CS 374 Assignment #2
Bayesian Decision Theory and the Naive Bayes Classifier
Due the week of March 1, 2021

—

This week's topics are Bayesian Decision Theory and the Naive Bayes Classifier. The Naive Bayes classifier is a simple probabilistic classifier that gets its name (i.e., "naive") from the fact that it makes a very strong assumption: that there are no dependencies among the attributes of an example, given the class. Despite its simplicity, the Naive Bayes classifier often proves to be quite effective in practice.

This week you will have the opportunity to explore Naive Bayes through a variety of different types of exercises.

1 Bayesian Decision Theory

1.1 Reading

Please read Chapter 3, Sections 3.1-3.4 in Alpaydin. (Both editions 3 and 4 are fine.3 This chapter gives an introduction to probabilistic reasoning, much of which is likely to be familiar to you. In particular, it introduces Bayes' Rule in the context of classification. Bayes' Rule allows us to compute the probability that an example belongs to one of several classes, thereby allowing us to select the classification that is most appropriate.

In certain contexts, we will see that it is best to choose the class with highest probability. In other cases, we might combine this with other information, such as the cost of making a classification error.

1.2 Exercises

Please do the following exercises. I hope you find them to be quite easy. If you have any trouble at all, let me know.

- (Not to turn in – for your understanding only) Exercise 4 on page 63 of Alpaydin (fourth edition) or page 61 (third edition).
- (To turn in) Exercise 6 on page 64 of Alpaydin (fourth edition) or page 62 (third edition).

In the tutorial session, I will expect you to introduce Bayes' Rule and discuss how it can be used for classification. Then explain why errors in classification can have different cost and how that might impact the way we make classification decisions. You might incorporate your solution to exercise 6 into your presentation.

2 Implementing a Naive Bayes Classifier

We now turn to the Naive Bayes classifier, which makes the conditional independence assumption stated in the introduction above. For this, you'll read sections of a chapter written by Mitchell as an addendum to his textbook. (The link is available from the course assignments web page. The reading is also available from the Glow module for Week 2.) Specifically, read through Section 2.

One of the best ways to understand an algorithm is, of course, to implement it. So your next task will involve implementing a Naive Bayes classifier. This type of classifier requires a great deal of information about the domain of examples to be classified, including:

- prior probabilities of the classes;
- likelihoods, i.e., conditional probabilities of the attribute values given the classes.

This is where the "learning" comes in. The information must be estimated from sample data.

2.1 Reading

Begin by reading Section 3.4.2 on page 59 of Mitchell (i.e., the actual textbook, not the chapter you read above). This describes a specific classification domain. (Yes, it's contrived, but it's a "classic", especially when the game to be played is golf, rather than tennis.)

Next read Section 4.2 in Witten and Frank. This section is on Naive Bayes learning and classification, and it very nicely describes the process of estimating probabilities from training data and then using these probabilities to classify new examples (in case it wasn't already clear from reading Mitchell).

2.2 Exercise 1

Implement a Naive Bayes learner and classifier. For this exercise, you should assume that all attributes have nominal (i.e., discrete) values. You need not handle real-valued attributes.

Your program should begin by reading a file called *domainName*.arff, where *domainName* is the name of the domain of application. I will provide you with four such files:

- weather.nominal.arff - the weather domain described in Mitchell and also in Witten and Frank;
- contact-lenses.arff - given a description of a patient, determine the type of contact lenses that should be prescribed;
- soybean.arff - given a description of a soybean plant, determine the type of disease it has;
- titanic.arff - given information about a passenger on the Titanic, predict whether that person would have survived or not.

You can copy these files from

```
~andrea/shared/cs374/NominalData
```

or from the Glow module for Week 2. Note that this is a standard format of files that we will be experimenting with later in the semester.

Each file begins with a preamble that describes the domain. This is followed by a section that specifies the attributes and the possible values for each attribute. The final attribute listed is the class to be predicted. Each attribute is on a separate line and has the following form:

```
@attribute outlook {sunny, overcast, rainy}
```

It begins with the string "@attribute". This is followed by the attribute name. The possible attribute values are listed in curly braces, separated by commas. Please be careful when reading in this information. In particular, the word "attribute" is sometimes uppercase and at other times lowercase. Attribute values might be separated by commas only, or they may be separated by commas and blanks. Look at all of the data files to get a sense of the variety.

Each file also contains data – i.e., specific examples and their associated classifications. The data part of the file begins with the line

```
@data
```

Each line of data has the form:

```
sunny,hot,high,FALSE,no
```

This gives the values for each of the attributes defined earlier in the file. The final value is always the class value. Again, be careful about format. Sometimes examples are separated by commas only; sometimes they are separated by commas and blanks.

There are times when the values for certain attributes are unknown. These are indicated by "?". While there might be missing attribute values, you can assume that all class values are fully specified.

Once you have read in all of the data, you will need to train and test a Naive Bayes classifier. For now, assume you have divided the data in the arff file into training and test sets. Use the training data to calculate the prior probabilities and likelihoods that are necessary for classifying new examples. You should implement the Laplace estimator as described in Witten and Frank. (It's fine to do the estimation suggested on page 92.) This is especially important for the soybean domain.

Once the training is complete, you should classify the test examples. The output should include the number of test examples classified correctly, the number classified incorrectly, and the percent accuracy on the test set.

Now let's consider two different ways to treat your arff data as training and test sets.

- Use the entire set of data for both training and testing. This is not the right way to estimate the performance of your classifier on unseen examples, but it does provide useful information. It gives you a sense of how well the classifier reflects the data on which it was trained.
- Partition your data into training and test sets. This provides a much better estimate of what you can expect on previously unseen examples. There are several ways to do this. Since two of our data sets are very small, I would like you to perform a "hold-one-out" test. Say you have n examples in your data set. Train n classifiers, each time holding out one of the examples as a test example and training on the rest. For each of the n classifiers, keep track of whether it correctly or incorrectly classifies its held-out example. That, then, is the estimate of what you can expect on unseen examples.

Perform both of these kinds of tests for an input data set. That is, say how your classifier does when tested on its own training set, and say how it performs on a hold-one-out cross-validation.

You may implement your program in Java or Python, but regardless of the language you choose:

- If your code is written to be opaque, you will have a much more difficult time explaining it in the tutorial session. Neat, clean, general, transparent code is always best.
- Your code must compile and run on the Unix machines in the lab. If you are coding in Python, you may want to use standard libraries. These may already be installed. If not, specify in your instructions (see below) what I will need to have to run the code myself.
- You must include brief instructions on how to run your code (especially spelling out any parameters that need to be supplied, as well as their format).
- You must be able to demonstrate the running code during the tutorial session, ideally from your laptop. Be sure you have it available and ready to run.

Follow the turn-in link to turn in the code in a single file (i.e., zipped an tar'd if you have more than one file to turn in).

During the tutorial session I will expect you to outline the design of your code, explain your design decisions, and demonstrate how it works. Don't be surprised if I give you a new arff file to test during our session.

2.3 Exercise 2

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others.

In this exercise, you will consider whether any of the information learned by your Naive Bayes classifier can be used to shed light on what sorts of people were likely to survive.[1]

The data I provide are a subset of the Kaggle training data. The attributes include:

- passengerClass – 1st, 2nd, or 3rd.
- sex.
- age, discretized into spans of 10 years.
- sibSp – if the passenger had a spouse or siblings aboard, this is the total.

[1] This exercise is adapted from an exercise by Jessica Wu at Harvey Mudd College, which she adapted from the Kaggle Titanic competition.

- paCh – if the passenger had parents or children aboard, this is the total.
- embarked – this is the passenger's point of embarkation.

Passengers with missing values for any feature have been removed.

Consider the probabilities/likelihoods learned by your Naive Bayes classifier. (Print them out.) Is there any information in there that helps shed light on the factors that contributed to a passenger's survival? Write up your observations. During the tutorial session, be prepared to present your observations and to back them up with data from your classifier. What are the limits to what you can extrapolate from the classifier?

You do not need to turn in your analysis. Instead, organize your data and thoughts so you can present both during the tutorial session. You could even consider a few slides, though this isn't strictly necessary (just an idea for how you might organize your disucssion).

3 Optional: Error of Bayesian Classifiers

Next, you will switch gears and do a bit of theoretical work involving lower and upper bounds on the error of the Bayes decision rule. This will also take you out of the realm of nominal-valued data and into real-valued data. Note that this section of the assignment is optional. You may skip it and go on to the next section, if you would like.

3.1 Reading

Read Section 2.1 of Duda, Hart, and Stork. The first part of this section repeats some of what you learned from the reading in Alpaydin.

3.2 Exercises

(Note that the following is problem 1 on page 65 of Duda, Hart, and Stork.)

In the two-category case, under the Bayes decision rule the conditional error is given by Eq. 7. Even if the posterior densities are continuous, this form of the conditional error virtually always leads to a discontinuous integrand when calculating the full error by Eq. 5.

- Show that for arbitrary densities, we can replace Eq. 7 by $P(error|x) = 2P(\omega_1|x)P(\omega_2|x)$ in the integral and get an upper bound on the full error.
- Show that if we use $P(error|x) = \alpha P(\omega_1|x)P(\omega_2|x)$ for $\alpha < 2$, then we are not guaranteed that the integral gives an upper bound on the error.
- Analogously, show that we can use instead $P(error|x) = P(\omega_1|x)P(\omega_2|x)$ and get a lower bound on the full error.
- Show that if we use $P(error|x) = \beta P(\omega_1|x)P(\omega_2|x)$ for $\beta > 1$, then we are not guaranteed that the integral gives a lower bound on the error.

If you complete this optional part of the assignment, be prepared to present your solution to this problem during tutorial.

4 Not Optional: Logistic Regression

Finally, you will learn about logistic regression and its relationship to the Naive Bayes classifier.

4.1 Reading

For this you'll return to the "out of text" Mitchell chapter. Please read Section 3, but only through 3.1. Feel free to read beyond this, but you'll only need these pages to do the exercise below.

4.2 Exercise

In the tutorial meeting, I will expect you to give a high-level explanation of logistic regression. (Please don't let the term "regression" in the name confuse you. Be sure you understand the first paragraph in the reading.) What is a fundamental difference between Naive Bayes and Logistic Regression? How are they related? For this final question, do Exercise 3 on page 16 of the Mitchell "out of text" chapter. Be prepared to present it in tutorial, as well as to turn it in.

5 What you will be turning in this week

In summary, please turn in

- A single pdf with solutions to your problem set questions. These are
  - Exercise 6 in Chapter 3 of Alpaydin.
  - The optional exercise from Duda, Hart, and Stork (if you did it).
  - Exercise 3 on page 16 of the "out of text" Mitchell chapter.
- All of your code to read data, run Naive Bayes, and test your classifier. This should be in a single tar'd and compressed file.

To turn these in, follow the link on the course website or in the Glow module for Week 2.