6CCS3ML1 (Machine Learning)

Practical 4: Naïve Bayes and K-means

(Version 2.2)

1 Overview

This practical builds on Lectures 3 where we discussed several learning mechanisms that are based on probability theory. Here you will program two of these. First you will build a Naïve Bayes model, and then you will re-use your K-means implementation on another dataset. For this practical we will move beyond the simple datasets in `scikit-learn`, and start using the UCI Machine Learning Repository.

You will also make use of some elements of `scikit-learn`, some new and some you have met before.

2 Getting started

Start by downloading `bayes-hd-starter.py`. This is a simple skeleton which loads a dataset. The dataset to use is also on KEATS. It is in the file `processed.cleveland.data`, so you should download that data as well, and place it in the same folder as `bayes-hd-starter.py`.

The dataset is part of the Heart Disease Data Set from the UCI Machine Learning Repository at:

```
https://archive.ics.uci.edu/ml/datasets/Heart+Disease
```

Open the above URL in a browser, because you will need to refer to it.

Now run `bayes-hd-starter.py`. It should load the data, and report how many complete records it finds. I got the output:

```
Number of samples:  297
```

Take a look at the code in `bayes-hd-starter.py`. The data in `processed.cleveland.data` is in CSV form (you can look at it in an editor or, more helpfully, in a spreadsheet since CSV is a standard format understood by spreadsheets). Because of this, it is a bit more complicated to load the data than it is to load a `scikit-learn` dataset. However, once the data is loaded, you can use it exactly like you used the `scikit-learn` datasets. The features are stored in variable `data`, the diagnosis is stored in the variable `target`.

3 A quick calibration

The full Heart Disease Data Set contains a large set of attributes measured from several groups of medical patients who were tested for Heart Disease. The dataset gives both the attributes for each patient, and the diagnosis, in the form of what stage of heart disease is present. This is therefore a classification problem. We want to predict the diagnosis from the attributes. As a benchmark for the classifier you will write, first build a decision tree and test that:

1. Split the dataset to use 80% for training and 20% for testing.

   See the instructions for Lab 2 if you can't remember how to do this.

2. Build a decision tree using the training data and test it using the test data.

   See the instructions for Lab 2 if you can't remember how to do this.

3. Output the score of the decision tree. I got:

   ```
   Decision tree score:  0.58
   ```

The low score from the decision tree tells you that this is not an easy problem. Looking at the data we can see that the diagnosis has 5 possible values:

```
0, 1, 2, 3, 4
```

0 indicates no heart disease, and 1, 2, 3, and 4 indicate various stages of the disease. A classifier that was randomly guessing would get 20% of its classifications right, on average. So, while the decision tree doesn't do well, it does a lot better than random.

The score from the decision tree gives you something to aim for: can you build a classifier that does better?

4 Naïve Bayes

Now you should write a Naïve Bayes classifier for the Heart Disease data.

To do this, you should follow the process laid out in Lecture 3. We know that when we have complete data, we can establish the maximum likelihood estimates by counting.

So, for example, we can establish the prior probability that the diagnosis, attribute `num`, will have value `0`, indicating no disease, by counting the proportion of instances in the training data. In other words, first go through the training data, checking when the target takes value `0`, and counting the cases when it does. We then divide our count by the number of training examples, and the result is:

$$p(num = 0)$$

We then repeat this for the remaining values of num.

Then, we use the same kind of method to get conditional probabilities. For example:

$$p(fbs = 0|num = 0)$$

is the proportion of training examples where $num = 0$ where it is also the case that $fbs = 0$. In other words, you count the number of cases where $num = 0$ and $fbs = 0$, and then divide by the number of cases where $num = 0$. To get the complete distribution, you need to do this for all values of num and fbs. To find the values of fbs and other features, consult:

```
https://archive.ics.uci.edu/ml/datasets/Heart+Disease
```

This is exactly the process we saw in Lecture 3 for the tennis example.

To build a decent classifier, you will need to establish conditional probabilities for several features. I suggest that for now you restrict yourself to features that have discrete values. These are the features:

`sex`, `cp`, `fbs`, `restecg`, `exang`, `slope`, `ca`

Once you have the conditional probabilities, you can compute the most likely value of `num` as described in the slides for Lecture 3. And you can test how well your classifier performs. Computing the score as the proportion of times that the classifier made the correct diagnosis, I got:

```
My Naive Bayes scores:  0.53
```

The lower result is most likely because I wasn't using the full set of features, while the decision tree was using them all.

5 Compare against `scikit-learn`

Naturally `scikit-learn` includes an implementation of Naïve Bayes. Indeed, it contains several:

```
http://scikit-learn.org/stable/modules/naive_bayes.html
```

Try using the "Multinomial Naive Bayes" on the Heart Disease data. This is basically what you have implemented, with a trick to avoid overfitting when there is not much data.

How does this perform?

I got:

```
Naive Bayes score:  0.6
```

which is better than the decision tree and my implementation. `:-)`

6 K-means redux

Now that you know how to load data in CSV format, you can try your K-means implementation out on more interesting data than the Iris dataset. For example, try the `StoneFlakes.dat` dataset that is on KEATS.

This is another set from the UCI Machine Learning Repository, and the page describing the data is here:

```
https://archive.ics.uci.edu/ml/datasets/StoneFlakes
```

You should pick a set of features to work with, and experiment with different numbers of clusters to find a good clustering. You might want to use the technique from `classify-iris.py` (available on KEATS under Practical 1) to plot various pairwise combinations of attributes to do this.

Since the data does not have ground truth labels — unlike the Iris dataset, there is nothing to say which class/cluster an element should be in — you won't be able to use the Rand index to assess how good the clusters are. Instead, if you want something suitable that is implemented in `scikit-learn` you will have to use the Silhouette Coefficient or the Calinski-Harabaz index. See the bottom of this page:

```
http://scikit-learn.org/stable/modules/clustering.html
```

for documentation. You could, of course, implement the Davis-Bouldin index (see Lecture 1). This fits well with K-means since it uses the distance of data points to the cluster centre, which is, of course, what you use to cluster the points in the first place.

7 More

If you want to try other things:

1. If you didn't already, modify your Naïve Bayes classifier to ensure that it doesn't overfit in cases where there are no examples of particular combinations of feature values. In other words, make sure that none of the conditional probability values are zero. You can do this by applying the method on slides 57 and 58 of Lecture 3.

2. Investigate how your Naïve Bayes classifier varies in performance with different subsets of features.

   What is the highest performance that you can obtain?

   Which set of features gives you this result?

3. If you look at the documentation on:

   `https://archive.ics.uci.edu/ml/datasets/Heart+Disease`

   you will see that it says that most work on the heart disease data has only tried to predict whether heart disease is absent (`num = 0`), or present (`num > 0`). This should be an easier problem than the one that you solved with your Naïve Bayes classifier.

   Modify your Naïve Bayes classifier to solve this simpler problem.

   How well does it perform?

4. When we picked features to use for the Naïve Bayes classifier, we chose attributes that had discrete values. There are several other attributes that use continuous values. You can turn these into discrete values by "discretising" them. That is, choose a small number of ranges of each value, and give each value that falls in a particular range the same label. Then treat the labels as discrete values.

   Modify your Naïve Bayes classifier to use some of the continuous-valued attributes.

   Does using the continuous-valued attributes improve performance?

   When doing this, it might be helpful to plot a continuous-valued attribute against the diagnosis to see if particular ranges seem to be associated with particular diagnoses. Running a clustering algorithm such as K-means might be helpful also.

8 Version list

- Version 1.1, February 4th 2020.
- Version 2.0, January 31st 2021.
- Version 2.1, January 4th 2023.
- Version 2.2, January 23rd 2024.