CS 539 Machine Learning

Project 5

Bayesian Networks, Expectation Maximization

Chris Winsor

The Mushroom Dataset

The Mushroom dataset is available from UC Irvine Center for Machine Learning Repository. The dataset is available at http://archive.ics.uci.edu/ml/datasets/Mushroom

The dataset characterizes mushrooms with respect to whether they are edible or poisonous. There are 22 attributes and 8124 samples. There is a single class variable <edible, poisonous>

Attributes describe observable characteristics of the mushrooms, for example cap, stem, population and habitat. All attributes are nominal unordered (categorical).

The dataset does have missing data (stalk-root attribute only) and this is indicated by the value "?".

Guiding Questions

Question 1: Predicting Poisonous based on characteristics of the mushroom.
We investigate if the characteristics of the mushroom can be used to predict whether the mushroom will be poisonous.

Question 2: Is there a common "Species" variable hidden in the data?
Mushrooms, like all plants, come in species[1]. The species of mushroom determines its observable characteristics. Species also determines whether the mushroom is poisonous. These are the attributes and target in our mushroom dataset. As such we suspect there is a dependency relationship where species is the independent variable and characteristics of the mushroom (including poisonous) are the dependent variables.

The Advanced Topic investigates this question. It uses EM (Expectation Maximization) to see if there is a common hidden variable (presumably species). Using EM we would like to discover this hidden variable and then use it to predict if the mushroom will be poisonous.

[1] A good example is the Fisher Iris dataset (Sir Ronald Fisher (1936)). This dataset explicitly identifies three species as the target variable and asks to predict this from attributes which are observable features of the Iris plant.

The Spambase Dataset

The Spambase Dataset originated from the University of California (UCI) Data Repository. It is available at http://archive.ics.uci.edu/ml/datasets/Spambase

The database consists of 4601 samples. There are 57 attributes and one class variable.

48	numeric	real[0..100]	word_freq_WORD	number of times WORD shows up in the email
6	numeric	real[0..100]	char_freq_CHAR	percentage of characters in the email that are CHARl
1	numeric	real[0..100]	capital_run_length_average	average length of uninterrupted sequences of capital letters
1	numeric	integer[1..]	capital_run_length_longest	length of longest sequence of capitals
1	numeric	integer	capital_run_length_total	total number of capitals in the e-mail
1 (class)	nominal	{0,1}	spam	denotes whether the e-mail was (1) or was not (0) spam

Guiding Questions

Question 1: Predicting Spam based on characteristics of the e-mail.

Is it possible to predict whether e-mail is spam, based on characteristics of the mail? We will again use Naive Bayes and Bayes network to do this.

Algorithms and Code - NaiveBayes

We review the Weka implementation of Naive Bayes. Three methods provide the majority of functionality

- buildClassifier() builds the NaiveBayes model
- updateClassifier()   (used by buildClassifier to)
- distributionForInstance() predicts the class probability distribution, given an instance.

NaiveBayes.buildClassifier()

buildClassifier() takes a set of instances, builds the structure Naive Bayes network, and populates the conditional probability tables.

```
buildClassifier(Instances m_Instances) {

    // sanity check the classifier can handle the data
    // clean up data (remove instances with missing class variable)
    // discretize the instances (if requested) using a filter

    // "Reserve space" - this creates instances of the nodes (attribute and class)
    m_Distributions = new Estimator[numberOfAttributes][numberOfClasses];
    m_ClassDistribution = new DiscreteEstimator[numberOfClasses];

    for (each attribute) {
       for (each class variable) {
           // create a node in the Naive Bayes network for this [attribute,class]
           // the object type reflects the nature of the attribute (numeric or nominal)
           // and if numeric, whether kernel or normal estimator is desired

           switch (attribute.type()) {   // based on the type of attribute...

           NUMERIC:   // if the attribute is numeric
              if (useKernelEstimator) {  // and we want to use a kernel estimator
                  // make the node using KernelEstimator subclass
                  m_distributions[attribute][class] = new KernelEstimator();
              } else {
                  // make the node using a NormalEstimator subclass
                  m_distributions[attribute][class] = new NormalEstimator();
              }

           NOMINAL:  // if the attribute is nominal
              // make the node using DiscreteEstimator subclass
              m_distributions[attribute][class] = new DiscreteEstimator();
           }
       }
    }
    // "Compute counts" - use the data instances to generates CPTs
    for (each m_Instances) {
       updateClassifier(instance);  // described below
    }
}
```

NaiveBayes.updateClassifier()

updateClassifier() takes a single data instance and updates the conditional probability tables for the entire network.

```
updateClassifier(Instance instance) {

       // loop through the attributes
       for (each attribute) {

           // perform an "addValue" to the node representing this attribute
           // this adds the weighted value of the sample value to the CPT
           m_Distributions[attribute][class].addValue(instance.value(attribute),
                                                       instance.weight());
       }
       // perform an "addValue" to the node which represents the class
       m_ClassDistribution.addValue(instance.classValue(),
                                   instance.weight());
    }
}
```

NaiveBayes.distributionForInstance()

distributionForInstance() calculates the class probabilities, given a data sample. This is where Bayes theorem is applied (prior times the conditional results in the posterior).

```
// create an array of probabilities (one for each target class)
double [] probs = new double [m_NumClasses];

// start the class node with the prior probability
// the prior probability comes from the samples of  the class variable itself
for (int j=0 to numClasses) {
       probs[j] = m_ClassDistribution.getProbability
}

// loop through each attribute node in the Naive Bayes network
for (each attributeNumber) {
// and then through each class node in the Naive Bayes network
       for (each classNumber j) {
// we calculate the conditional probability output by this attribute node
// it is calculated as the node raised to the power of that node's weight
       temp = max(1e-75,
pow(m_Distributions[atttibuteNumber][classNumber].getProbability(instance.value(attribute)),
       m_Instances.attribute(attIndex.weight()));
// Apply Bayes theorem
// the posterior probability of the class node
// is its prior times the conditional probability
// the network is Naive Bayes which assumes each attribute is independent
// so it is the prior times each of the attribute's conditional
// we do this once for each attribute node (each attribute feeds into the class).
probs[j] *= temp;
}
}
```

Algorithms and Code - BayesNet

We describe Weka's NaiveBayes classifier.

Classes for establishing the structure of a Bayes Network are
- class SearchAlgorithm;
- class LocalScoreSearchAlgorithm extends SearchAlgorithm;
- class K2 extends LocalScoreSearchAlgorithm;

Classes for estimating conditional probability tables (CPTs) for nodes in the Bayes Network once structure is known are:
- class BayesNetEstimator;
- class SimpleEstimator extends BayesNetEstimator;

Then there is the "BayesNet" class, which brings the elements above together and runs them.
- class BayesNet extends AbstractClassifier;

I will first review the overall Bayesnet class, then K2::search() and then SimpleEstimator::distributionForInstance();

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
xxxxxxxxxxx class BayesNet  xxxxxxxxxx;
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
// this has
SearchAlgorithm m_SearchAlgorithm = new K2();
BayesNetEstimator m_BayesNetEstimator = new SimpleEstimator();

// method to build the network
BayesNet::buildClassifier(Instances instances) {
    // build the network structure
    m_SearchAlgorithm.buildStructure();  // calls SearchAlgorithm.buildStructure
    // build the Conditional Probability Tables
    estimateCPTs();
}

// method to use the network...
BayesNet::distributionForInstance() {
    // this calls SimpleEstimator::distributionForInstance, described below
    return m_BayesNetEstimator.distributionForInstance(this,instance);
}
```

K2::search()

K2::search() builds the structure of the network. It does this by tentatively adding parents to each node and re-scoring the network to see if that parent has improved the network's performance.

```
K2::search(BayesNet bayesNet, Instances instances) {

    // create an ordered list of attributes
    // start with the order the attributes show up in the instance data
    for (each attribute iOrder in the instance data)
       nOrder[iOrder] = nAttribute++;

    // optionally randomize that order if the user requested it

    // prepare to 'score' the network by establishing base score (with no parents)
    foreach (attribute) {
       baseScore[attribute] = calcNodeScore(attribute);
    }

    // perform the K2 algorithm of searching for the best scoring set of parents
    foreach (attribute att, in the order chosen above) {
       bestScore = baseScore[attribute];  // the zero-parent score
       // while the node has fewer parents than the max
       bProgress = (the number of parents of this attribute is less than max parents);
       while (bProgress) {
           for (each attribute att2 that is ordered earlier than att) {
              // get a score with the parent added
              fScore = calcScoreWithExtraParent(att, att2);
              // compare score with additional parent to the baseline and take best
              if (fScore > bestScore)
                 // note that attribute and score
           }

           // if found parent which improves the score add it to this node's parent list
           if (nBestAttribute != -1) {
              basesNet(attribute).addParent(bestAttribute);
              // see if we are still below the max number of parents
              bProgress = (bayesNet(attribute).nbrOfParents() < maxParents())
            } else {
              // we have found that no parents improve the score
              // so no further addition of parents is considered
              bProgress = false;
           }
       }
    }
```

SimpleEstimator::distributionForInstance()

SimpleEstimator::distributionForInstance() is the means of using the net. Given an instance it uses the net to generate the class distribution.

```
SimpleEstimator::distributionForInstance(BayesNet bayesNet, Instance instance) {

    for (each class) {  // initialize the final probabilities
       fProbs[class] = 1.0;
    }

    for (each class) {
       for (each attribute in the given data instance) {
           for (each parent of the node for that attribute) {
              // calculate the index into the conditional probability table
              if (parent == class) {
                  iCPT = iCPT * nNumClasses + iClass;
              } else {
                  iCPT = iCPT * instancess.attribute(nParent).numvalues() + 
instance.value(nparent);
              }
           }

           // once we have the index into the CPT we calculate the output probability
           // the technique add the logs of the conditional probabilities
           // note the exp of the sum of the logs is the same as finding the product
           if (iAttribute == instances.classIndex()) {
              logfP += log(bayesNet.m_Distributions[iAttribute][iCPT].getProbablity(iClass));
           } else {
              logfP += 
log(bayesNet.m_Distributions[iAttribute][iCPT].getProbability(instance.value(iAttribute)));
           }
           fProbs[iClass] += logfP;
       }

       // go from log-space to normal-space
       for (each class)
           fProbs[iClass] = exp(fProbs[iClass] - fMax);

       // normalize and return
       normalize(fProbs);
       return fProbs;
    }
}
```

Preprocessing the Mushroom Dataset

Attributes
All attributes are nominal unordered (categorical). Bayes networks support this type of data directly so no

preprocessing is required.

The dataset does have some missing attribute values. The preprocessing here converts the "?" data value to "no_data".

<u>Class Variable</u>
The target class is a binary nominal and is suitable as-is for the Bayes networks.

Preprocessing the Spambase Dataset

<u>Attributes</u>
The original dataset has 57 attributes.
They are numeric values, unbinned.
There is a lot of cross-correlation between attributes.

Preprocessing Steps::

- Remove highly correlated attributes
- Bin the remaining attributes

<u>Class Variable</u>
There is one class variable which identifies if the mail is spam or not. No preprocessing is required here.

Performance Metrics

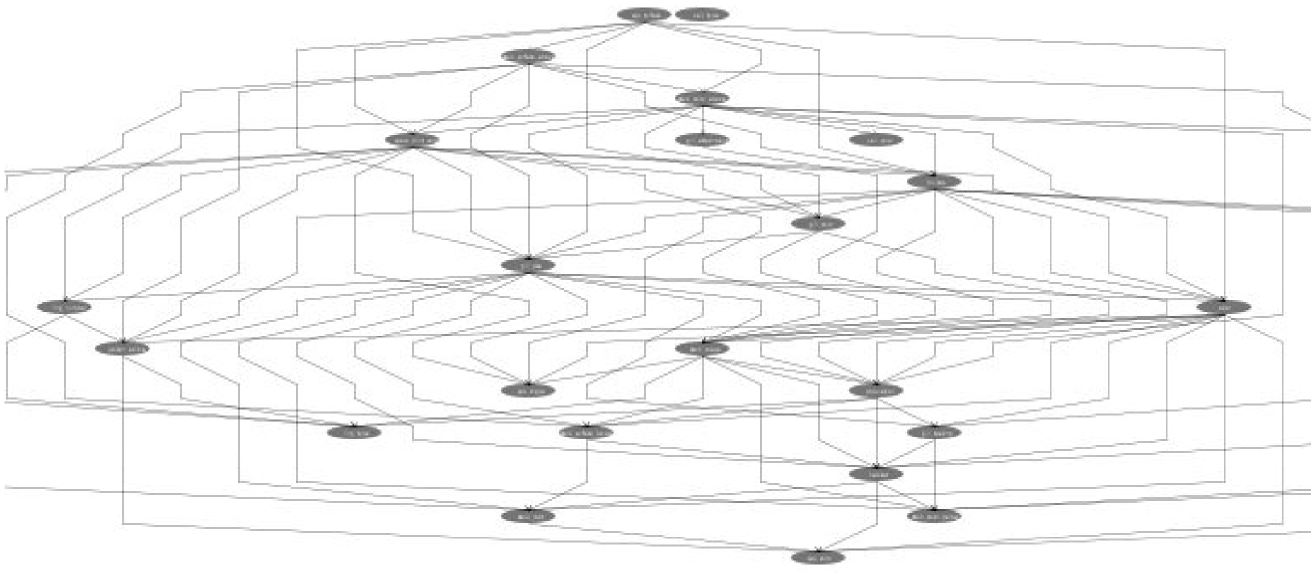
General Comparisons

Training Time: Neural network is characterized as by far the longest training time, between 4.7 and 6.1 seconds. All others (including Naive Bayes and Bayes Net) had training time in the tens of milliseconds.

Accuracy: Experiment 2 explores accuracy in detail, but it is noteworthy that J4.8 and the Neural Network achieved 100% accuracy on the testing data while Naive Bayes and Bayes Net achieved 95% to 96% accuracy.

Leaves and Rules: J4.8 had 23 to 24 leaves, and a total of 23 to 24 rules.

Structure of the Bayes Net: When Bayes Network was run with 10 max parents and random attribute order it would generate a structure something like the following. The intuitive understandability of the relationships is low.



Accuracy - Naive Bayes

Paired T-Testing compared Naive Bayes to ZeroR, OneR, J4.8, Neural Network, and Bayes Network. In each case a 75% / 25% split was used (75% of the data for training and 25% of the data for testing). The experiment was performed 10 times. Algorithms were run with the default settings in all cases except Neural Network which was set to single-stage (zero hidden nodes) with training_time parameter of 100. This was done to keep training time reasonable.

Results: Naive Bayes, with an accuracy of 95.43%, is statistically more accurate than ZeroR, and statistically less accurate than all other methods.

```
Dataset               (1) bayes.Na | (2) rules (3) rules (4) trees. (5) functi (6) 
bayes
--------------------------------------------------------------------------------
--
agaricus-lepiota-wit (10)   95.43 |   51.80 *   98.55 v   100.00 v   100.00 v   96.11 
v
--------------------------------------------------------------------------------
--
                          (v/ /*) |   (0/0/1)   (1/0/0)    (1/0/0)    (1/0/0)    
(1/0/0)

(1) bayes.NaiveBayes '' 5.9952312017856973E18
(2) rules.ZeroR '' 4.8055541465867952E16
(3) rules.OneR '-B 6' -2.4594270021478615E18
(4) trees.J48 '-C 0.25 -M 2' -2.17733168393644448E17
(5) functions.MultilayerPerceptron '-L 0.3 -M 0.2 -N 100 -V 0 -S 0 -E 20 -H 0' 
-5.9906078170482104E18
(6) bayes.BayesNet '-D -Q bayes.net.search.local.K2 -- -P 1 -S BAYES -E 
bayes.net.estimate.SimpleEstimator -- -A 0.5' 7.4603744325877594E17
```

Accuracy - Bayes Net

Paired T-Testing compared Bayes net to ZeroR, OneR, J4.8, Neural Network, and Naive Bayes. The dataset, split, and parameters were the same as the prior experiment.

Results: Bayes Net, with an accuracy of 96.11% is statistically more accurate than ZeroR and Naive Bayes, and statistically less accurate than OneR, J48 and Neural Network.

```
Dataset               (1) bayes.Ba | (2) rules (3) rules (4) trees. (5) functi (6) 
bayes
--------------------------------------------------------------------------------
--
agaricus-lepiota-wit  (10)   96.11 |   51.80 *   98.55 v   100.00 v   100.00 v   95.43 
*
--------------------------------------------------------------------------------
--
                          (v/ /*) |   (0/0/1)   (1/0/0)    (1/0/0)    (1/0/0)    
(0/0/1)
```

```
(1) bayes.BayesNet '-D -Q bayes.net.search.local.K2 -- -P 1 -S BAYES -E
bayes.net.estimate.SimpleEstimator -- -A 0.5' 7.4603744325877594E17
(2) rules.ZeroR '' 4.805541465867952E16
(3) rules.OneR '-B 6' -2.4594270021478615E18
(4) trees.J48 '-C 0.25 -M 2' -2.1773316839364448E17
(5) functions.MultilayerPerceptron '-L 0.3 -M 0.2 -N 100 -V 0 -S 0 -E 20 -H 0'
-5.9906078170482104E18
(6) bayes.NaiveBayes '' 5.9952312017856973E18
```

Algorithm Options

Naive Bayes with, and without, Supervised Discretization

In this experiment we compare performance of Naive Bayes when run with, and without, the Supervised Discretization option. To do this we used the original spambase dataset (no discretization or attribute selection). We ran a 75% / 25% split (10 iterations). We then performed Paired T-Tester analysis with a 95% confidence interval.

Results: We conclude there is a significant improvement in performance with the Supervised Discretization option.

```
Dataset                    (1) bayes.Na | (2) bayes
--------------------------------------------------
spambase                   (10)   79.84 |   90.45 v
--------------------------------------------------
                                (v/ /*) |   (1/0/0)

(1) bayes.NaiveBayes '' 5.9952312017856973E18
(2) bayes.NaiveBayes -D 5.9952312017856973E18
```

Naive Bayes with, and without, Feature Selection

In this experiment we compare Naive Bayes with and without Feature Selection. We the spambase dataset that has been preprocessed in two different manners - the first has been discretized but not feature-selected, and the second has been both discretized and feature-selected. Feature selection was done using CfsSubsetEval.

Results: There is a slight improvement in performance observed when feature selection is used.

	Accuracy
Without Feature Selection	90.86%
With Feature Selection	92.96%

Bayes Net with, and without, Net Initialization as Naive Bayes

In this experiment we investigate initialization of the Bayes net where the choices are to initialize as Naive Bayes or not. We use both the mushroom dataset and the spambase dataset. We keep maxNumberOfParents consistent at 5, and randomOrder at TRUE. We run 10 iterations of a 75%/25% split. We then perform Paired T-Testing with a 95% confidence level.

Results: We can NOT conclude there is a significant difference between Bayes Net run initialized as Naive Bayes, or not.

```
Dataset                    (1) bayes.Bay | (2) bayes.
----------------------------------------------------
'spambase-weka.filters.su (10)    93.85 |    92.81
agaricus-lepiota-with-att (10)   100.00 |   100.00
----------------------------------------------------
                                (v/ /*) |    (0/2/0)

(1) bayes.BayesNet '-D -Q bayes.net.search.local.K2 -- -P 5 -R -S BAYES -E
bayes.net.estimate.SimpleEstimator -- -A 0.5' 7.4603744325877594E17
(2) bayes.BayesNet '-D -Q bayes.net.search.local.K2 -- -P 5 -N -R -S BAYES -E
bayes.net.estimate.SimpleEstimator -- -A 0.5' 7.4603744325877594E17
```

Bayes Net with Different Search Methods

Here we investigate three search methods (K2, Hillclimber, Repeated Hillclimber). We use the mushroom dataset and the spambase dataset. We keep maxNumberOfParents consistent at 5, and randomizeOrder at TRUE. We run 10 iterations of a 75%/25% split. We then perform Paired T-Testing wtih a 95% confidence level.

Results: We can NOT conclude that Bayes predicts with a higher level of accuracy using any of the techniques described. This holds for both the mushroom and the spambase datasets.

```
Dataset                    (1) bayes.Bay | (2) bayes (3) bayes.
--------------------------------------------------------------
'spambase-weka.filters.su (10)    92.53 |   93.91      93.84
agaricus-lepiota-with-att (10)   100.00 |   99.97     100.00
--------------------------------------------------------------
                                (v/ /*) |   (0/2/0)    (0/2/0)

(1) bayes.BayesNet '-D -Q bayes.net.search.local.K2 -- -P 5 -N -R -S BAYES -E
bayes.net.estimate.SimpleEstimator -- -A 0.5' 7.4603744325877594E17
(2) bayes.BayesNet '-D -Q bayes.net.search.local.HillClimber -- -N -P 5 -S BAYES -E
bayes.net.estimate.SimpleEstimator -- -A 0.5' 7.4603744325877594E17
(3) bayes.BayesNet '-D -Q bayes.net.search.local.RepeatedHillClimber -- -U 10 -A 1 -N
-P 5 -S BAYES -E bayes.net.estimate.SimpleEstimator -- -A 0.5' 7.4603744325877594E17
```

Summary of Results

Our testing on the mushroom dataset showed that both Naive Bayes and Bayes Network were statistically less accurate than OneR, J48, and Neural Network. Still - both Naive Bayes and Bayes Net achieved over 95% prediction accuracy which is very respectable, especially for Naive Bayes with its simple algorithm and structure.

Although Bayes Network was accurate in its prediction, the network's structure provided no real insight into the mushrooms., or the relationship between (say) Cap Shape and Gill Size.

Advanced Topic

Overview:

For an Advanced Topic we investigate the mushroom dataset using EM (Expectation Maximization). We want to see if we can discover a common hidden variable in the mushroom dataset. We have suggested this hidden variable might represent the mushroom species.

EM uses an iterative procedure to determine the number of clusters (N) and means and standard deviation for each:

```
–E-step
        Compute p_ij = P(C=i|x_j)
        This is the probability that data x_j was generated by cluster i
–M-step
        Compute new means, covariance and weights

        u_i =   sum_over_j(  p_ij   *   x_j   /   p_i   )
        cov_i = sum_over_j(  p_ij  *  x_j  *  x_j  /  p_i  )
        w_i = p_i
```

(Equations above are from Russell/Norvig, Second Edition, Page 727)

Procedure:

Using the full Mushroom dataset we allow Weka's EM (weka.clusters.EM) to create a model. We specified 6 clusters, this number came from experimentation on this dataset and seemed to be a reasonable number. Once the model was generated we then applied the data instances and had the model output the logDensityPerCluster for each instance. The logDensityPerCluster is a measure of the liklihood that instance is part of that cluster.

```
EM.logDensityPerClusterForInstance(dataPrimary.instance(line));
```

The result is 6 logDensity values (one for each cluster) for each data instance. We then appended the target class (poisonous) to each instance. This formed the input to the Naive Bayes network. A subset of the data is as follows:

Species 1	Species 2	Species 3	Species 4	Species 5	Species 6	Poisonous
-49.303	-15.085	-63.013	-70.275	-68.998	-106.177	p
-45.839	-13.686	-54.319	-61.121	-85.319	-92.585	e

<etc>

We then ran Naive Bayes using the above data such that the class (poisonous) would be predicted based on the attributes (species). The following are the results

```
=== Run information ===
Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     em_output_hidden_class_values_run2_no_attributes
Instances:    8124
Attributes:    7
              Species 1
              Species 2
              Species 3
              Species 4
              Species 5
              Species 6
              Poisonous
Test mode:    10-fold cross-validation
=== Summary ===
Correctly Classified Instances          7300                89.8572 %
Incorrectly Classified Instances        824                 10.1428 %
Total Number of Instances               8124
=== Confusion Matrix ===
      a    b   <-- classified as
 3111  805 | a = p
   19 4189 |    b = e
```

Conclusion

The accuracy of the combined EM and Naive Bayes procedure is lower than that of all other techniques (except ZeroR).   But a significant distinction is that this model introduces a hidden variable (species) which we believe is hidden in the data and is the 'cause' of the observed characteristics of the mushroom.