

CS240A: Databases and Knowledge Bases
Project: Data Mining with Declarative Programming

Devanshi Patel
UID: 504945601
devanshipatel@cs.ucla.edu

Saloni Goel
UID: 604944130
salonigoel@g.ucla.edu

Introduction

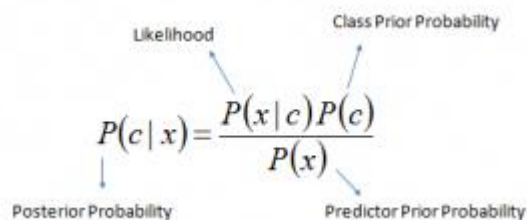
The goal of this project is to implement popular data mining algorithms, like Naive Bayes Classifier and K-Nearest Neighbors Classifier, in declarative systems (Datalog, RDBMS) and explore the challenges and difficulties faced from a programmer's point of view.

For the declarative systems, i.e., Datalog and RDBMS, we used DeALS and IBM DB2 respectively, to implement the algorithms.

Naive Bayes Classifier

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.

Bayes theorem provides a way to calculate posterior probability given likelihood and prior probability as shown below:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$


The Naive Bayes classifier has a lot of applications including spam classification, medical diagnosis and weather prediction. Our first goal is to construct a general purpose NBC in SQL, which works on any structured dataset, irrespective of the number or data type of the columns

and the number of classes. We have two datasets, namely, the Mushroom Dataset, and the Bank Marketing Dataset, to test our classifier.

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

1. **Load data** (load_data.sql): This script is used to load the two datasets in SQL tables with each record having a unique identifier. This ID is auto-generated and is useful for further processing to distinguish among different rows of tables.
2. **Verticalize data** (verticalize.sql): We need to compute counts for each attribute value and label pair to compute probability. However, if the dataset has a large number of columns we need to write the same query for all columns. This is highly inefficient and thus it is preferred to have a generic query.

To achieve this, we wrote the verticalization script that converts the data into a form that is suitable for any number of columns. The script helps in representing the data in a vertical format i.e. a horizontal row of table is expanded into multiple rows, one for each column.

3. **Handling numeric attributes** (attnumeric.sql): 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 have 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:

$$P(A_i | c_j) = \frac{1}{\sqrt{2\pi\sigma_{ij}^2}} e^{-\frac{(A_i - \mu_{ij})^2}{2\sigma_{ij}^2}}$$

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.

4. **Partition data** (partition.sql): In order to perform the task of mining, this script partitions the data into training and testing set (in the ratio of 4:1), with each set comprising of verticalized records of the original dataset. We select random rows and place it into training and test set to avoid any kind of bias. It should be noted that before partitioning, we update the table to set the isnumeric flag to true for all numeric attributes.

5. Building the Naive Bayes Classifier (nb.sql): 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: The missing values can be handled by either replacing it with median/mode or by ignoring them. For the mushroom dataset, only one attribute had missing values; we handled those values by considering them as separate attribute value. Since about 30% values were missing, it does not make sense to replace it by median/mode.

In case of banking dataset, six columns had missing values. Out of these columns, job and mstatus had very less missing values and hence we decided to ignore those values from rows. For rest of the attributes, the number of missing values were large so we decided to treat them as an attribute value.

b. Zero Frequency Problem : There are cases when an attribute value doesn't occur with every class value in the training dataset. In such cases, the probability will be zero as we are multiplying probabilities. Thus, it will nullify the probabilities of other columns. This is problematic and hence, it is desirable to apply a small sample correction in all probability estimates such that no probability value is ever set to zero.

One method is to add 1 to the count of every attribute value-class combination. This method is known as Laplace smoothing. In order to implement this in our classifier, we add 1 to the counts of each attribute value before computing the actual probability.

6. Testing the Naive Bayes Classifier (nbtest.sql): The trained model of the Naive Bayes classifier can be utilized to predict the class labels of the testing dataset. The script follows the following steps:

- i.** Get all relative probabilities from the Naive Bayes Classifier. For each record the probability with respect to a particular attribute is returned back. It should be noted that the calculation of probabilities is slightly different for numeric attributes. These probabilities are calculated using the equation presented earlier.
- ii.** Sum of all the calculated probabilities for each Id is calculated by summing the log values of the probability.
- iii.** This sum of probabilities is added to log of class probabilities to get final probability.
- iv.** The maximum of the probabilities is calculated which is later used as a grouping variable.

- v. As a final step, the predicted class label is found by using the maximum probability and the weighted probabilities calculated in the previous step.
- vi. We have also created a table that contains the accuracy of the classifier. This is computed by dividing the correct predictions by total number of predictions.

Results

1. For the purpose of this project, we trained and tested the classifier on only two datasets, but the same can run on different datasets, just by changing the load script, as per the required dataset.
2. We obtained the following accuracies for the provided datasets:
 - a. Mushroom Dataset: 95.75%
 - b. Bank Marketing Dataset: 85.77%
 - c. Also, for the Bank Marketing Dataset, if we ignore the attribute 'duration', we observed that the accuracy dropped to 80%. This aligns well with the fact that this attribute highly influences the target output variable which is mentioned in the dataset description file.
3. We faced various problems while building this classifier:
 - a. **Handling numeric attributes:** Initially we were not sure whether to use binning, gaussian distribution or categorization to handle numeric attributes. We tried different methods for each numeric attribute and observed its effect on accuracy of the classifier. Eventually, we ended up with gaussian distribution for continuous attributes and binning for discrete attributes.
 - b. **Handling missing values:** We did so by considering them as an attribute value. We also tried ignoring them, but that reduced the accuracy of the classifier. Also, replacing them by the mean/median is not possible, since the missing attributes are categorical.
 - c. **Handling the frequency zero problem:** Initially, when the classifier encounters unseen combination of attribute value and class label, it used to assign zero probability. But this is not correct as we are ignoring the effect of other features. To remedy this, we used the Laplace smoothing technique, which adds 1 to the count for every attribute value-class combination (Laplace estimator), due to which the probabilities will never be zero.

K-Nearest Neighbors Classifier

K-Nearest Neighbors classifier is a non-parametric machine learning classification model that predicts the class of a test instance based on the class labels of K training instances closest to it. In k-NN classification, the output is a class membership.

An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

Our next goal is to create the KNN classifier in SQL and DeALS. For this classifier, we will test the performance on the Hill-Valley dataset in both the systems.

KNN Classifier (DB2)

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

1. **Load data** (load.sql): This script was used to load the training and testing datasets in SQL tables with each record having an automated ID number, which is useful for further processing.
2. **Verticalize data** (verticalize.sql): In order to make the classifier work on any number of columns, this script helps in representing the data in a vertical format i.e. a horizontal row in the format is expanded into multiple rows. This was done similarly as done in the Naive Bayes Classifier.
3. **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:
 - i. First, we compute squares of difference in values for each of corresponding attributes between each test Id and all the training records.
 - ii. Next, we calculate distance by taking the square root after summing up distances for each attribute by grouping train Ids.
 - iii. We now calculate top K records having lowest values for Euclidean distance by using the Rank functionality available in SQL. This function makes it really easy to calculate top K values.
 - iv. The next step is to find the majority class among the top K records. This majority class gives the predicted label.
 - v. Finally, we 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.

Results

1. For the purpose of this project, we trained and tested the classifier on only one dataset, but the same can run on different datasets, just by changing the load script, as per the required dataset.
2. We obtained the following accuracies for the provided dataset:

For $K = 5$,

- i. Accuracy is 57.5% with 40 records.
 - ii. Accuracy is 56% with 100 records.
 - iii. Accuracy is 53.13% when tested with the whole dataset, i.e., 606 records.
3. We tried to test with $k=10$ but since KNN classifier is highly inefficient, it takes a really long time to execute.

KNN Classifier (DeALS)

After implementing KNN in DB2, we now build the classifier in DeALS. The following steps are involved:

1. **Load Data and Verticalize:** We wrote a script in Python that loads the data from the files and directly generates facts in the verticalized format. We directly feed these facts in DeALS. Additionally, since it might be difficult for DeALS to work with huge amount of data, we work with only a subset of data with limited number of rows and columns.
2. **Build the KNN classifier** (knn.deal): This script builds the KNN classifier from the training dataset and can be used to predict labels of test dataset. We first compute squares of difference in attribute values for all rows in test dataset. Then, we take sum of squares by grouping using test Id. Since DeALS does not have a functionality to calculate square root, we work directly with squares as it won't cause any change in answers.

After this, we need to sort the square values in ascending order. This is not as easy in DeALS as it is in SQL. In SQL, we can just use order by clause to sort the values in ascending order. But, in case of DeALS, we had to define the sorting function manually using a set of rules. After sorting and fetching top K values, we count number of labels for each class for each test Id. From the counts, maximum value is selected which represents the majority class out of K closest training records. This gives the predicted label of the test record.

It is worth noting that it is not possible to build the classifier without recursion. This is because in order to compute the top K values, we need to write a sorting function and this function cannot be implemented without recursion. Thus, we use recursion in our DeALS program to build KNN classifier. Additionally, this recursion can be made even more efficient if we utilize pre-mappability (PreM) during the calculation of nearest K neighbors.

PreM is applicable in cases where the recursive rule contains any aggregate such as sum, count, min or max. It allows us to optimize perfect model semantics for exo- programs. It is very easy to formulate PreM in DeALS by adding drop-in goals. However, the challenge lies in proving PreM. One easy way to prove is usage of drop-in goal in the body of rule. Since this does not change the mapping defined by our rule, we can say the PreM is proved. Alternatively, it is also possible to prove PreM by observing the join in the body of recursive rule. If we can derive the functional dependency using mixed transitivity, PreM is said to be satisfied.

Observations

1. We found that building these classifiers is easier in declarative systems, rather than procedural ones because:
 - a. In procedural language, you define the whole process and provide the steps how to do it. You just provide orders and define how the process will be served.
 - b. Whereas, in declarative language, you just set the command or order, and let it be on the system how to complete that order. You just need your result without digging into how it should be done.
 - c. Also, programs made with the declarative languages are smaller than the ones in procedural.
 - d. A major advantage of declarative languages is the error recovery mechanism which is really helpful if we are using it for critical applications. It is easy to specify a construct that will stop at the first error instead of having to add error listeners for every possible error.
 - e. Though we were new to programming in these languages, due to the simplistic nature of declarative languages, we were able to pick up the basic concepts of the paradigm quickly and write the programs with ease.
 - f. Additionally, calculating aggregates and grouping is very easy and simple in case of declarative languages.
 - g. Sorting was comparatively simpler in SQL compared to DeALS which required writing a set of rules to sort.

- h.** Another advantage of declarative languages is that it allows us the flexibility if we want to implement procedures for certain tasks. For eg: we have created procedures that help in verticalization of tables.

2. However, we faced some challenges while using these systems:

- a.** Since such declarative 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.
- b.** Also, declarative systems never have the full power of a programming language.
- c.** Sometimes, the declarative version of the procedural language is much more complex to implement and understand.
- d.** Sometimes, the error message generated by declarative languages is difficult to understand and debug i.e. some error messages are not intuitive.
- e.** DeALS does not provide a way to load facts directly from file and this is highly inconvenient. Moreover, the verticalization construct available in DeALS is not very useful as it cannot distinguish ID from other column values. So it tries to verticalize ID as well. Hence, we had to write a separate script in Python that directly generates verticalized facts.
- f.** We also faced some challenges initially when trying to verticalize tables in SQL. This is because we were not familiar with Dynamic SQL queries. Gradually, we learnt how to use them and we found that the knowledge of dynamic SQL and procedures made implementation much easier and readable.
- g.** Additionally, we struggled initially while trying to handle numeric attributes. Deciding which method should be used to handle these attributes is not easy. Sometimes, a particular way of handling attributes results in degrade of accuracy.
- h.** A major shortcoming of SQL and DeALS is that both cannot handle large amount of data. While testing for KNN in DB2, when I tried to evaluate the performance using a bigger K value, it gave an error saying that “The transaction log for database is full”. Thus, it fails to work with even medium amount of data. Even DeALS failed while trying to load all 100 attributes from the dataset. To remedy this, we worked with a subset of data in DeALS.
- i.** It is worth noting that although the declarative languages worked well for simple classifiers like NB and KNN, these languages cannot be used for more complex and advanced machine learning tasks like Deep Learning. The declarative languages lack a lot of constructs that are needed to perform complex tasks.