CS 240A Final Project
Data Mining with Declarative Programming

Ajitesh Chandak
005024265
ajitesh@g.ucla.edu

Gurupavan Mazumdar
804945280
mazumdar.pavan@gmail.com

The goal of this project was to implement Naïve Bayes and KNN in declarative programming languages like SQL and DeALS. Following is the analysis of the implementation focusing on difficulties faced during implementation and some test results. Please note that detailed explanation of the working of the code is included in the README file.

- Project Experience:

The best part about this project was that it was paced in a very optimal manner. The development and design techniques were not too easy so as to feel a waste of time but at the same time they were not so tedious that they seemed impossible. Every step of the project we faced some milestone which we brainstormed and overcame with a feeling of success. This kept the engaging as well as giving of learning something new.

- Advantages and Disadvantages of working in a declarative style

The biggest advantage of declarative style programming is that you can directly get all the records in the database that you want by filtering by some criteria. For example, you can query using a SELECT statement and a WHERE clause with Class as EATABLE. This will return all the records for your use at convenience. There is no need to create a data structure like an array of strings to be later used. Moreover, in declarative programming since you don't specify how the code should run but are only interested in what the code should do, it is usually smaller. In any case, handling datasets with huge records is easier in declarative language.

Of course, the inconvenience is that declarative languages don't boast the Machine Learning libraries that modern languages like Python or R do. Considering the project that we implemented, a modern language would have a library that can be imported and called to do Bayes Classification. All that would need to be done is- load the dataset and process it according to the input arguments as required by the library and you are done. The drawback is that you are using a black box function whose functionality you don't know. But since it is an in built language library we can be fairly certain about the optimization, speed and accuracy.

Moving on to the more difficult ML techniques like training a neural network would also become quite difficult and extensive in SQL or Datalog whereas libraries such as TensorFlow are available for Python.

- Major Obstacles and Solutions during the course of the Project.
  - *Challenges*

One of the first problems we faced during the design of the project is in coming up with a method to handle generic data sets for Naïve Bayes classifier. With the requirement that the program should work on any data set, given any number of columns, class labels at unknown column numbers and no foreknowledge of features, the challenge was how to incorporate this in SQL where schema is fixed.

Moving deeper into the Bayes Classifier, the next challenge was that there can be two types of features. Categorical and Non-Categorical. Non-Categorical features would include numerical values, discrete values with infinite range and well as continuous values. Even if we knew how to process such attributes, how to automatically identify them and apply two different procedures to calculate likelihood for finding posterior probabilities.

Apart from these problems and a few problems regarding decision making as to what schema should look like, Bayes classifier on SQL fell into place nicely.

With KNN in SQL, one of the problems immediately identified was that the number of attributes in the dataset is huge (100 features!). Given that KNN requires calculation of a distance metric, we knew that would have to compute 100x606 calculations for one test sample (given that there are 606 sample in the training data). Multiply that with another 606 for the number of test data. That would be computationally expensive. Is there a vectorized way to implement this or are there any other optimizations that we could do? That was the question to be answered.

With KNN in DeALS, one of the problem was that it was difficult in getting the dataset transformed from the horizontal tabulated format to a verticalized format. Given was a special DeALS operator '@' that would perform the task, but then the tuple ID was missing. Further sorting of the training tuples based on the Euclidean distance calculated was not straight forward and robust.

  - *Solutions*

For the first problem we decided to go with suggestion in the specification i.e., verticalization. We decided that the schema for a verticalized table would need to contain original column number, column name and its value for the particular tuple. We created a separate verticalized table for class labels which contained tuple IDs and their corresponding classes. Since we didn't know the column no. of the class label, we took that as an input argument in our stored procedure using which we could split out verticalized tables. One table without that particular column and one with it.

For implementing naïve Bayes we needed the count of samples given a particular attribute, its value and corresponding class. This was simply achieved by joining the two verticalized tables on tuple ids. A new table was created which stored attribute name, value, class and corresponding count.

The counting method only worked for categorical attributes. For non-categorical attributes, we decided to go with fitting a Gaussian distribution with them. The first problem was to identify such attributes. This was simply done by adding a look up table which contains a list of columns that were continuous.

Next instead of saving count for these attributes, we created a separate table which stored mean and variance given attributes and class labels. During decision making phase we simply checked whether an attribute was continuous or not, if it was continuous then looked up the mean and variance from the table, plug in the attribute value from the test data as 'x' and calculated likelihood using the Gaussian PDF formula:

$$p(x_i|\mu) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x_i-\mu)^2}{\sigma^2}}$$

Here, μ would be the class and X_i is the value of specific attribute which is under consideration.

For verticalization of the data for DeALS input, a separate Java program was used to output the dataset as DeALS fact statements. This enabled to customized the statements as desired with the formatting necessary. Sorting the training tuples based on the distance was achieved by using the concept of magic sets to limit the computation only to the top K training tuples as needed, so no other values were computed. This made the computation a bit more optimized.

- *Other Roadblocks*:

The computationally expensiveness of KNN was only partially solved by us. Even before solving that we had to decide how to implement those calculations. We decided to take a page from our previous Naïve Bayes implementation and verticalize our data sets, so now we could iterate row by row given a column instead of column by column and row by row which is difficult to do.

Of course that still left us with 100x606 rows in our verticalized train table. We considered if it was possible to write a recursive query in SQL but feared stack overflow issues and decided to continue with our implementation. The execution time was around 2.5 hours on a MacBook Air 1.6 GHz Intel Core i5 (I5-5250U) which we felt could be improved but we were unsure how.

There seems to be a bug with the interpreter though with respect to whitespaces. For statements like facts and predicates, blank spaces between the name of the predicate and the opening parenthesis '(' does not seem to be an issue when interpreted. But for the statement like 'query prediction (X)' an error is thrown as 'No predicate matching query form for 'prediction|0' could be found. Fail' because of the space between 'prediction' and the opening parenthesis '('. Removing the space fixes the bug. Something that feels noteworthy to be pointed out.

Running the KNN DeALS code for all 606 training tuples resulted in excessive computation time of about 6+ hours on a MacBook Air 1.6 GHz Intel Core i5 (I5-5250U) for a single testing tuple. It seems that there might be another better way to implement the same logic, but if not, this time complexity is just not acceptable.

- Results

After overcoming the above mentioned difficulties we were able to successfully implement Bayes in SQL and KNN in DeALS and SQL. Following were the results in terms of accuracy that we obtained

Bayes in SQL:

For the first dataset, mushroom, all the attributes were discrete and categorical and we obtained an accuracy of $\frac{1611}{1619} = 99.5\%$.

For the second dataset, banking, with 20 features, in which 10 were categorical and 10 were numeric we achieved an accuracy of $\frac{728}{821} = 88.7\%$.

KNN in SQL:

The hill climbing dataset produced an accuracy of $\frac{317}{606} = 52.3\%$ with K as 20.

KNN in DeALS:

Accuracy calculation was not possible because we were not able to run the program for all the test tuples. However, for the 5 testing tuples that were manually tested for, 2 out of 5 were correctly predicted.