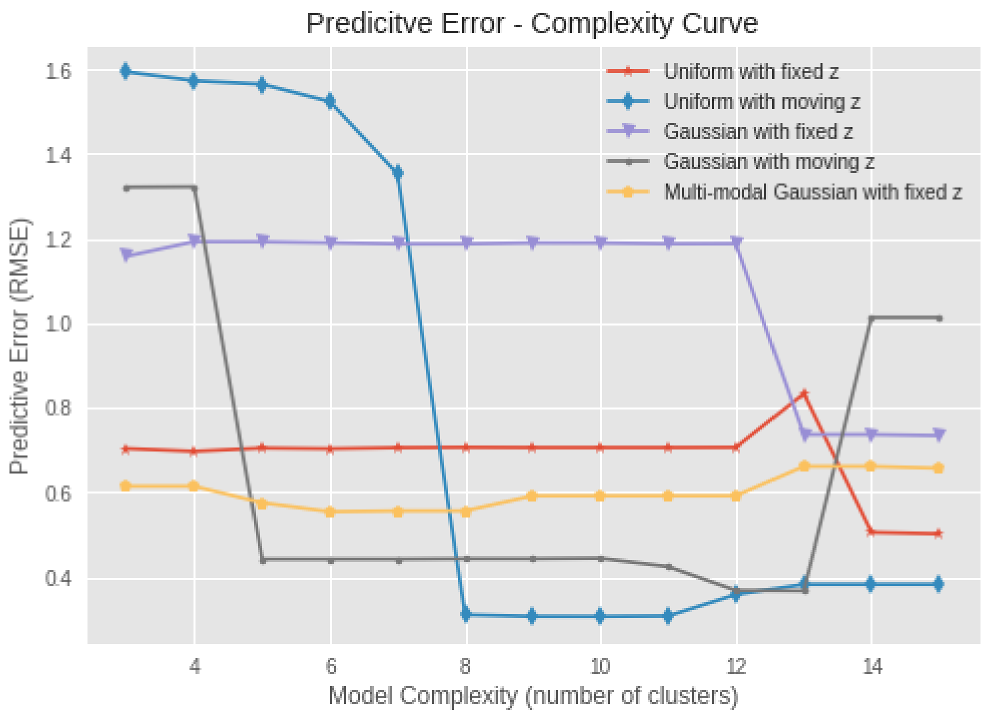


Figure 4.1 – The plot shows the predictive error (RMSE) for different number of clusters for the RPCL algorithm, that is the model's complexity. There are 5 curves, 1 for each query generation strategy, for "Individual household electric power consumption" dataset.

In Figure 4.1 we show the Predictive Error – Model Complexity curves. That is, the respective RMSE values for different number of K clusters formed by the RPCL (predictive) algorithm, for each one of the 5 different query generating scenarios.

Regarding "Airfoil Self-Noise" dataset, in Table 2. we provide the RMSE, NRMSE and MAE values for the optimal k number of clusters as selected by the RPCL algorithm.

	Uniform with fixed z	Uniform with moving z	Gaussian with fixed z	Gaussian with moving z	Multi-modal Gaussian with fixed z
RMSE	0.493635	0.495313	0.537719	0.529189	0.139419
NRMSE	2.853363	5.352731	2.248283	3.167619	7.159423
MAE	4.047996	3.937969	4.269954	4.012563	3.515772

Table 4.2 – RMSE, NRMSE, MAE values for 5 query generation strategies for "Airfoil Self-Noise" dataset.

Due to the significantly smaller size of the second dataset, we were able to create larger training and testing samples. While for the for the "Individual household electric power consumption" dataset, we used a set of 200 query-answer pairs and a testing set of 20 queries, for the “Airfoil Self-Noise” dataset we used a training set of 1,000 query-answer pairs and 100 queries subsequent to testing, respectively.

Thus, with respect to the second dataset, since the computational power required to execute the model was smaller, we additionally calculated the average values for the RMSE metric over different k number of clusters, iteratively. In other words, we executed, repeatedly, our model (training-testing phases) 100 times, to observe how the model complexity – predictive error curve converges and reason over the results. In Figure 2, we observe the approximated curves for the 5 different query generating scenarios, over the “Airfoil Self-Noise” dataset.

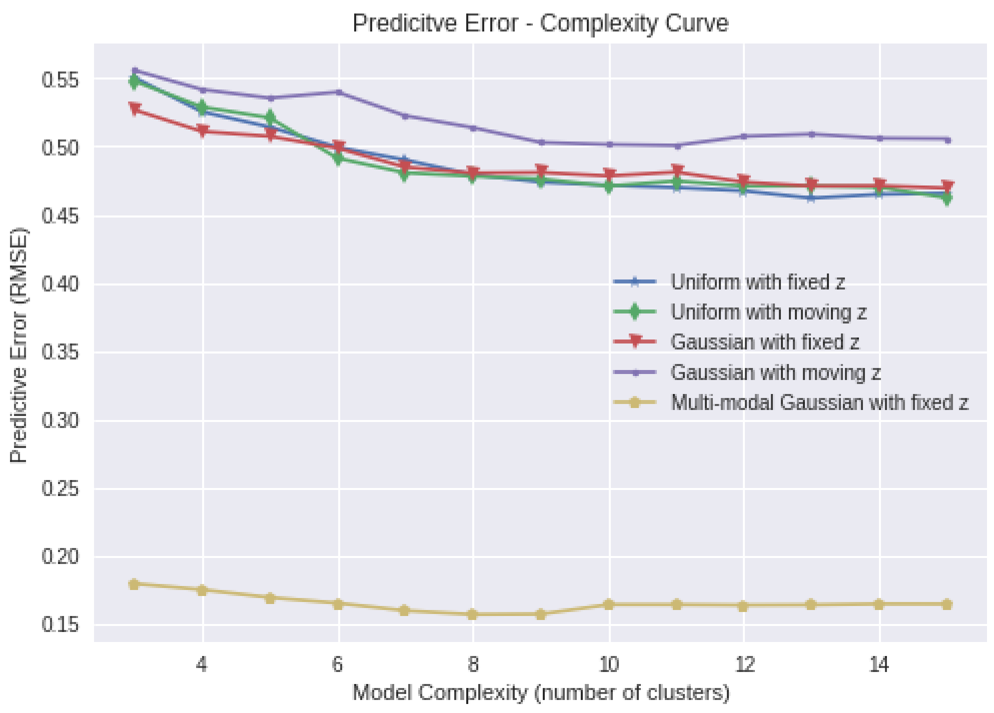


Figure 4.2 – The plot shows the predictive error (RMSE) for different number of clusters for the RPCL algorithm, that is the model’s complexity. There are 5 curves, 1 for each query generation strategy, for “Airfoil Self-Noise” dataset. The curves show the average RMSE after 100 iterations of our model.

Chapter 5

Conclusion

5.1 Discussion of Evaluation Results

Our model predicts the answer to given queries using the RPCL algorithm. We could say that the algorithm resembles the online version of K-Means algorithm with a competitive learning functionality, which penalizes the 2nd winner (rival) centroids and awards the first winners. Since the algorithm that was used for the answer prediction is not a batch algorithm, a large number of tests has to be run, to get a better understanding of how the error between the true and predicted answer converges, with respect to the number of K clusters formed and the query generation scenario used.

For the "Airfoil Self-Noise" dataset we run the model 100 times and the Prediction Error – Model Complexity curves shown in Figure 2. are the results of this experiment.

We notice that the smallest average RMSE values are obtained in the case where our model uses the Multi-modal Gaussian distribution to generate random queries with a fixed z value. That is, our query space is defined by 2 Gaussian distributions forming a 2-dim space, denoted s, where $s \epsilon R^2 \subseteq [min_1, max_1] x [min_2, max_2]$. $Min_{1,2}$ and $max_{1,2}$ are the smallest and largest values respectively for the 2 attributes that we used from the initial datasets X_1 and X_2. In other words, our model has to learn from a significantly smaller query space in comparison with learning from a query space formed by 2 Uniform distributions. In addition the circle's radius z, that is used to calculate the true answer to our queries, is fixed improving the learning process. Regarding the query generation techniques, where a moving z value was used, we actually add a 3rd attribute to our predictive model, increasing the query's space to a 3-dim space, that is $s \epsilon R^3$.

From the remaining 4 query generation techniques, the Gaussian distribution with a moving z value produces the biggest predictive error. Unlike using a fixed z value, in this case the z parameter is generated randomly from a Gaussian distribution, meaning that each query now has 3 random attributes $q = [X_1, X_2, z]$.

5.2 Occam's Razor and Over-fitting

When developing a predictive model it is important to understand the principles of model complexity [19]. Occam's Razor states that *"All else being equal, simpler models should be favored over more complex ones."* If we could think of model complexity and the rate at which the error decreases as two plotted lines which are dependent to each other, then we could say that the most favorable model can be found at the point where the error's decrease rate remains stable or decreases, as the complexity continues to grow. This idea, is applied in the elbow-method of K-Means clustering algorithm, to choose the optimal K number of clusters.

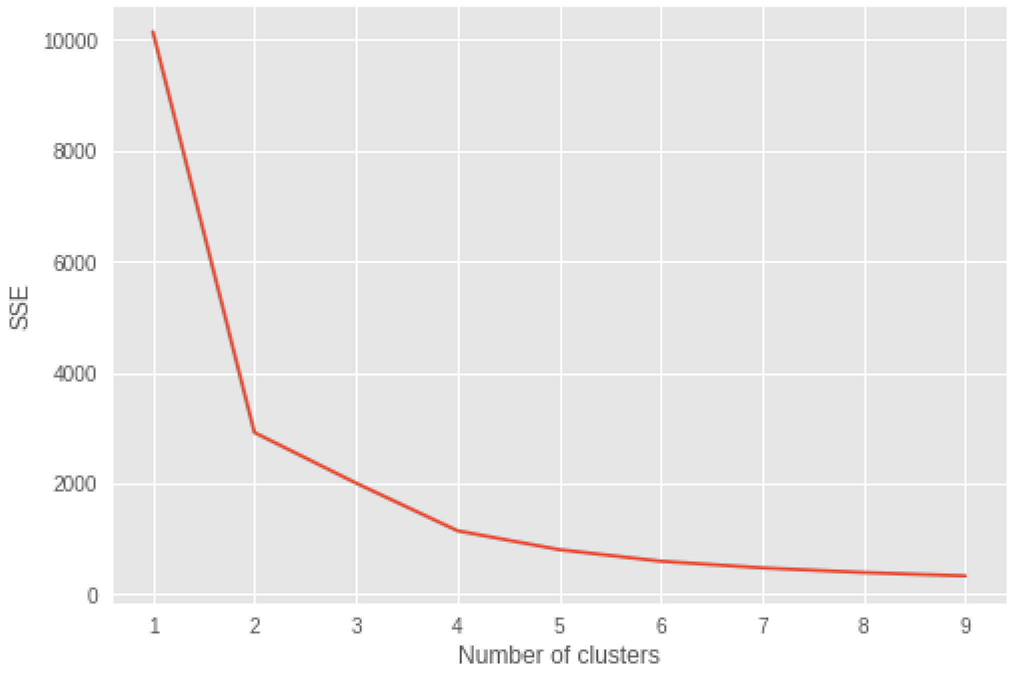


Figure 5.1 – The plot shows how the SSE (Sum of Squared Errors, that is distances of the clustered data points from their respective centroids) changes according to the number of clusters for K-Means algorithm.

As we see in Figure 3. the point at which the error decrease rate is stabilized, gives us the optimal K value for the algorithm, which for this case is $k = 4$.

For the implemented version of the RPCL algorithm that we used in our model, we attempt to find the optimal number of k clusters without using any supplementary methods such as the Elbow method.

So far, we mentioned that as the model complexity increases, the error decreases and we showed how the methodology of choosing the optimal k is correlated with Occam's Razor theorem. What happens if we continue increasing the model's complexity, that is increasing the number of clusters up to the point where $k = n$, where n is the number of data points?

An over-fitted model is a model that reflects the errors in the training dataset, instead of accurately predicting unseen data. The problem lies in the fact that the model has effectively memorized existing data points instead of trying to predict how unseen data points should be [20].

In our case, the more we increased the complexity of our model, after a specific threshold the predictive error begun to increase, which explains the over-fitting phenomenon. This can be clearly observed, in Figure 4. where we iteratively run the model for the "Airfoil Self-Noise" dataset.

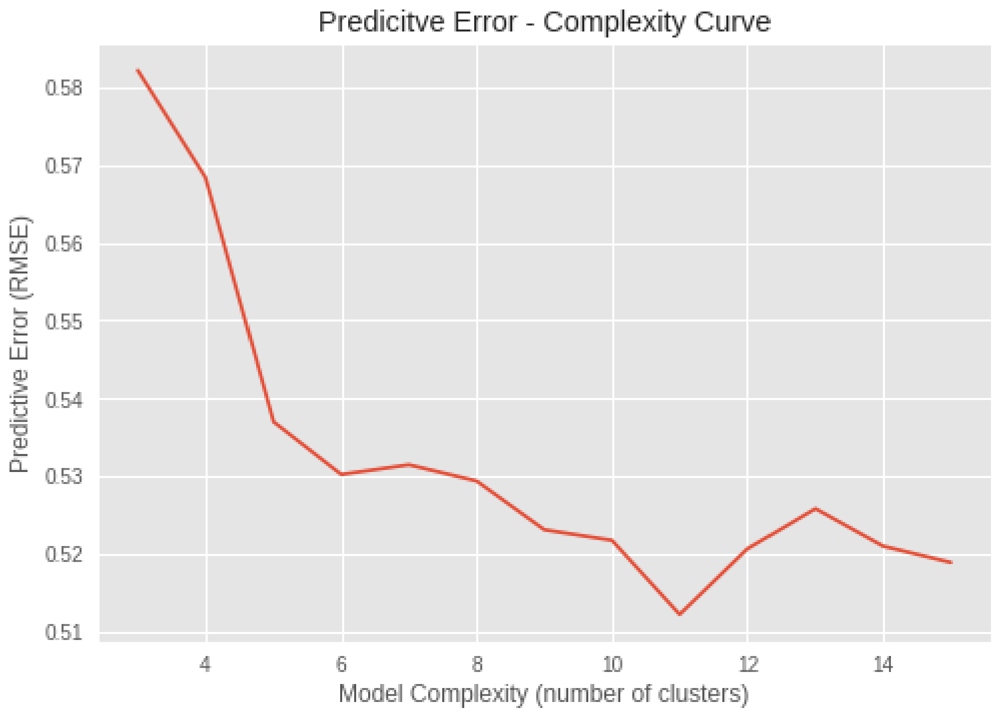


Figure 5.2 – The plot shows the predictive error (RMSE) with respect to the model's complexity (number of clusters) after 100 runs of the model (training and testing phase) for the "Airfoil Self-Noise" dataset using the Gaussian distribution with moving z for the query generation. For K>11 we notice that the error increases.

5.3 Summary

We trained and tested a predictive model over 2 datasets downloaded from the UCI repository. Our model is based on 2 algorithms – one for calculating the true answer to a given query from our training or testing set and another for predicting the answer to the test queries. A simple version of the RPCL algorithm (the conscience strategy was not included) was implemented, to cluster the generated datasets so that we could predict a given query by classifying it to the closest, in terms of Euclidean distance, cluster.

We run evaluation tests for our model using 2 datasets and 5 random query generation techniques for each one. In each scenario we provided the RMSE, NRMSE and MAE metrics. Moreover, for the "Airfoil Self-Noise" dataset we iteratively run the model and acquired the mean values of the 3 error metrics. Since our version of the RPCL algorithm is an online clustering algorithm, we had to execute the predictive model many times in order to converge and provide results that could be commented.

Overall, we notice that our model can predict the answer to a given query with a relatively low error. We consider that the answers assigned as true over the generated training dataset are close to the true values. For the datasets used, our model performs well for a rather small number of clusters and the predictive accuracy decreases as the number of clusters for the RPCL algorithm (model's complexity) is increased.

5.4 Future Work

Whilst our predictive model is capable of predicting the answer to a given query without requiring direct access to the RDBMS, we would like to explore further how we can improve our current predictive model by implementing different models and comparing the results with the measurements we provided in Chapter 4. Our suggestions for future work are the following:

- Change the RPCL algorithm in order to perform on any given initial centroids, as the current version can be sensitive to initial seeds' assignment.

- Explore how to improve the predictive functionality of our model, by implementing different competitive learning algorithms and compare the results with those that we achieved for the online versions of the RPCL algorithm. We would like to include the conscience strategy and implement the Dynamically Penalized Rival Competitive Learning Algorithm (DRPCL) [21] to see which algorithm can give the best results in terms of accuracy.

- Test batch-like algorithms, since the online versions can be sensitive to how a given dataset is fed into the algorithm and thus a large number of iterations is needed until convergence.

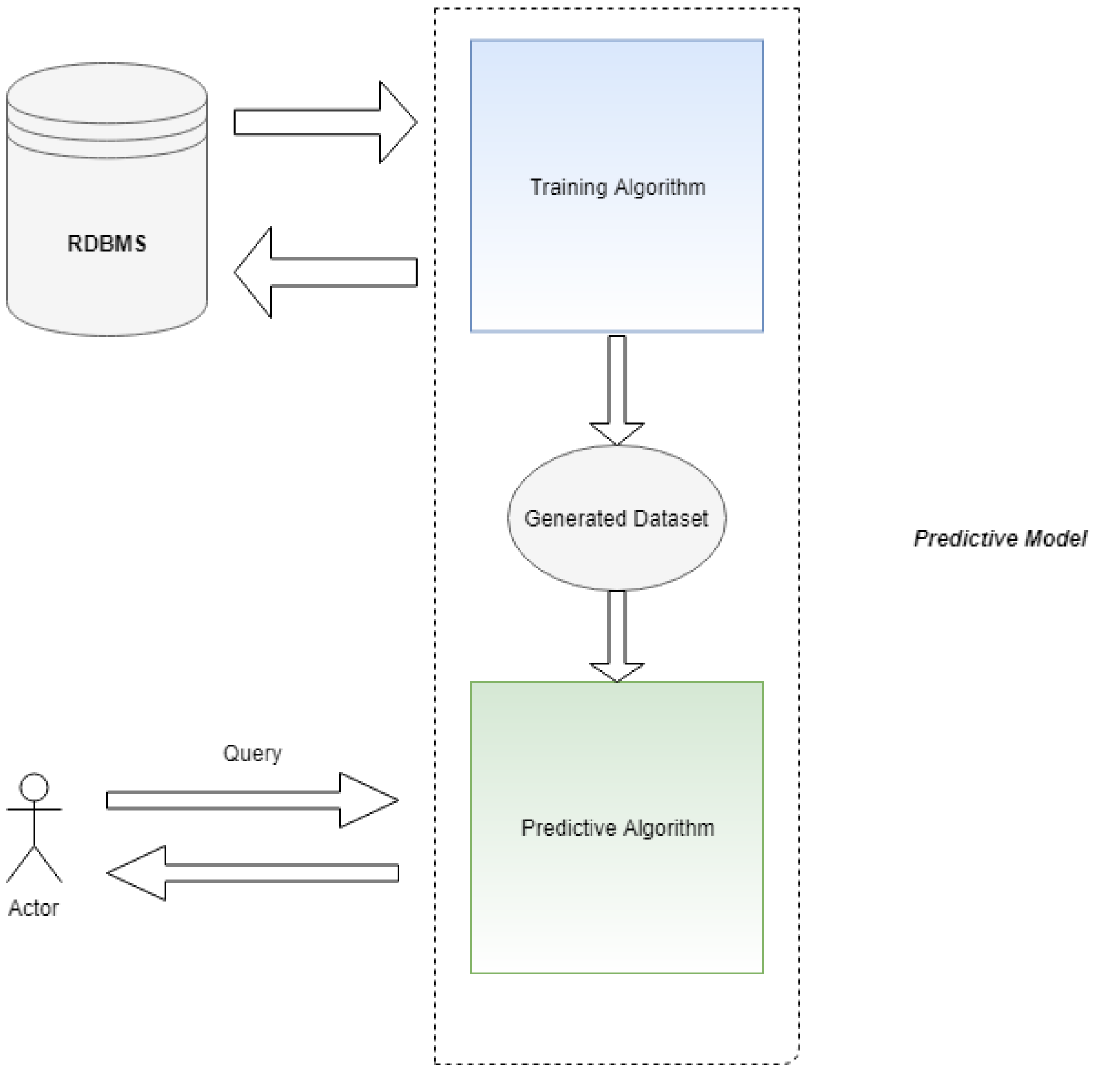
- For the purpose of our project, we used small datasets from the UCI repository. However, a real-scale predictive analysis model performs over large datasets. We would like to leverage the power of Hadoop distributed platform to perform a partition-based clustering technique on the initial dataset once, with respect to the training phase of our model [22].

- Implement a predictive model using Probabilistic clustering (e.g. EM, Mixtures of Gaussians, RBFs) and compare the results with the ones we could get by implementing vectorized clustering algorithms (e.g. K-Means, FSCL, RPCL).

Appendix A

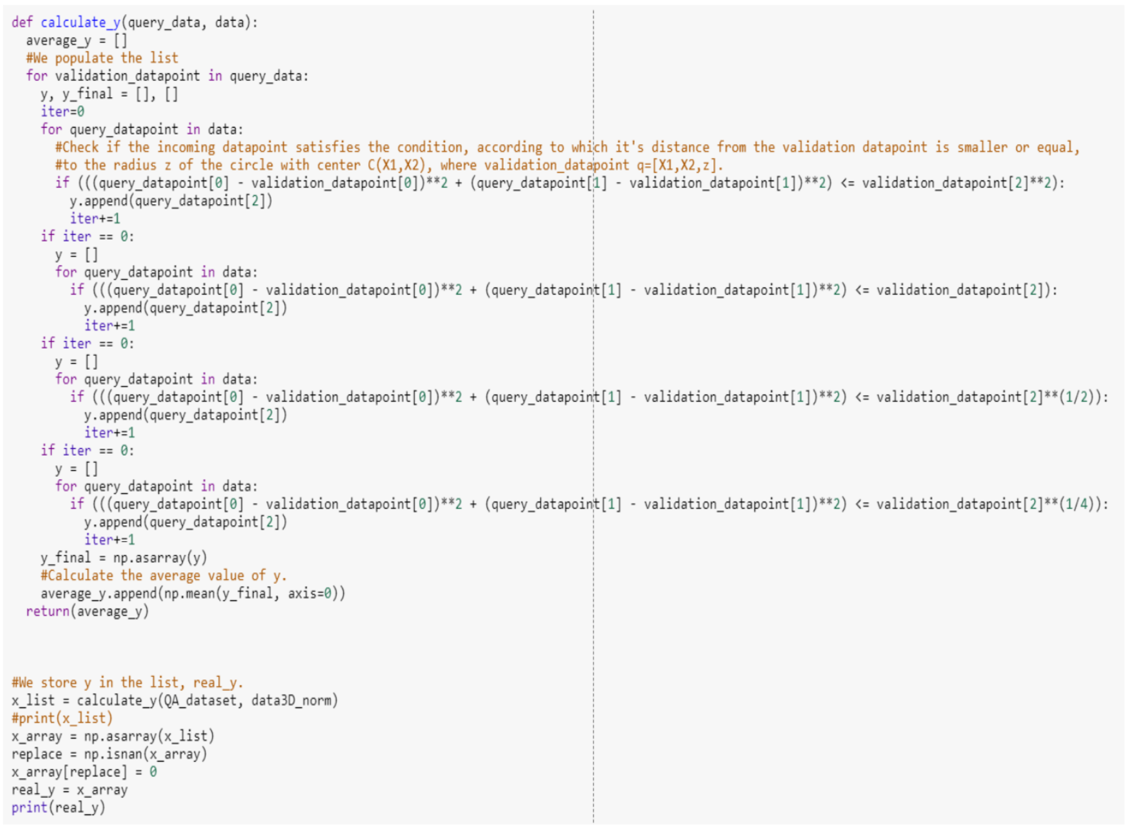
A.1

Flowchart presenting the steps followed throughout our Predictive Model's process. The model consists of 2 phases: Training and Predictive phase.



A.2

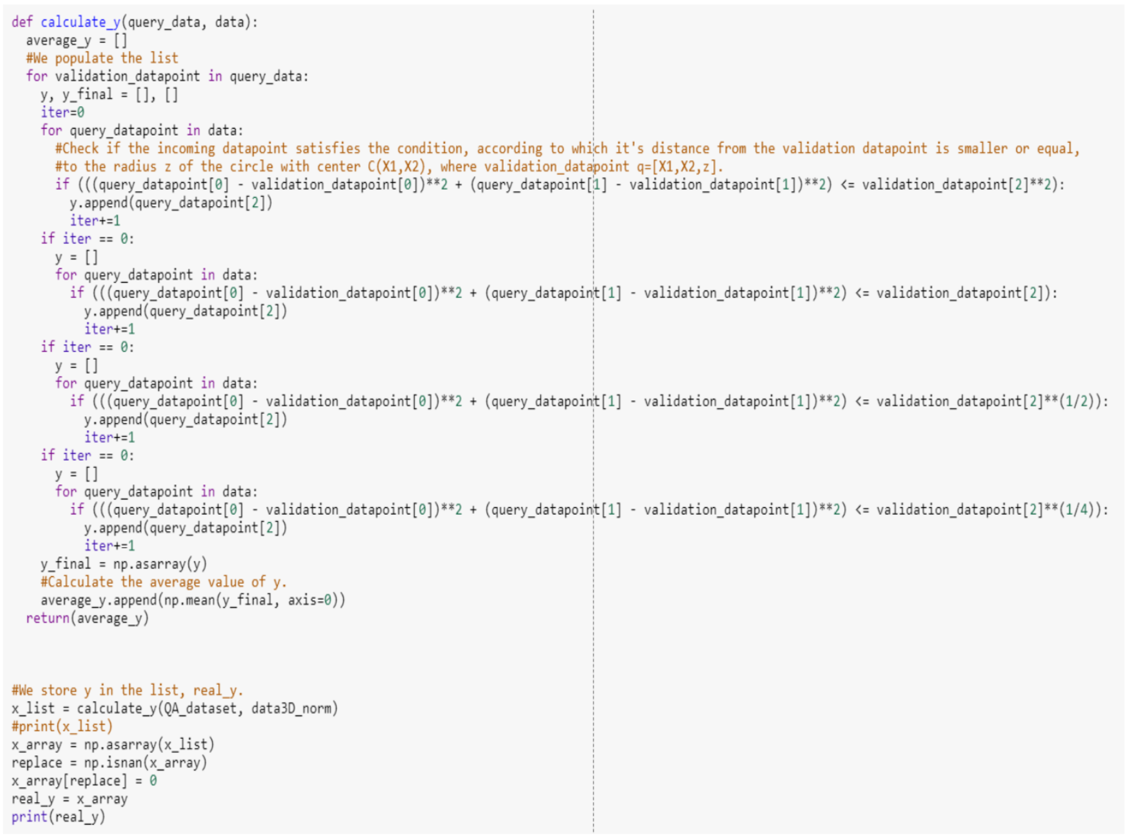
Snippet of code, which shows the calculate() function. It is the algorithm which calculates the true response to a given query.



```python
def calculate_y(query_data, data):
  average_y = []
  #We populate the list
  for validation_datapoint in query_data:
    y, y_final = [], []
    iter=0
    for query_datapoint in data:
      #Check if the incoming datapoint satisfies the condition, according to which it's distance from the validation datapoint is smaller or equal,
      #to the radius z of the circle with center C(X1,X2), where validation_datapoint q=[X1,X2,z].
      if (((query_datapoint[0] - validation_datapoint[0])**2 + (query_datapoint[1] - validation_datapoint[1])**2) <= validation_datapoint[2]**2):
        y.append(query_datapoint[2])
        iter+=1
    if iter == 0:
      y = []
      for query_datapoint in data:
        if (((query_datapoint[0] - validation_datapoint[0])**2 + (query_datapoint[1] - validation_datapoint[1])**2) <= validation_datapoint[2]):
          y.append(query_datapoint[2])
          iter+=1
    if iter == 0:
      y = []
      for query_datapoint in data:
        if (((query_datapoint[0] - validation_datapoint[0])**2 + (query_datapoint[1] - validation_datapoint[1])**2) <= validation_datapoint[2]**(1/2)):
          y.append(query_datapoint[2])
          iter+=1
    if iter == 0:
      y = []
      for query_datapoint in data:
        if (((query_datapoint[0] - validation_datapoint[0])**2 + (query_datapoint[1] - validation_datapoint[1])**2) <= validation_datapoint[2]**(1/4)):
          y.append(query_datapoint[2])
          iter+=1
    y_final = np.asarray(y)
    #Calculate the average value of y.
    average_y.append(np.mean(y_final, axis=0))
  return(average_y)



#We store y in the list, real_y.
x_list = calculate_y(QA_dataset, data3D_norm)
#print(x_list)
x_array = np.asarray(x_list)
replace = np.isnan(x_array)
x_array[replace] = 0
real_y = x_array
print(real_y)
```

A.3

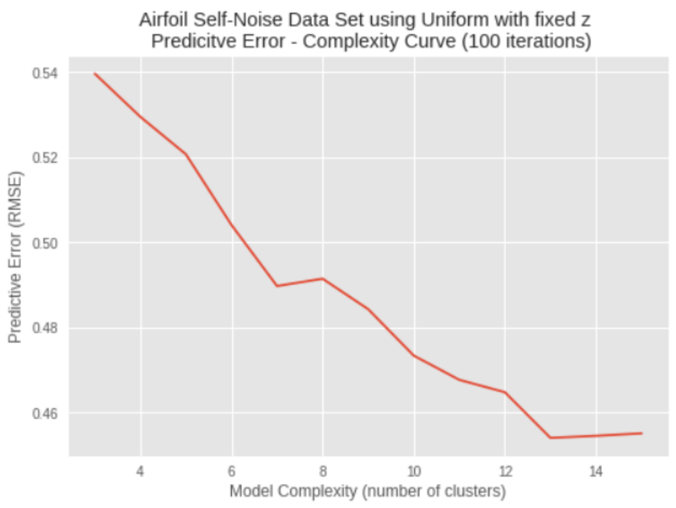
Snippet of code, that shows the RPCL() class which uses the fit() function. This is the algorithm which we use to cluster the dataset using the RPCL algorithm, We later use the clustered results of the algorithm to classify/assign any new queries to their corresponding cluster.

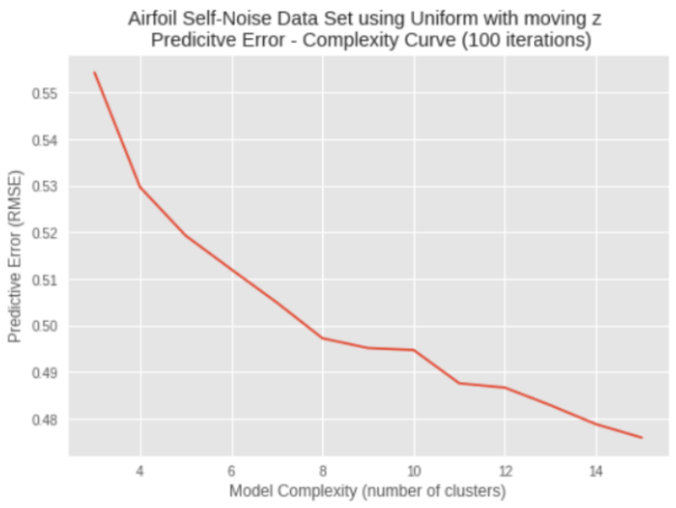


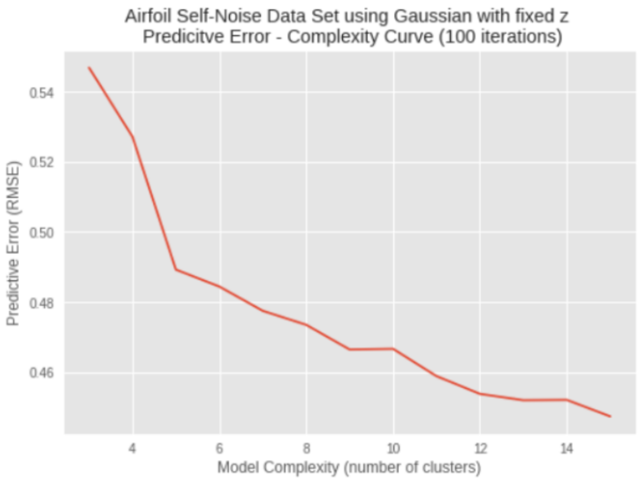
```python
def calculate_y(query_data, data):
  average_y = []
  #We populate the list
  for validation_datapoint in query_data:
    y, y_final = [], []
    iter=0
    for query_datapoint in data:
      #Check if the incoming datapoint satisfies the condition, according to which it's distance from the validation datapoint is smaller or equal,
      #to the radius z of the circle with center C(X1,X2), where validation_datapoint q=[X1,X2,z].
      if (((query_datapoint[0] - validation_datapoint[0])**2 + (query_datapoint[1] - validation_datapoint[1])**2) <= validation_datapoint[2]**2):
        y.append(query_datapoint[2])
        iter+=1
    if iter == 0:
      y = []
      for query_datapoint in data:
        if (((query_datapoint[0] - validation_datapoint[0])**2 + (query_datapoint[1] - validation_datapoint[1])**2) <= validation_datapoint[2]):
          y.append(query_datapoint[2])
          iter+=1
    if iter == 0:
      y = []
      for query_datapoint in data:
        if (((query_datapoint[0] - validation_datapoint[0])**2 + (query_datapoint[1] - validation_datapoint[1])**2) <= validation_datapoint[2]**(1/2)):
          y.append(query_datapoint[2])
          iter+=1
    if iter == 0:
      y = []
      for query_datapoint in data:
        if (((query_datapoint[0] - validation_datapoint[0])**2 + (query_datapoint[1] - validation_datapoint[1])**2) <= validation_datapoint[2]**(1/4)):
          y.append(query_datapoint[2])
          iter+=1
    y_final = np.asarray(y)
    #Calculate the average value of y.
    average_y.append(np.mean(y_final, axis=0))
  return(average_y)


#We store y in the list, real_y.
x_list = calculate_y(QA_dataset, data3D_norm)
#print(x_list)
x_array = np.asarray(x_list)
replace = np.isnan(x_array)
x_array[replace] = 0
real_y = x_array
print(real_y)
```

A.4

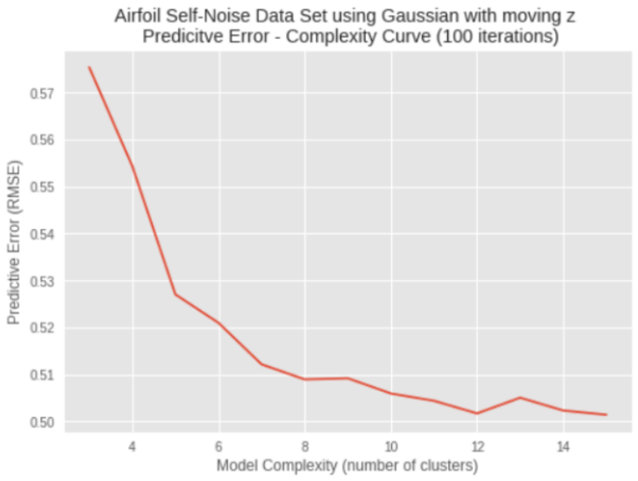
Predictive Error – Complexity Curves for the “Airfoil Self-Noise” Dataset using 5 query generation scenarios. The resulting curves are the representation of the RMSE values with respect to the corresponding number of clusters k, after 100 iterations of the predictive modeling process.



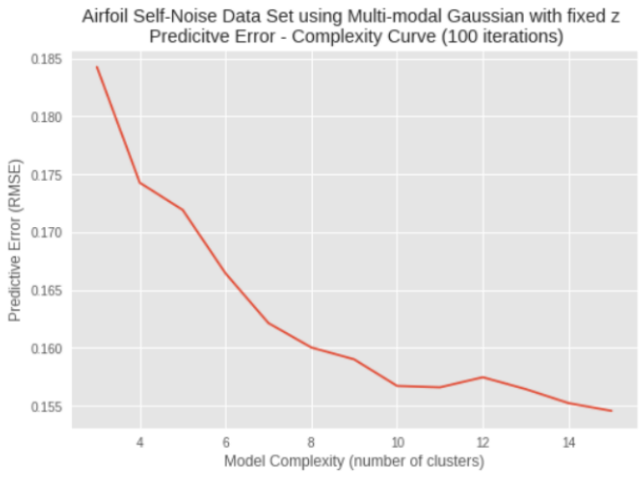




Airfoil Self-Noise Data Set using Gaussian with fixed z
Predicitve Error - Complexity Curve (100 iterations)
Predictive Error (RMSE)
0.54
0.52
0.50
0.48
0.46
4
6
8
10
12
14
Model Complexity (number of clusters)



Airfoil Self-Noise Data Set using Gaussian with moving z
Predicitve Error - Complexity Curve (100 iterations)
Predictive Error (RMSE)
0.57
0.56
0.55
0.54
0.53
0.52
0.51
0.50
4
6
8
10
12
14
Model Complexity (number of clusters)



Airfoil Self-Noise Data Set using Multi-modal Gaussian with fixed z
Predicitve Error - Complexity Curve (100 iterations)
Predictive Error (RMSE)
0.185
0.180
0.175
0.170
0.165
0.160
0.155
4
6
8
10
12
14
Model Complexity (number of clusters)

Bibliography

[1] M. K. Singh and D. Kumar G. *Effective Big Data Management and Opportunities for Implementation,* IGI Global, 2016.

[2] C. Anagnostopoulos and P. Triantafillou, *Efficient Scalable Accurate Regression Queries in In-DBMS Analytics.* IEEE 33rd International Conference on Data Engineering (ICDE), 2017.

[3] L. Xu and A. Krzyzak. *Rival Penalized Competitive Learning for Clustering Analysis, RBF Net, and Curve Detection.* IEEE Transactions on Neural Networks, 1993.

[4] A. Coates and A. Y. Ng. *Learning Feature Representations with K-Means.* Neural Networks, 2012.

[5] D. T. Pham et. al. *Selection of K in K-Means clustering.* Manufacturing Engineering Centre, 2004.

[6] E. Liberty et. al. *Online K-Means.* Yahoo!, Algorithm Engineering and Experiments (ALENEX), 2016.

[7] D. E. Rumelhart and D. Zipser. *Feature Discovery by Competitive Learning. Parallel distributed processing: explorations in the microstructure of cognition.* MIT Press Cambirdge, 1986.

[8] S. C. Ahalt et. al. Competitive Learning Algorithms for Vector Quantization. Neural Networks, 1990.

[9] G. Hebrail and A. Barard. *Individual household electric power consumption Data Set.* UCI Machine Learning Repository, 2012.

[10] T. F. Brooks et. al. *Airfoil Self-Noise Data Set*. UCI Machine Learning Repository, 2014.

[11] D. J. Bora and Dr. A. K. Gupta. *Effect of Different Distance Measures on the Performance of K-Means Algorithm: An Experimental Study in Matlab*. International Journal of Computer Science and Information Technologies, 2014.

[12] R. Loohach and K. Garg. *Effect of Distance Functions on K-Means Clustering Algorithm.* International Journal of Computer Applications, 2012.

[13] A. Singh et. al. *K-Means with Three different Distance Metrics.* International Journal of Computer Applications, 2013.

[14] L. Bottou and Y. Bengio. *Convergence Properties of the K-Means Algorithms.* Advances in Neural Information Processing Systems 7, 1995.

[15] L. I. Smith. *A tutorial on Principal Components Analysis.* Computer Science Technical Report, 2002.

[16] L. Maaten and G. Hinton. *Visualizing Data using t-SNE.* Journal of Machine Learning Research, 2008.

[17] Pedregosa et. al. Scikit-learn: Machine Learning in Python. JMLR 12, pp.2825-2830, 2011. Retrieved from: http://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

[18] J. Wesner. *MAE and RMSE – Which Metric is Better?* Human in a Machine World, March 23, 2016. Retrieved from:https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d

[19] T. Wood. *Occam's Razor and Model Complexity.* Contemporary Analysis Data Science, March 6, 2012. Retrieved from: https://blog.canworksmart.com/model-complexity

[20] Overfitting. Techopedia. Retrieved from: https://www.techopedia.com/definition/32512/overfitting

[21] G. Budura et al. *Competitive Learning Algorithms for Data Clustering*. Polytechnic University of Timisoara, 2005.

[22] T. H. Sardar and Z. Ansari. *Partition based clustering of large datasets using MapReduce framework: An analysis of recent themes and directions.* Future Computing and Informatics Journal, 2018.