CS240A Final Project 2

Nikita Tulpule
UID: 004761086

December 14, 2016

Introduction: Data Mining in SQL and Datalog

In this project, I have implemented Naive Bayes and 1R classifiers using Datalog and SQL. The classification has been implemented using DeAL and DB2. The datasets I considered are titanic, mushroom and letter. All the scripts and this project report are available at this link.

Task A: DeAL

DeAL was used to generate a Naive Bayes and 1R classifier. The classifiers are fairly straightforward. Missing data values are ignored. I implemented these on titanic dataset along with the sample buyscomputer dataset.
I created a bash script to transform data files to DeAL facts. The convert.sh script needs to be executed first. It takes following arguments - datafile, file with deal rules(proj.deal for NBC and proj2.deal for 1R) and number of parameters. The datafiles are space separated. The deal rules currently work on datasets with two classes.
Next run the trial.deal file to execute the classifier.
Scripts and how to run instructions can be found at this link.

Scripts

1. Naive Bayes

```
%facts
testdata(1,1,first).
testdata(1,2,adult).
testdata(1,3,male).
testlabels(1,yes).
patterns(2,1,first).
patterns(2,2,adult).
patterns(2,3,male).
labels(2,yes).
....data omitted....
patterns(2200,1,crew).
patterns(2200,2,adult).
patterns(2200,3,female).
```

```
labels(2200,no).
testdata(2201,1,crew).
testdata(2201,2,adult).
testdata(2201,3,female).
testlabels(2201,no).

%rules
query getcnt(X,No).
getcnt(X,count<A>) <- labels(A,X).
gettotal(count<A>) <- labels(X,A).
getclasprob(X,C) <- getcnt(C,No),gettotal(Y),X=No/Y.

getelement(A,D,C,count<No>) <- patterns(No,A,D),labels(No,C).

getelementprobability(Col,A,B,P) <-
   getelement(Col,A,B,X),getcnt(B,Y),P=X/Y.

mul_prob(Id,C,sum<S>) <-
   testdata(Id,Col,Val),getelementprobability(Col,Val,C,P),S=log P.

mul_prob_data(Id,C,P) <-
   mul_prob(Id,C,P1),getclasprob(P2,C),P=P1+log P2.

finalprob(Id,C) <-
   mul_prob_data(Id,yes,Y),mul_prob_data(Id,no,N),if(Y>N then
   C=yes else C=no).

correct(count<Id>)<-testlabels(Id,L),finalprob(Id,C),L=C.
total(count<A>) <- testlabels(X,A).
accuracy(Z) <- correct(X),total(Y),Z=(X/Y)*100.
query accuracy(Z).
```

Results of Naive Bayes on titanic dataset: 77.67% accuracy.

```
nikita@nikita:~/Documents/CS240/project2/cs240A-DeALS-F16$
   truncate -s 0 trial.deal
nikita@nikita:~/Documents/CS240/project2/cs240A-DeALS-F16$ sh
   convert.sh data_titanic.data proj.deal 3
nikita@nikita:~/Documents/CS240/project2/cs240A-DeALS-F16$ time
   bash runfile trial.deal
Executing query 'getcnt(X, No).'.
Query Id = getcnt(X,No).1481893580335
Query Form = getcnt(X, No).
Return Status = SUCCESS
Execution time = 12ms
getcnt(yes, 533).
getcnt(no, 1117).
2 results

Executing query 'accuracy(Z).'.
Query Id = accuracy(Z).1481893610634
```

```
Query Form = accuracy(Z).
Return Status = SUCCESS
Execution time = 102923ms
accuracy(77.6769509981851).
1 result

real 2m33.787s
user 3m41.036s
sys  0m1.924s
nikita@nikita:~/Documents/CS240/project2/cs240A-DeALS-F16$
```

2. 1R

```
%facts
testdata(1,1,first).
testdata(1,2,adult).
testdata(1,3,male).
testlabels(1,yes).
patterns(2,1,first).
patterns(2,2,adult).
patterns(2,3,male).
labels(2,yes).
.......data omitted...
testdata(1649,1,crew).
testdata(1649,2,adult).
testdata(1649,3,male).
testlabels(1649,no).
patterns(1650,1,crew).
patterns(1650,2,adult).
patterns(1650,3,male).
labels(1650,no).
%rules
%rules
bestrule(Col,Val,Class,count<Id>) <-
   patterns(Id,Col,Val),labels(Id,Class).
maxclass(Col,Val,C) <-
   bestrule(Col,Val,yes,Y),bestrule(Col,Val,no,N),if(Y>=N then
   C=yes else C=no).

accuracycnt(Col,Val,C,count<Id>) <-
   maxclass(Col,Val,C),patterns(Id,Col,Val),labels(Id,Class),C =
   Class.

accuracy(Column,sum<A>) <- accuracycnt(Column,Value,Class,A).

maxacc(max<A>) <- accuracy(Column,A).
maxcolumn(Column) <- maxacc(X),accuracy(Column,A),A=X.
query maxcolumn(Y).

oner(Id,C)<-testdata(Id,Col,Val), maxcolumn(Column),Col=Column,
```

```
   maxclass(Column,Val,C).

correct(countd<Id>)<-testlabels(Id,L),oner(Id,C),L=C.
total(count<A>) <- testlabels(X,A).
fin_accuracy(Z) <- correct(X),total(Y),Z=(X/Y)*100.
query fin_accuracy(Z).
```

Results of 1R on titanic dataset: 70.21% accuracy.

```
nikita@nikita:~/Documents/CS240/project2/cs240A-DeALS-F16$ time
   bash runfile trial.deal
Executing query 'maxcolumn(Y).'.
Query Id = maxcolumn(Y).1481892616247
Query Form = maxcolumn(Y).
Return Status = SUCCESS
Execution time = 741ms
maxcolumn(3).
1 result

Executing query 'fin_accuracy(Z).'.
Query Id = fin_accuracy(Z).1481893426015
Query Form = fin_accuracy(Z).
Return Status = SUCCESS
Execution time = 16053ms
fin_accuracy(70.2179176755448).
1 result

real 15m21.479s
user 10m22.024s
sys  0m21.352s
```

Task B: DB2

Classification was implemented as follows:

1. Select the dataset and load it into DB2 as a table. This is done by the load.sql file. Loading database is specific to different datasets. Load files for buy_computer, titanic, mushroom and letter datasets can be found at the specified link.

2. Randomly partition DataSet into a TrainSet and a TestSet.
   This is implemented using the rand() function. Train data is assigned three times the number of instances that test data has. This is done by partition.sql This file is also dataset specific. It verticalises the dataset into traindata and testdata.

3. Next phase is actually building the classifier. The probability values are calculated and stored in tables for the test phase. naive_bayes.sql is the script for this. It is a general purpose script and will work for all datasets.

4. For testing the classifier, naive_test.sql is used. It performs the actual prediction for the test data. The predicted values are compared with the actual values to get accuracy of the classifier.

5. oner.sql is the script for the 1R classifier. It finds the column with least error rate for prediction and then assigns predicted class for test data. The script also displays accuracy for the classifier.

6. I have tried boosting by two methods. One is to duplicate incorrectly classified data and the other is to duplicate correctly classified data. The results obtained are summarized in the results section.

The load and partition phases are different for different datasets. Missing data has been ignored for the purpose of this project. All the other scripts are general purpose and will work the same way for all datasets.

Scripts

1. load_titanic.sql

```
--load script for titanic dataset
--titanic data stored as comma separated text
--purpose of load file is load the data into table dataset from
   given data file

CONNECT TO SAMPLE@
DROP TABLE DATASET@
CREATE TABLE DATASET(id integer NOT NULL GENERATED ALWAYS AS
   IDENTITY (START WITH 1, INCREMENT BY 1, NO CACHE ) , shipclass
   varchar (10),age varchar(10), gender varchar (10), class
   varchar (5))@
LOAD FROM '/home/db2inst1/project2/data/data_titanic.data' OF del
   INSERT INTO DATASET(ShipClass, Age, Gender, Class)@
SELECT * FROM DATASET@
CONNECT RESET @
TERMINATE@
```

   The scripts for other datasets can be found at this link.

2. partition_titanic.sql

```
--partition file to partition dataset into test and train
--to be executed after load script
--dataset here is titanic
--two new tables are created for train data nd test data

CONNECT TO SAMPLE@
DROP TABLE TESTDATA@
DROP TABLE TRAINDATA@
CREATE TABLE TESTDATA(instance integer, column integer, val
   varchar(10), class varchar(5))@
CREATE TABLE TRAINDATA(instance integer, column integer, val
   varchar(10), class varchar(5))@
```

```
BEGIN ATOMIC
  FOR temp AS SELECT * FROM DATASET ORDER BY id DO
  IF rand() > 0.75 THEN
    INSERT INTO TESTDATA VALUES
    (temp.id, 1, temp.shipclass, temp.class),
    (temp.id, 2, temp.age, temp.class),
    (temp.id, 3, temp.gender, temp.class);
  ELSE
    INSERT INTO TRAINDATA VALUES
    (temp.id, 1, temp.shipclass, temp.class),
    (temp.id, 2, temp.age, temp.class),
    (temp.id, 3, temp.gender, temp.class);
  END IF;
  END FOR;
END@

SELECT * FROM TRAINDATA@
SELECT * FROM TESTDATA@

CONNECT RESET@
TERMINATE@
```

The scripts for other datasets can be found at this link. .

3. naive_bayes.sql

```
--script is for the training phase of the naive bayes classifier
--probability of each value for each column is calculated per class
--bayes_class table contains total probabilities of each class

CONNECT TO SAMPLE@
DROP TABLE NAIVE_BAYES@
DROP TABLE BAYES_CLASS@
CREATE TABLE NAIVE_BAYES(column integer, val varchar(10), class
   varchar(5), prob decimal(11,05))@
CREATE TABLE BAYES_CLASS(class varchar(5), prob decimal(11,05))@

INSERT INTO NAIVE_BAYES(Column, Val, Class, Prob)
  WITH GroupedData(Column, Val, Class, PSum) AS
    (SELECT Column, Val, Class, count(Column) FROM TRAINDATA
       GROUP BY GROUPING SETS( (Column, Class), (Column, Class,
       Val)))
  (SELECT col_class.column, col_class.val, col_class.class,
     CAST(col_class.PSum AS decimal(4,0))/CAST(class_val.PSum AS
     decimal(4,0))
    FROM GroupedData AS col_class, GroupedData AS class_val
    WHERE col_class.column = class_val.column
    AND col_class.class = class_val.class
    AND col_class.val IS NOT NULL
    AND class_val.val IS NULL)@
```

```
INSERT INTO BAYES_CLASS(Class, Prob)
  WITH
    Total(Val) AS (SELECT count(distinct(instance)) FROM
       TRAINDATA),
    Class(Class, Val) AS (SELECT Class, Count(distinct(instance))
       FROM TRAINDATA GROUP BY Class)
   (SELECT num_class.class, CAST(num_class.val AS decimal(10,0)) /
      CAST(num_total.val AS decimal(10,0)) FROM Total AS num_total,
      Class AS num_class)@

--SELECT * FROM NAIVE_BAYES@
--SELECT * FROM BAYES_CLASS@

CONNECT RESET@
TERMINATE@
```

4. naive_test.sql

```
--script contains test phase of naive bayes
--it is to be executed after naive_bayes.sql
--test_results combines 1.calculation of partial probabilities for
   given testdata 2. adding the values for all the values for
   particular test instance 3. summing log of values according to
   the bayes theorem 4. finding max probability to assign class
   label
--results table stores final accuracy of the classifier =
   100*correct_classified/total instances

CONNECT TO SAMPLE@
DROP TABLE TEST_RESULTS@
DROP TABLE RESULTS@

CREATE TABLE TEST_RESULTS(id integer, actual varchar(5), predicted
   varchar(5))@
CREATE TABLE RESULTS(accuracy decimal(11,05))@

INSERT INTO TEST_RESULTS(id, actual, predicted)
  WITH
    Partial_Probability(id, predicted, probability) AS
      (SELECT test.instance, nbc.class, nbc.prob
        FROM TESTDATA AS test, NAIVE_BAYES AS nbc
        WHERE test.column = nbc.column AND test.val = nbc.val),
    Sum_Probability(sumdatarow, sumpredicted, sumprob) AS
      (SELECT id, predicted, SUM(LOG(probability)) FROM
         Partial_Probability GROUP BY id, predicted),
    Probability(wdatarow, wpredicted, wprob) AS
      (SELECT sp.sumdatarow, sp.sumpredicted,
         sp.sumprob+LOG(nb.prob)
        FROM Sum_Probability AS sp, Bayes_Class AS nb WHERE
           sp.sumpredicted = nb.class),
```

```
    Max_Probability(maxdatarow, maxprob) AS
      (SELECT wdatarow, MAX(wprob) FROM Probability GROUP BY
         wdatarow)
  (SELECT test.instance, test.class, wp.wpredicted
    FROM TESTDATA AS test, Max_Probability AS mp, Probability AS
       wp
    WHERE test.instance = mp.maxdatarow
    AND test.instance = wp.wdatarow
    AND mp.maxprob = wp.wprob
    AND test.column = 1)@

INSERT INTO RESULTS(accuracy) VALUES
  100*CAST((SELECT COUNT(Predicted) FROM TEST_RESULTS WHERE
     Predicted = Actual) AS decimal(10,0))
  /CAST((SELECT COUNT(Predicted) FROM TEST_RESULTS) AS
     decimal(10,0))@

--SELECT * FROM TEST_RESULTS@
SELECT * FROM RESULTS@

CONNECT RESET@
TERMINATE@
```

5. oner.sql

```
--script for 1R classifier
--maxclass is a table which stores count for values of columns
   present per class
--split is the table used to find column with least error count
   using maxclass and traindata tables
--classifier assigns values based on the column attribute with
   least error rate
--results_oner contains accuracy for the classifier =
   100*correctly classified/total instances

CONNECT TO SAMPLE@
DROP TABLE MAXCLASS@
DROP TABLE SPLIT@
DROP TABLE RESULTS_ONER@

CREATE TABLE SPLIT(col integer)@
CREATE TABLE MAXCLASS(column integer, val varchar(10), num_class
   integer, class varchar(5))@
CREATE TABLE RESULTS_ONER(accuracy decimal(11,05))@

INSERT INTO MAXCLASS
  WITH TEMP1 AS(SELECT column, val,COUNT(class) AS num_class,
     class FROM TRAINDATA GROUP BY column,val,class),
  TEMP2 AS(SELECT column,val,MAX(num_class) AS max_num_class FROM
     TEMP1 GROUP BY column,val),
```

```
  TEMP3 AS(SELECT T1.column, T1.val, T1.num_class, T1.class FROM
     TEMP1 AS T1,TEMP2 AS T2
  WHERE T1.num_class=T2.max_num_class AND T1.column=T2.column)
  SELECT * FROM TEMP3@

INSERT INTO SPLIT
  WITH TEMP1 AS(SELECT T.column,COUNT(T.column) AS ERROR
    FROM TRAINDATA AS T,maxclass as f
    where T.column=f.column
    and T.val=f.val
    and T.class!=f.class
    group by T.column ),
  TEMP2 AS(SELECT column,ERROR FROM TEMP1 WHERE ERROR= (SELECT
     MIN(ERROR) FROM TEMP1))
  SELECT column FROM TEMP2@

INSERT INTO RESULTS_ONER VALUES
  100*CAST((SELECT COUNT(DISTINCT T.instance) FROM TESTDATA AS T ,
     SPLIT AS S, MAXCLASS AS F
    WHERE T.column=S.col AND F.column=S.col AND T.column=F.column
       AND T.val=F.val AND T.class=F.class) AS decimal(10,0))
  /CAST((SELECT COUNT(DISTINCT instance) FROM TESTDATA) AS
     decimal(10,0))@

SELECT * FROM MAXCLASS@
SELECT * FROM SPLIT@
SELECT * FROM RESULTS_ONER@

CONNECT RESET@
TERMINATE@
```

6. boost.sql

```
--script for boosting naive bayes
--incorrectly classified instances are duplicated in the training
   data
--this script is to be executed after executing naive bayes
   atleast once
--to check results of boosting, we need to execute naive bayes
   train and test scripts after this
--pre boost and post boost give count of train data before and
   after boosting

CONNECT TO SAMPLE@

SELECT COUNT(*) AS PRE_BOOST FROM TRAINDATA@
INSERT INTO TRAINDATA(SELECT T.* FROM TESTDATA AS T, TEST_RESULTS
   AS R WHERE T.INSTANCE=R.ID AND R.ACTUAL!=R.PREDICTED)@
SELECT COUNT(*) AS POST_BOOST FROM TRAINDATA@
```

```
CONNECT RESET@
TERMINATE@
```

7. boosting.sql

```
--script for boosting naive bayes
--correctly classified instances are duplicated in the training
   data
--this script is to be executed after executing naive bayes
   atleast once
--to check results of boosting, we need to execute naive bayes
   train and test scripts after this
--pre boost and post boost give count of train data before and
   after boosting

CONNECT TO SAMPLE@

SELECT COUNT(*) AS PRE_BOOST FROM TRAINDATA@
INSERT INTO TRAINDATA(SELECT T.* FROM TESTDATA AS T, TEST_RESULTS
   AS R WHERE T.INSTANCE=R.ID AND R.ACTUAL=R.PREDICTED)@
SELECT COUNT(*) AS POST_BOOST FROM TRAINDATA@

CONNECT RESET@
TERMINATE@
```

Results

Results of accuracy for multiple datasets using Naive Bayes and 1R classifiers. Results on titanic show similar performance for both classifiers. The results seem surprising for mushroom dataset. 1R performance is exceptional, the data seems to be highly dependant on a particular column. The classifiers showed accuracies of around 30% for letter dataset. The average of accuracies over 10 passes are as follows.
Titanic: 76.86(NBC), 76.52(1R)
Mushroom: 84.45(NBC), 98.57(1R)

Results of boosting on Naive Bayes are as follows. I have implemented two types of boosting - one which duplicates incorrectly classified instances and another which duplicates correctly classified instances. However the gains were marginal for titanic and mushroom datasets.

Dataset: Titanic and Script: boosting.sql
Iteration 1: 75.82%
Iteration 2: 78.80%
Iteration 3: 78.80%
Iteration 4: 75.49%

Dataset: Mushroom and Script: boost.sql
Iteration 1: 84.07%
Iteration 2: 84.12%

Iteration 3: 84.36%
Iteration 4: 84.22%

For letter dataset however, very high increase in accuracy can be observed. Boosting seems to increase the accuracy of NBC by around 12% from 28.12% to 40.39%.
Dataset: Letter and Script: boosting.sql
Iteration 1: 28.12%
Iteration 5: 32.75%
Iteration 9: 35.02%
Iteration 13: 36.45%
Iteration 17: 37.31%
Iteration 21: 37.78%
Iteration 25: 38.24%
Iteration 29: 38.52%
Iteration 33: 38.86%
Iteration 37: 39.08%
Iteration 41: 39.39%
Iteration 45: 39.71%
Iteration 49: 39.97%
Iteration 53: 40.07%
Iteration 57: 40.25%
Iteration 59: 40.39%