Machine Learning Approach Towards Predicting Query Response Time

Kaushal Rai
University of Wisconsin-Madison
kaushal.rai@wisc.edu

Ansh Jain
University of Wisconsin-Madison
jain98@wisc.edu

ABSTRACT

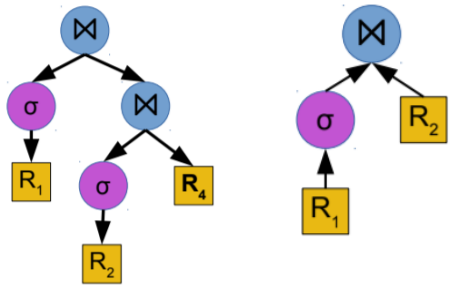
Predicting query response time has numerous applications in database management systems. Accurate estimation of a query's response time helps the system in scheduling, optimization, admission control, progress monitoring, system sizing, and deadlock prevention. Hence, in this project, we aim to explore various machine learning models to predict SQL query response time. We develop operator-level query predictors for some of the most common database operations, which include sequential scan, quick sort, hash join, index scan, hashing, limit, and nested loop index scan. This kind of model can generalize well for new and unseen queries and performs well for this task. We aim to particularly focus on supervised machine learning models for prediction such as support vector machines, linear regression, neural networks and analyze their result on the TPC-H benchmark. Further, we conduct a comparative study between these different techniques to establish the best-performing model for each operator.

1 INTRODUCTION

The potential of predicting query execution time accurately is crucial in modern databases. It eases the load on database management and administration systems. It can benefit a wide range of database challenges such as:-

(1) Query Scheduling - Predicting the query scheduling time, can help in latency aware scheduling
(2) Admission Control - We can prevent increasing the workload beyond a threshold and prevent thrashing
(3) Query Optimization - Different execution plans of a query might also result in different execution times. The system can choose the best plan based on minimum execution time.

Traditional approaches such as cost estimation for different query plans do not accurately reflect the execution time. Therefore there is a need for accurate prediction models that can evaluate the response/execution time of a query. The limitation of traditional approaches and the need for time estimation has motivated us to explore machine learning techniques for such a prediction. However, this idea of using machine learning to predict the response time presents numerous challenges. One of these challenges is that the queries are of various lengths and involve different types of operators as shown in Fig. 1. Encoding this information into feature vectors is difficult. Further, any model should generalize well to new queries. This can allow a model to be trained offline on a limited set of queries, and used in a real-time environment to predict unseen queries on the fly. It also saves the cost of online training which might cause strain on the database system and block resources for query execution.



(a) Query structure 1 **(b) Query structure 2**

Figure 1: Query visualization

In this project, we try to tackle these challenges and develop predictive operator-level models for query execution time predictions. We find a configuration that works optimally for all the operators by comparing multiple machine learning models based on different aspects. More specifically, we have analyzed supervised learning models - Neural Networks, Support Vector Machines, and Linear Regression. For all the models, we analyze the accuracy to suggest the best predictor for all operators. For this purpose, in our project, we use root mean square values between predicted and actual outputs.

The dataset used for training is the TPC-H decision support benchmark run on PostgreSQL. TPC-H consists of 22 query templates that have an industry-wide relevance with each query asking a critical business question. It also consists of multiple data set sizes ranging from several million to several billion elements to test different scaling factors. To learn the effect of scaling the data, the training set consists of data of varying sizes - 0.1, 0.5, 1, 2, 3 GB.

For the scope of this project, we have focused on only estimating specific query types -

- Sequential Scans
- Index Scans
- Hashing
- Sorting via Quick Sort
- Hash Join Operation
- Limit Operations
- Nested Loop Index Scan

Even though the results of this project have been impacted by the restricted amount of data that we have, we have selected queries having more data so that the trained model gives appreciable results.

A major challenge we faced was to identify the feature vector to represent each query. After going through numerous literature, we identified that the feature vectors should capture the various aspect of queries such as the size of the left and right child, width of left and right child, number of output tuples, etc. The features used are explained in more detail in the subsequent sections.

The remaining part of the report is structured as follows - Section 2 includes related work in this field, Section 3 describes the background required for the project. Section 4 describes our method followed by results in Section 5 and Conclusion in Section 6.

2 RELATED WORK

There have been a lot of work that use machine learning for query time prediction such as [4], [8], [12], [5], [7] etc. The work [12] aims to predict the execution time using a perceptron network, which is sufficiently complex to perform such a task without overfitting. The model is however evaluated on the TPC-DS benchmark database whereas our project focuses on the TPC-H benchmark. Another research work published in [8] presents a deep learning approach that eliminates the need for hand-engineered features by humans. The model can adapt to any kind of structure generated by the optimizer. This is achieved by discovering complex relations between both the operator as well as the query plan level. Therefore, the model adapts to different patterns of query execution and workload characteristics.

The main bottleneck of the task explored in our project is predicting the query with information available before query execution starts. This is tackled by some of the earlier papers such as [5]. The model proposed in [5] was able to predict elapsed times of queries for 85% of test queries within 20% of their actual time. The authors try multiple feature vectors to predict the query time. The first one is named the SQL feature vector which includes values such as number of non-equality selection predicates, number of sort columns, number of aggregation columns, etc. This feature vector, however, did not prove to attain high accuracy, because queries with the same feature vectors were found to have different runtimes. Therefore another feature vector called Query plan feature vector was explored which includes an instance count and cardinality sum for each operator in the query plan. This feature vector performed better than the SQL one.

Another important prior work for our project is [4]. This work studies different models that predict query execution time on different granularities. The authors showcase that the two extremes of course-grained plan level prediction and fine-grained operator level prediction offer a trade-off between bias and variance. For the plan level modeling, two types of models have been explored i.e. SVM and Kernel Canonical Correlation Analysis. The SVM requires feature selection by hand which is true for most algorithms whereas the KCCA performs dimensionality reduction automatically and therefore removes unnecessary and correlated features. The authors observed that the SVM model with the full set of features performed poorly when compared to a few selected features. Therefore they build the model iteratively by first using a small number of features and increasing the accuracy by adding more in order of correlation with the output. On the other hand, for the fine-grained model, two different prediction models are created for each operator i.e. start-time model and run-time model. The start-time model estimates the time until the first output tuple is produced, whereas, the run-time model estimates the total time of the operator including its sub-trees in the execution plan. For each operator, a fixed set of features are used including start time of left child operator, run-time for left child operator, start-time for right child operator, estimated input tuples, estimated selectivity, etc. For each operator, the multiple linear regression model is used and evaluated.

For our project, we take inspiration majorly from the above two research papers and implement another model for execution time prediction. The features used are similar to the above papers, and are listed in section 4. The main idea is that the features should be readily available for most database systems before the start of query execution. Further, we train our models on the operator level as they are more accurate in most scenarios according to prior research although they can reduce the speed and latency of prediction. For the entire query plan tree, the execution time is predicted by traversing the plan in a post-order manner and predicting the time for each operator. The prediction time of the root node is the final execution time.

3 BACKGROUND

In this section, we explain some important concepts for our project including the query optimizer, TPC-H dataset, and the machine learning algorithms used.

3.1 Query Optimizer

A query can be executed through different algorithms, and the query plan can have different tree structures. The job of a query optimizer is to choose the most efficient query plan. The optimization of a query can be done in multiple ways such as by minimizing the output tuples of intermediate queries, by reducing disk reads through indexing and hashing, etc. The optimizer compares different plans through cost estimates for each plan. In an ideal case, the cost should reflect the time taken to execute the query, however, this is not true in most cases. The cardinality estimates are one of the primary components for cost estimation, but it is has been studied in depth that the selectivity estimates have large errors [6]. Some of the reasons for these errors are also studied in research literature [10]. In practical scenarios, data is often skewed and contains related attributes which can result in high deviations in estimation. Even the query optimizers that use the time taken as a cost estimate, do not accurately measure this time and therefore the costs do not correlate to the actual time taken in practical use cases. The use of predictive models such as neural networks and SVMs counter the above issues, as they are trained on actual training times of different queries. These models, when given sufficient data, are highly accurate and predict the time for complicated queries better than hand-tuned cost estimators.

Query	Sequential Scan	Index Scan	Hash Join	Nested Loop Join	Sort
Q1	√				
Q2	√				
Q3			√	√	
Q4		√		√	
Q5				√	√
Q6					√
Q7			√	√	
Q8			√	√	
Q9				√	
Q10				√	
Q11				√	
Q12			√		√
Q13			√		
Q14		√			
Q15				√	
Q16				√	√
Q17				√	
Q18				√	√
Q19				√	
Q20				√	
Q21				√	
Q22			√		

Figure 2: Query Type and Corresponding Operation

3.2 TPC-H

TPC-H benchmark is a decision support system [2] that consists of complex business-related queries having broad industry-wide relevance. TPC-H consists of twenty-two queries templates that are executed for benchmark testing. Each of the twenty-two queries executes some fundamental operations [11]. Queries and their corresponding operations have been summarised in Fig. 2

TPC-H benchmark queries are often more complicated than most OLTP transactions. They consist of eight tables i.e. SUPPLIER, PART, PARTSUPP, LINEITEM, CUSTOMER, ORDER, NATION, and REGION. The schema for each of these tables has been summarised in 3. The schema figure has been borrowed from the following source [3]. The parenthesis along with each table name mentions the prefix column name for that table. The arrows show the one-to-many relationship mapping between different tables. The number mentioned below each table name shows the cardinality of the table (SF refers to the scaling factor). TPC-H includes multiple data set sizes to allow us the flexibility to test scaling factors. All tables adjust to the corresponding scaling factor, except for the LINEITEM table. The size for LINEITEM table is specified by the benchmark.

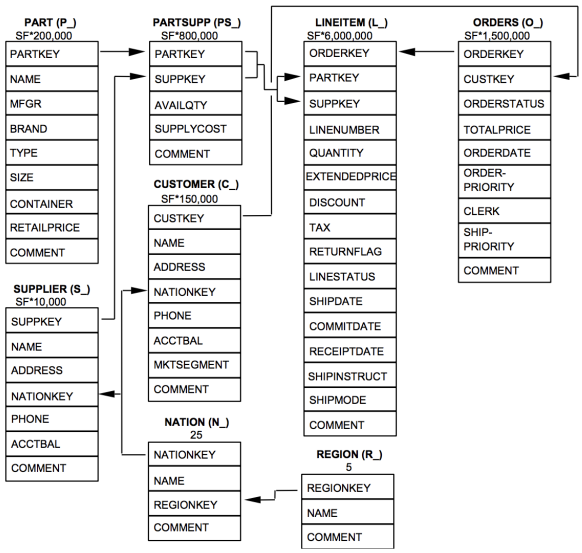


Figure 3: Schema of TPC-H

3.3 Support Vector Machine

SVMs have supervised machine learning models primarily used for classification, regression, and outliers detection tasks. SVMs are particularly suitable for high dimensional

data and identify a subset of training points called the support vectors that govern the hyperplane separating the training points into different classes. Refer to figure 4. The blue training points and the green training points lying on the dotted line form the support vectors, whereas the red line is the d-1 dimensional hyperplane separating the data points where d represents the number of features in training data.

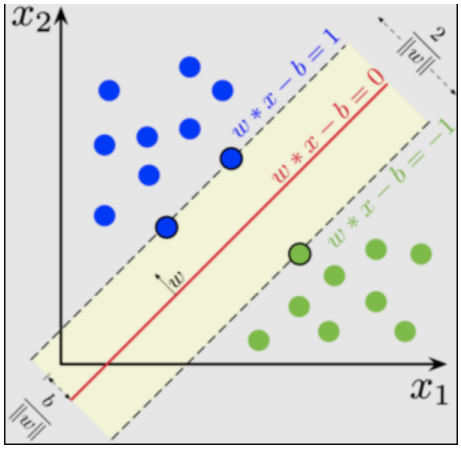


Figure 4: SVM

Since our project involves a regression task rather than a classification task, we will be using the support version regression (SVR) version of SVM. The SVR is based on the same idea as SVM, wherein we try to find a hyperplane that best fits the training points given.

3.4 Linear Regression

Linear regression models the relationship between scalar variables and explanatory variables. It tries to fit a linear model on the dataset by minimizing the sum of error between the actual target and the model's prediction. While predicting the output of multiple explanatory variables, linear regression is known as multiple linear regression. Linear regression can be used to predict the values of multiple correlated dependent variables as well, in which case it is known as multivariate linear regression.

3.5 Neural Networks

Neural networks refer to the class of models that is formed by layering multiple circuits of neurons. Each layer of neurons consists of several nodes which are formed by the weighted combinations of the nodes in the previous layers. Fig. 6 shows a typical neural network with a five feature input layer, 3 hidden layers, and a three feature output layer. Usually, given the large number of parameters associated with neural networks, they need a high volume of data to produce acceptable

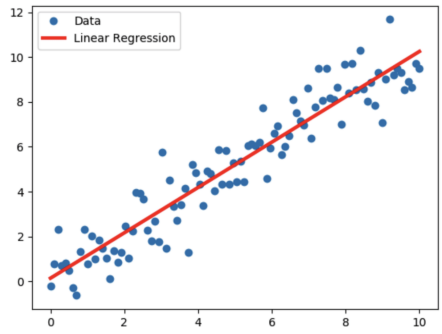


Figure 5: Linear Regression

results. To avoid overfitting of data, often drop out and pooling layers are added in between the layers of standard neural network.

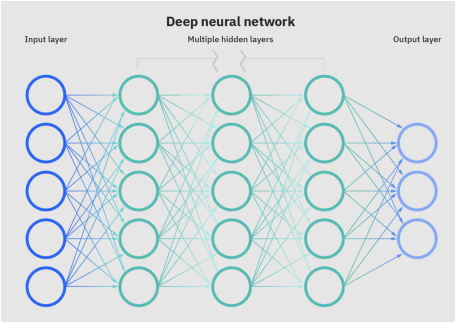


Figure 6: Neural Networks

4 METHOD

To gather the dataset for our project, we referred to the following Github repository [1]. The data was generated by running the TPC-H template queries on databases of multiple sizes on PostgreSQL. The dataset included a total of 2200 query execution along with their execution time. All the queries were divided into different folders based on the operators used. Out of all the operators given in the dataset, some did not have enough distinct data to ensure the training of our machine learning models. Therefore we decided to select the top 7 operations having the highest number of distinct data points. These are Sequential Scans, Index Scans, Hashing, Sorting via Quick Sort, Hash Join Operation, Limit Operations, and Nested Loop Index Scan. The dataset also consists of a number of training points that had the same set of features but distinct execution times. This is expected since the execution time for the same query on the same data can also differ.

The choice of training operator-level models arises from the need for generalization. It is crucial that our model is

able to predict "unseen" query plans which are not part of the training set. The operator-level modeling enables us to break down a query plan into a set of known operations and estimate the global cost by utilizing the local operator costs. This approach is similar to how SQL queries are actually executed and allows us to make predictions for complex and new queries.

While going through multiple pieces of literature, as mentioned in the related work section, we identified that the following set of features would be adequate to represent all the operators in scope for our project [5] [4] [9] - Number and width of left child input tuple, Number and width of right child input tuple, Number of output tuples, estimated I/O in number of pages, estimated operator selectivity, estimated total size of all tuples in bytes, estimated left and right child start and end time and whether the left and right child are blocking operators or not. Fig. 4 summarises the list of features used. Not all features make sense for all the operators that we have discussed. Therefore each operator has a different number of features with some features having default values. We have also normalized the data. This is essential since the different features have different ranges, and it is important to bring them to the same scale. Therefore, to ensure that the features equally impact the resultant parameters we have normalized the dataset.

Features
Number of left child input tuple
Width of left child input tuple
Number of right child input tuple
Width of right child input tuple
Number of output tuples
Estimated I/O in number of pages
Estimated operator selectivity
Estimated total size of all tuples in bytes
Estimated left child start time
Estimated left child end time
Estimated right child start time
Estimated right child end time
Is left child a Blocking operator(0/1)
Is right child a Blocking operator(0/1)

Table 1: Table showing the list of features used

To develop and train the machine learning models, we have used the scikit-learn library. For SVM, we have used kernelized support vector regression variation, having a radial basis function kernel (RBF) with a gamma value of 1. The model has been trained for 1000 iterations. The linear regression model is also trained in a similar way for all the operators.

To implement neural networks, we used the PyTorch framework. The neural network is a simple feed-forward NN with 1 hidden layer having 4 hidden neurons. The architecture was finalized after testing multiple variations and testing the output for each architecture. Also, in our implementation, we have used leaky Relu as the activation function. We tried to experiment with a Relu instead of a leaky Relu layer since the Relu function doesn't give a negative value and it makes sense to have a non-negative response time. However, the model was failing to converge with a Relu layer and hence a leaky Relu layer was used. For all the seven operators used, we have reported the corresponding root mean squared losses for all the models in the result section to help in identifying the best model.

5 RESULT

We had different number of data points for each of the operator as described below:

- Sequential Scan has 3500 datapoints were we have followed a 3100 and 400 train test split
- Index Scan has 2800 data points with 2400 and 400 train test split
- Hashing has 1600 data points with 1400 and 200 train test split
- Sorting via Quick Sort has 700 data points with 600 and 100 train test split
- Hash Join Operation has 1600 data points with 1300 and 300 train test split
- Limit Operation has 2100 data points with 1800 and 300 train test split
- Nested Loop Index Scan has 2100 data points with 1800 and 300 train test split

Table 5 summarises the result for each of the operators and the corresponding models. The table shows the root mean squared error for each of these operators. The formula for root mean squared error is given by:

$$\sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}{N}}$$

where $(x_i - \hat{x}_i)$ is equal to the deviation between our predicted time and the actual execution time. Linear regression performs the best for sequential scans. However, it performs the worst for the Quick Sort algorithm by some margin. SVM seems to give a better result for Quick Sort Algorithm. For hash join, linear regression gives the most optimal result, but all the models perform fairly well on it. SVM has a much better result for index scans as compared to the other two. Hashing gives the best results on a neural network, but overall all models have a comparable result for hashing. Hashing also shows the best results amongst all the operators. For limit operations, SVM shows the best result whereas for

Nested Loop Index Scan, neural networks have the better results.

Operation	SVM	Linear Regression	Neural Network
Sequential Scan	1840.05	361.04	885.05
Sort-QuickSort	5579.29	14378.54	9103.95
Hash Join	7683.32	3759.47	3906.35
Index Scan	4.11	500.60	9103.95
Hash	88.80	104.10	59.02
Limit	4650.45	63537.53	11719.12
Nested Loop Index Scan	5291.47	13268.74	2563.05

Table 2: Root mean squared error for test data for different models with different operations

6 CONCLUSIONS

Overall it appears that the SVM model has the better performance compared to the other two models across the majority of the operators. Due to the limited data that we have, neural networks do not perform as well as they were expected to perform. In the future, if more data is made available, it is possible that neural networks will have better performance compared to the other two. However, the training time required for SVMs is much shorter, compared to the training time required by the neural network. Another advantage of operator-level modeling is that we can deploy different models for different operators, whichever has the best performance. This technique is simple and would not require compromise on the accuracy of any of the operators. The result of various models and overall high error is also attributed to the fact that the same set of features produces different response times (as the execution of the same query on the same machine can take different times). This confuses the model and hinders the convergence process. Future work can be conducted to identify a better way to do this analysis by taking into account the different execution times of the same query and introducing some hardware statistics in the features as well. Also, in this project, we have assumed that we have a good estimate of data such as the number of output tuples, numbers of tuples processed, the width of left and right child, etc. before the query is executed. This might not be a practical assumption for a lot of queries. Therefore future work can also be focused on estimating these data points along with predicting the query response time.

REFERENCES

[1] 2016. https://github.com/pvam/pvam.github.io/tree/master/thesis/Code/Training

[2] 2021. TPC-H benchmark specification. http://www.tpc.org/tpch/

[3] visited. 2021. Snowflake's Documentation on TPC-H. https://docs.snowflake.com/en/user-guide/sample-data-tpch.html

[4] Mert Akdere, Ugur Cetintemel, Matteo Riondato, Eli Upfal, and Stanley B Zdonik. 2012. Learning-based query performance modeling and prediction. In *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 390–401.

[5] Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet L Wiener, Armando Fox, Michael Jordan, and David Patterson. 2009. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 592–603.

[6] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *Proceedings of the VLDB Endowment* 9, 3 (2015), 204–215.

[7] Jiexing Li, Arnd Christian König, Vivek Narasayya, and Surajit Chaudhuri. 2012. Robust estimation of resource consumption for sql queries using statistical techniques. *arXiv preprint arXiv:1208.0278* (2012).

[8] Ryan Marcus and Olga Papaemmanouil. 2019. Plan-structured deep neural network models for query performance prediction. *arXiv preprint arXiv:1902.00132* (2019).

[9] Vamshi Pasunuru. 2016. Predicting Query Execution Time using Statistical Techniques.

[10] Michael Stillger, Guy M Lohman, Volker Markl, and Mokhtar Kandil. 2001. LEO-DB2's learning optimizer. In *VLDB*, Vol. 1. 19–28.

[11] Pranav Vaidya and Jaehwan (John) Lee. 2008. Characterization of TPC-H Queries for a Column-Oriented Database on a Dual-Core Amd Athlon Processor. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management* (Napa Valley, California, USA) *(CIKM '08)*. Association for Computing Machinery, New York, NY, USA, 1411–1412. https://doi.org/10.1145/1458082.1458306

[12] Elif Ezgi Yusufoglu, Murat Ayyildiz, and Ensar Gul. 2014. Neural network-based approaches for predicting query response times. In *2014 International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 491–497.