ME4 Machine Learning - Tutorial 2

Lecture 2 discussed Bayesian methods. In this tutorial we will put that theory into practice. First of all we will calculate a distribution based on a combination of Normal distributions, and taking that as the likelihood we will calculate the posterior probability. We will also apply the Bayesian methods through the Scikit-learn library.

Use np.random.seed(5) for this tutorial (but generally default to 0).

1 Prior, posterior and likelihood calculation

1.1 Define a function to generate the Gaussian shape

In this tutorial we will make use of the Gaussian function a lot. This is defined as:

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\left(\frac{x-\mu}{\sigma} \right)^2 /2 \right]$$

In Python, define a function which takes inputs of x, mu (mean, μ) and sig (standard deviation, σ, sigma) and returns the value of the Gaussian function. So you can check your function, the Gaussian function with std = 0.5 and mean zero, calculated at x = 1, is 0.108.

1.2 Define two probability distributions

I have two states of nature, ω_1 and ω_2. What do $p(x|\omega_1)$ and $p(x|\omega_2)$ represent, where x is some observable quantity?

We wish to define two probability distributions for $p(x|\omega_1)$ and $p(x|\omega_2)$. $p(x|\omega_1)$ is a sum of two normal distributions/Gaussians, one with $\mu = 2$ and $\sigma = 1.5$ and one with $\mu = 7$ and $\sigma = 0.5$. Both are unscaled relative to each other, i.e. just summed together. $p(x|\omega_2)$ is defined in the same way, except with Gaussians at $\mu = 8$ and $\sigma = 2.5$, and $\mu = 3.5$ and $\sigma = 1$. Generate these two probability distributions, across a range of x values for 200 points between -10 and 20. Scale both distributions such that they integrate to 1 overall (hint: the command p /= np.trapz(p, x), where p is defined as a function of x, will divide p by the area beneath the curve). Plot these using

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
plt.plot(...)
```

where you will need to populate the plot() command.

1.3 Calculate the posterior distribution

The prior probabilities are $P(\omega_1) = P(\omega_2) = 0.5$. Calculate the posterior distribution and plot it for both states of nature. Compare to the likelihood for both states of nature. In both cases set the x axis limits to -3 to 15. If you were to change the probabilities to $P(\omega_1) = 0.9$, $P(\omega_2) = 0.1$, what do you think would happen to the posterior and why? Try it and see.

2 Classification with Bayes

We will now use scikit learn's naive_bayes tool to do some classification of data.

Make a dataset of 1000 samples with 2 informative features and no others through sklearn.datasets.make_classification().

We wish to use this dataset to both train our model and also test whether it is doing the right thing. To do this we need to split it into a testing dataset and a training dataset. We could do this manually ourselves, however, scikit learn has the function sklearn.model_selection.train_test_split(). Use this to split the data up, using 33% of the data in the test set and the residual in the training set. You will need to look up the documentation for the function to be able to do this, as well as a suitable 'import' line, but it should not be overly complicated.

Assuming that the two training data sets (parametric and classification data respectively) are in vectors X_train and y_train, and we have imported GaussianNB from sklearn.naive_bayes, the model can be fitted using

```
clf = GaussianNB()
clf.fit(X_train, y_train)
```

This 'fit' function is common across many of the tools we will use – the developers have tried to provide a consistent interface, even though the underlying principles are very different. In this case we are using the Gaussian (Normal) distribution, and the fitting approach will be finding suitable parameters to describe the distribution of points in X_train, depending on the classes given by y_train. This will be the likelihood. The method will then calculate a posterior distribution from this and perform predictions based on that. You can look online to find more information.

We now wish to produce an image showing the classification areas for the two parameters. The function clf.predict() can be used to predict the classification for any particular value (or sets of values) in a given X. We therefore need to make a suitable X which scans these two parameters across a region, which can subsequently be plotted. We have already done something very similar in tutorial 1 part 2, so try to reuse code from that. Place 200 points in each dimension, corresponding to each of the two features. Dimensions should be set to -3 to 3 in each direction. You can have a go at feeding this dataset into clf.predict yourself (note you may need to reshape and use a transpose), or to help, the code below should work assuming Xgrid defined as in tutorial 1:

```
classVals = clf.predict(Xgrid)
classVals = np.reshape(classVals, [200, 200])
```

Note that predict() is another common scikit-learn function. It takes a set of X values and produces estimated outputs, Y, which in our case are the classification values classVals. classVals can then be plotted using contourf as before. Plot the training data set on top of this (just use the scatter commands from tutorial 1 without adding a new figure first). Do you think the plot is a good match to the underlying data? You can try some other values for the random seed at the top to see how this performs with other datasets.

Check the performance of the algorithm for the test data set - use clf.predict() operating on X_test, and the output (e.g. y_test_model) will be the modelled values which we will compare to the true values in y_test. Use a for loop to calculate the numbers of correctly classified points and output this number as a percentage of the total:

```
nTot = len(y_test)
nMatch = 0
for i in range(len(y_test)):
        if y_test[i] == y_test_model[i]:
                nMatch += 1

print(100 * nMatch / nTot)
```

You can also plot an image of probability, which is output from clf.predict_proba(). Note that this will output two columns corresponding to the posterior probabilities associated with each classifier; you should select one of these then reshape the array to give something which can be plotted as an image - taking probVals as the output of clf.predict_proba():

```
probGrid = np.reshape(probVals[:, 0], [200, 200])
```