Data Mining with Declarative Programming

CS240A: Databases and Knowledge Bases

Tianyu Zhang Jiapeng Wan

UID: 805332081

zhangtianyu@g.ucla.edu

**Introduction**

The objective of this project:

(i) to explore the new power of Datalog and SQL made possible by aggregates in recursive queries,

(ii) to become familiar with the Datalog/RaSQL system that support them on Apache Spark.

We also seek to understand better the challenges and di\_culties facing users. Thus, in your report you should also comment on the following:

1. di\_culties you faced in implementing these algorithms and

2. ease-of-programming with these declarative languages.

**1. Graph Algorithms and Sorting**

**2. Naive Bayes Classifier (NBC)**

The Naive Bayes Classifier is one of the fast, simplest, yet popular machine learning model due to its interpretability. It is also very easy to build and is particularly useful for large datasets. It is a simple "probabilistic classifier" based on applying Bayes' theorem with strong (naive) independence assumptions between the features. In other words, it assumes that presence of a particular feature in a class is unrelated to presence of any other feature. Even if the features depend on each other, all of them contribute independently to the probability.

In order to do so, we created the following scripts where each denotes a step in the data mining process:

**1. Load data and verticalize the training table** (NBCpreprocess.py)**:** This script is used to download the two datasets in csv file. Then we need to verticalize the data by reading and write them into a separate file. The final format of dataset will be readable by our RaSQL system.

**2. Handling numeric attributes ( ):** In order to handle the numeric attributes, like in the case of Bank Marketing Dataset, we have applied categorization and binning to some attributes, and used Gaussian distribution for others, depending on the nature of the attributes. We treated discrete numeric attributes with less number of distinct values as categorical. For the attributes that made sense to be categorized, we performed binning and divided it into categories. The continuous numeric attributes were handled using Gaussian distribution using the equation given below: We compute the above probability for each attribute value-class pair. Here, mean and standard deviation are used to compute the probability that a particular attribute belongs to a class.

**2. NBC Model implementation ():** This script trains a Naive Bayes Classifier by performing computations on the training data-set. We store the mean and standard deviation for each numeric attribute in a separate table that can be used during prediction.

**(a). handle missing values**

**3. K-Nearest Neighbors Classifier (KNN)**

A KNN classifier is a non-parametric machine learning classification model that predicts the class of a test instance based on the class labels k training instances closest to it.

**3.1. Datalog Model (DeAL)**

**1. The problem of syntax in RaSQL system**

At first we want to implement the the KNN in RaSQL system, however, during the process of implementation. We found out that there are still many syntax problems in RaSQL system to write datalog program.

The problem is the result of a sequence of operation. In one of the operation, we need to store the result of square of the difference of two variables:

*squares(Id,Tid,Col,C) <- test(Id,Col,V1),train(Tid,Col,V2),C=(V1-V2)\*(V1-V2).*

*sumsq(Id,Tid,sum<(C)>) <- squares(Id,Tid,Col,C).*

The first query generates a table with Id, Tid, and column with the square of difference of two values of two tables. The second query will sum up the values with same Id and Tid and create a table. The problem is that, when we want to create another table from *sumsq*, such as:

*minsq(Id,Tid) <- sumsq(Id,Tid,C).*

it will throw an error which tells us that:

*cannot resolve '`Id1`' given input columns: [Id, Tid, Aggr\_1];*

This problem happens if and only if both queries are applied. The table can be generated with column which resulted from variable operation and can be generated with column which sums up column with corresponding Id. The error will occur if and only if the column is generated by operation and then sums up. Since no more value can be extracted from the table, the further operation can not be proceeded.

**2.The implementation of KNN on DeAL.**

To implement the program and run it successfully, we tried to implement it on DeAL.

**(a) Preprocess the data(KNN.fac).**

In our program, we download the file from the website in data format and then load the data on csv file, then we verticalize the data to make sure the format is readable by DeAL. Then we manually put the data to DeAL. Since the DeAl is not able to deal with large amount of data in file, we will only put a small portion of data to it. The dataset we used are:

Hill\_Valley\_without\_noise\_Testing.data

Hill\_Valley\_without\_noise\_Training.data

**(b) Build the KNN classifier (KNN.deal)**

This file builds the KNN classifier from the training dataset and can be used to predict labels of test dataset. At first, we compute squares of difference in attribute values for all rows in test dataset. Then, we sum the squares by grouping using test Id. Though DeALS does not have a function to get the square root, we will work with squares since it will not cause any change in answers. Then we sort the square values in ascending order. To sort the table, we need to manually define a sorting function. After sorting and extracting top K values from the table, we will count the labels for each class for each test Id. In that case, we will be able to get the maximum value which represents the majority class out of K closest training records. This gives us the predicted label of the test record.

Though we are trying to build the classifier without recursion, it is not applicable. Because to compute the top K values, we have to manually write the sort function which cannot be accomplished with out recursion.

Moreover, the recursion in sort function can be improved if we use PreM(pre-mappability) during the calculation of top k closest neighbors. When the queries contains aggregations such as sum<>, min<> or max<>, PreM can be used to optimize the model semantics for exoprograms. To formulate PreM in DeAL, we just need to add drop-in goals. By using the drop-in goal in the body of rule, we can prove the PreM because it does not change the mapping defined by queries. Moreover, we can also prove the PreM by mentioning that the join in the body of recursive rules. By deriving the functional dependency using mixed transitivity, PreM is satisified.

**3.2 SQL model (RaSQL)**

In order to build the classifier in SQL, we created the following scripts, each doing a particular task in the mining process.

**1. Preprocess Data (KNNpreprocess.py)**

The script is used to first download the data by reading .data file and writing .csv file. Since the converted data file does not have the format that we will need in our classifier. We need to verticalize the data and write them in new file which will be used by our program.

**2. Build the KNN Classifier(knn.sql)**

This script creates a KNN Classifier by employing the training data-set, and predicting the classes for the testing dataset. We used Euclidean distance to calculate the distance between two instances. We follow the steps given below:

(a). We compute the square of difference in values for each of corresponding attributes between each set Id and all the training records.

(b). We calculate the distances by summing up distances and take square root of them for each attributes grouped by train ids.

(c). Then we calculate top K records having lowest values for Euclidean distance by using the Rank function in SQL. This function helps us to calculate top K values.

(d). Find the majority class among the top K records. This majority class gives the predicted label.

(e). Calculate the accuracy of classifier by calculating the ratio of correct predictions to total number of predictions, similar to what was done for NB classifier.

**3.3 SQL model (recursive RaSQL )**

**Results**

we trained and tested the classifier on only one dataset, but the same can run on different datasets, just by changing the load script in terms of dataset names. The resulting accuracies for the dataset (Hill\_Valley\_without\_noise\_Testing.data & Training.data ) is:

For K = 5, 61.2% for 40 sets; 50.57% for 100 sets; 55.49% for 300 sets.

For K = 7, 59.3% for 40 sets; 56.64% for 100 sets; 62.2% for 300 sets.

As K grows larger, the program takes longer to execute.

**Conclusion**

We have faced many challenges and problems in terms of RaSQL system both in datalog languages and RaSQL languages. Since such systems have their own syntax, modifying them is much harder, as compared to other systems. Moreover, even a slight mistake in syntax might cause the entire program to fail. The declarative version of the procedural language is much more complex to implement and understand. The error message generated by declarative languages is difficult to understand and debug i.e. some error messages are not intuitive.

(a). The RaSQL can directly load the file with correct format. However, we need to preprocess the file manually to make it readable by programs. In that case, we will need to write a preprocess program and read, write, and verticalize the data in the file.

(b). The main problem of RaSQL system is that though it supports datalog, there are many operations and functions are not supported by RaSQL system. In that case, if want to take the sum of a column after the column did multiplication, there will be errors. These kind of problems are hard to be shown on documents. To make the file compile successfully, we need try by changing syntax and see if that works.