CS 240A : Databases and Knowledge Base
Project #2

Ronak Sumbaly
UID : 604591897

February 29, 2016

Project Details

The project is divided into 6 scripts each signifying a particular task in the mining process. Click on the section titles in the report to go to the Github repository.

1. **Load Data** : The database obtained from the Delve website is loaded into DB2 (Script : load_data.sql)
2. **Partition Data** : In order to perform the task of mining we partition the data into training and testing set with each set comprising of verticalized records of the original data-set (Script : partition_data.sql)
3. **Build Naive Classifier** : A Naive Bayesian Classifier is created by employing the training data-set (Script : naive_bayes.sql)
4. **Test Naive Classifier** : The results of the naive Bayesian classifier is utilized as a model to predict the class labels of the testing data-set records (Script : naive_testing.sql)
5. **Boosting** : In order to increase the accuracy of the model we perform boosting on the Naive Bayesian Classifier until no rise in accuracy is seen. (Script : boosting.sql)
6. **1R Classifier** : 1R Classifier is implemented and its accuracy is compared with the Naive Bayes Classifier. (Script : 1R.sql)

Project Accomplishments

1. The project was created on the Titanic data-set but can be ran on different data-set by just changing the Load and Partition script as per the required dataset.
2. After performing all the tasks mentioned above the final Naive Bayesian Classifier accuracy was found out to be 73.9 % and 1R Classifier accuracy = 78.18%
3. Boosting was performed on the Naive Bayes Classifier as well and ran for 2 runs since there was no increase in the accuracy the process was stopped at 2 runs itself.
4. In order to handle missing values the script has conditions to omit NULL values encountered during execution.
5. The dataset did not require discretization but this can be formed by converting continuous values to ranges and then proceeding forward in the same way as above.

Load Data

The following script loads the data-set into DB2 with each record having an automated ID number inserted alongside it for referral.

Listing 1: Load Data

```
------ LOAD ------
-- This script loads the data from an external file and stores it into a table called "DATASET."
-- run using (-tvf)

---- INITIALIZATION ----

CONNECT TO SAMPLE; -- CONNECT TO DATABASE

CREATE TABLE DATASET("PID" INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH
    1, INCREMENT BY 1, NO CACHE ) , "CLASS" VARCHAR (10), "TYPE" VARCHAR (10),"SEX"
    VARCHAR (10),"SURVIVED" VARCHAR (10)); -- CREATE TABLE WITH NAME DATASET
    AND FIELDS SIMILAR TO TITANIC DATASET

---- LOAD FROM DATA FILE OF DATASET ----

LOAD FROM 'C:/Users/Ronak Sumbaly/Downloads/Dataset.data' OF del INSERT INTO
    DATASET(Class, Type, Sex, Survived); -- LOAD DATASET FROM LOCAL DISK

CONNECT RESET ; -- RESET CONNECTION
TERMINATE; -- TERMINATE DB2 PROCESS
```

Partition Data

In order to support the task of data mining and modelling of a classifier, the imported data was split into training and testing data with the former containing about three times as many tuples as the latter. The script is as follows,

Listing 2: Partition Data

```
------ PARTITION ------
-- This script partitions the initial database table (dataset) into two
-- subsets: testset and trainset. It also verticalizes the data in the
-- sense that it creates a record in testset or trainset for each column
-- run using (-td~ -vf)

---- INITIALIZATION ----

CONNECT TO SAMPLE~

-- drop the tables we're about to create, in case they already exist
DROP TABLE TEST_DATASET~
DROP TABLE TRAIN_DATASET~

-- create the tables

CREATE TABLE TEST_DATASET("PID" INTEGER, "COLUMNNO" INTEGER, "ATT"
    VARCHAR(10),"DECISION" VARCHAR(10), "WEIGHT" INTEGER)~ -- CREATE TEST
    DATASET TABLE
CREATE TABLE TRAIN_DATASET("PID" INTEGER, "COLUMNNO" INTEGER, "ATT"
    VARCHAR(10),"DECISION" VARCHAR(10), "WEIGHT" INTEGER)~ -- CREATE TRAIN
    DATASET TABLE

-- transfer content to trainset and testset

BEGIN ATOMIC
        FOR temp AS SELECT * FROM DATASET ORDER BY PID DO --LOOP THROUGH ENTIRE
            DATASET
        IF rand() > 0.75 THEN -- SELECT RECORDS ON RANDOM AND ADD TO TEST IF <
            75 %
                INSERT INTO TEST_DATASET VALUES
                (temp.PID, 3, temp.sex, temp.survived, 1),
                (temp.PID, 1, temp.class, temp.survived, 1),
                (temp.PID, 2, temp.type, temp.survived, 1);

        ELSE -- SELECT RECORDS ON RANDOM AND ADD TO TRAIN
                INSERT INTO TRAIN_DATASET VALUES
                (temp.PID, 3, temp.sex, temp.survived, 1),
                (temp.PID, 1, temp.class, temp.survived, 1),
                (temp.PID, 2, temp.type, temp.survived, 1);
        END IF;
        END FOR;
END~

CONNECT RESET ~
TERMINATE~
```

Build Naive Bayes Classifier

In order to build the Naive Bayes model a **WITH statement** was used to group the data according to the record's class label and the record's attribute value. The sum of the weights of these groups was calculated. Each weights sum is then divided by the sum of the weights of each class. A condition is added in order to handle values that are NULL for a particular record. The script is as follows,

Listing 3: Naive Bayes

```
------- NBC -------
-- This script creates the NBC and NBC_class tables. NBC is a Naive
-- Bayesian Classifier that gives us P(a|c) for some attribute a and
-- classifier c. NBC_class gives us P(c).

---- INITIALIZATION ----
CONNECT TO SAMPLE;

-- drop tables
DROP TABLE NBC;
DROP TABLE NBC_CLASS;

-- create the tables
CREATE TABLE NBC("NUM" INTEGER DEFAULT 1, "COL" INTEGER, "ATT" VARCHAR(10),
    "DECISION" VARCHAR(10), "PROBABILITY" DECIMAL(11,10)); -- CREATE NBC TABLE
CREATE TABLE NBC_CLASS("NUM" INTEGER DEFAULT 1, "DECISION" VARCHAR(10),
    "PROBABILITY" DECIMAL(11,10)); -- CREATE NBC CLASS TABLE TO STORE
    PROBABILITY

INSERT INTO NBC(COL, ATT, DECISION, PROBABILITY) -- PERFORM NAIVE BAYES
    CLASSIFICATION
       WITH GROUPED_DATA(COLUMNNO, DECISION, ATT, SUM_WEIGHT) AS
       (SELECT COLUMNNO, DECISION, ATT, SUM(WEIGHT) -- CALCULATE THE OCCURRANCE
           OF EACH CLASS USING WEIGHT COLUMN
              FROM TRAIN_DATASET
               GROUP BY GROUPING SETS((COLUMNNO,DECISION), (COLUMNNO, DECISION,
                   ATT)))
       (SELECT GD1.COLUMNNO, GD1.ATT, GD1.DECISION,
                     CAST(GD1.SUM_WEIGHT AS DECIMAL(6,0)) / CAST(GD2.SUM_WEIGHT AS
                         DECIMAL(6,0)) -- DIVIDE THE SUM OF COUNTS WITH THE SUM
                         OF TOTAL NUMBER OF COUNTS OF SPECIFIC RECORD
           FROM GROUPED_DATA AS GD1 , GROUPED_DATA AS GD2
                     WHERE GD1.COLUMNNO = GD2.COLUMNNO
                     AND GD1.DECISION = GD2.DECISION
                     AND GD1.ATT IS NOT NULL -- ADD TO NBC TABLE ONLY IF THE
                         CLASS NAMES ARE MATCHING
                     AND GD2.ATT IS NULL);

INSERT INTO NBC_CLASS(DECISION, PROBABILITY)
       WITH TotalCount(VALUE) AS (SELECT SUM(WEIGHT) FROM TRAIN_DATASET WHERE
           COLUMNNO = 1),
              ClassCount(DECISION, VALUE) AS (SELECT DECISION, SUM(WEIGHT) FROM
                  TRAIN_DATASET WHERE COLUMNNO = 1 GROUP BY DECISION)
       (SELECT classObj.DECISION,
                     CAST(classObj.VALUE AS DECIMAL(10,0)) / CAST(totalObj.VALUE AS
                         DECIMAL(10,0)) -- DIVIDE THE SUM OF EACH CLASS FOUND IN
                         NBC TABLE WITH TOTAL NUMBER OF RECORDS TO GET
                         PROBABILITY
                     FROM TotalCount as totalObj, ClassCount as classObj);

CONNECT RESET;
TERMINATE;
```

Testing Naive Bayes Classifier

The testing part of the Naive Bayes Classifier involves

1. Get all relative probabilities from the Naive Bayes Classifier. For each record the probability with respect to a particular attribute is returned back.
2. Sum of all the calculated probabilities for each ID is calculated by summing the log values of the probability.
3. The weights of the probabilities is added to the final NBC table.
4. The maximum of the probabilities is calculated which is later used as a grouping variable.
5. As a final step the predicted class label is found by using the maximum probability and weighted probabilities calculated in the previous step. The aggregated percentages is presented in the end.

Listing 4: Test Naive Bayes

```

------ TEST ------
-- This script is used to test the model of the training data with the test dataset

---- INITIALIZATION ----

CONNECT TO SAMPLE;

-- drop tables
DROP TABLE TEST_RESULTS;
DROP TABLE COMBINED_RESULTS;

-- create tables
CREATE TABLE TEST_RESULTS("NUM" INTEGER DEFAULT 1, "COLUMN" INTEGER, "ACTUAL"
    VARCHAR(10), "PREDICTED" VARCHAR(10)); -- CREATE TABLE TO STORE CLASS
    LABELS FOR TEST DATASET
CREATE TABLE COMBINED_RESULTS("NUM" INTEGER DEFAULT 1, "ACCURACY"
    DECIMAL(11,10)); -- TEST THE ACCURACY

INSERT INTO TEST_RESULTS(COLUMN, ACTUAL, PREDICTED)
       WITH -- USAGE OF THREE TABLE TO GET THE PROBABILITY, WEIGHTS AND MAX
           PROBABILITY OF EACH RECORDS IN THE TEST DATASET
              PROBABILITY(ROW, PREDICTED, PROB) AS
              (SELECT TEST_DATASET.PID, NBC.DECISION, NBC.PROBABILITY
                     FROM TEST_DATASET, NBC
                     WHERE TEST_DATASET.COLUMNNO = NBC.COL
                     AND TEST_DATASET.ATT = NBC.ATT),
              PROBABILITYSUM(ROW, PREDICTED, PROB) AS
              (SELECT ROW, PREDICTED, SUM(LOG(PROB)) FROM PROBABILITY GROUP BY ROW,
                  PREDICTED),
              WEIGHTPROB(ROW, PREDICTED, PROB) AS
              (SELECT PROBABILITYSUM.ROW, PROBABILITYSUM.PREDICTED,
                  PROBABILITYSUM.PROB + LOG(NBC.PROBABILITY)
              FROM PROBABILITYSUM, NBC
              WHERE PROBABILITYSUM.PREDICTED = NBC.DECISION),
              MAXPROB(ROW, PROBABILITY) AS
              (SELECT ROW, MAX(PROB) FROM WEIGHTPROB GROUP BY ROW)
       (SELECT TD.PID, TD.DECISION, WP.PREDICTED
       FROM TEST_DATASET as TD, MAXPROB as MP, WEIGHTPROB AS WP
       WHERE TD.PID = MP.ROW
       AND TD.PID = WP.ROW
       AND MP.PROBABILITY = WP.PROB
       AND TD.COLUMNNO = 1); -- ASSIGN ONE IF CORRECTLY PREDICTED ELSE 0
```

Listing 5: Test Naive Bayes (Contd)

```
INSERT INTO COMBINED_RESULTS(ACCURACY)
       WITH
              SUCCESS(NUM) AS
                     (SELECT COUNT(PREDICTED) FROM TEST_RESULTS WHERE PREDICTED =
                         ACTUAL),
              TOTAL(NUM) AS
                     (SELECT COUNT(PREDICTED) FROM TEST_RESULTS)
              (SELECT CAST(SUCCESS.num AS DECIMAL(10,0))/CAST(TOTAL.num AS
                  DECIMAL(10,0)) - CALCULATE THE \% of CORRECTLY CLASSIFIED LABELS
              FROM SUCCESS, TOTAL);
```

Results

The Naive Bayes Classifier was ran on the testing dataset (541 tuples) after creation of the model using the training dataset (1660 tuples). The accuracy obtained was,

$$\text{Naive Bayes Classifier Accuracy} = 73.9371$$

Boosting

Since the script of Boosting is too large the former can be accessed by clicking on the title. The script is just a basis augmentation of already existing scripts and makes use of the weight column in the training and testing dataset. The script comprises of various procedures which update the weights of the records. The strategy was to add records that were incorrectly classified in the testing data-set and add it to the training data. The entire process of NBC model building is ran again multiple times until the accuracies don't change.

It was observed that Boosting ran twice and gave the same accuracy both times. The code was verified and can be used for remaining datasets which might give different results.

1R Classifier

In order to create a 1R Classifier we proceed as follows,

1. The frequent class labels for each attribute value is found and stored.
2. The count of the class labels for each of these attributes are found.
3. From these counts the maximum class label count is found.
4. Using this we find the number of rows in the training data that do not match this label and denote this as the error count. The minimum of this error is called as the splitting attribute.
5. Using the splitting attribute the 1R decision tree is formed and tested.

Listing 6: 1R Classifier

```
------ 1R Classifier ------
-- Script to model 1R Classifier

CONNECT TO SAMPLE;

-- drop the tables we're about to create, in case they already exist
DROP TABLE FREQUENT_CLASS;

-- create the tables
CREATE TABLE FREQUENT_CLASS("COLUMNNO" INTEGER, "ATT" VARCHAR(10),
    "CLASS_COUNT" INTEGER, "DECISION" VARCHAR(10));

INSERT INTO FREQUENT_CLASS
WITH TEMP1 AS(SELECT COLUMNNO, ATT,COUNT(DECISION) AS CLASS_COUNT,DECISION FROM
    TRAIN_DATASET GROUP BY COLUMNNO,ATT,DECISION),
TEMP2 AS(SELECT COLUMNNO,ATT,MAX(CLASS_COUNT) AS MAX_CLASS_COUNT FROM TEMP1 GROUP BY
    COLUMNNO,ATT),
TEMP3 AS(SELECT T1.COLUMNNO, T1.ATT, T1.CLASS_COUNT, T1.DECISION FROM TEMP1 AS T1
    ,TEMP2 AS T2 WHERE T1.CLASS_COUNT=T2.MAX_CLASS_COUNT AND T1.COLUMNNO=T2.COLUMNNO)
SELECT * FROM TEMP3; -- CALCUALTE THE MOST FREQUENT COLUMN IN THE DATASET

-- drop the tables we're about to create, in case they already exist
DROP TABLE SPLIT_ATTR;

-- create the tables
CREATE TABLE SPLIT_ATTR("COLNO" INTEGER);

INSERT INTO SPLIT_ATTR
WITH TEMP1 AS(SELECT T.COLUMNNO,COUNT(T.COLUMNNO) AS ERROR_COUNT FROM TRAIN_DATASET
    AS T, FREQUENT_CLASS AS F WHERE T.COLUMNNO=F.COLUMNNO AND T.ATT=F.ATT AND
    T.DECISION!=F.DECISION GROUP BY T.COLUMNNO),
TEMP2 AS(SELECT COLUMNNO,ERROR_COUNT FROM TEMP1 WHERE ERROR_COUNT= (SELECT
    MIN(ERROR_COUNT) FROM TEMP1))
SELECT COLUMNNO FROM TEMP2; -- SPLIT WITH RESPECT TO MOST FREQUENT ITEM

CALL GET_ACCURACY_1R_TREE(?,?,?);

---- CLEAN UP ----
TERMINATE;
```

Listing 7: 1R Classifier Procedure

```

CREATE OR REPLACE PROCEDURE
GET_ACCURACY_1R_TREE(OUT TOTAL_ROWS DECIMAL, OUT ACCURATE_ROWS DECIMAL, OUT ACCURACY
    Decimal(18, 13))
BEGIN

SELECT DECIMAL(COUNT(DISTINCT T.PID)) INTO ACCURATE_ROWS
FROM TEST_DATASET AS T , SPLIT_ATTR AS S, FREQUENT_CLASS AS F
WHERE T.COLUMNNO=S.COLNO AND F.COLUMNNO=S.COLNO AND T.COLUMNNO=F.COLUMNNO AND
    T.ATT=F.ATT AND T.DECISION=F.DECISION;

SELECT DECIMAL(COUNT(DISTINCT PID)) INTO TOTAL_ROWS FROM TEST_DATASET;

SET ACCURACY=(ACCURATE_ROWS/TOTAL_ROWS)*100;

END@
```