Naïve Bayes Tutorial
COMS3007

Benjamin Rosman, Devon Jarvis

February 24, 2020

Use your notes and any resources you find online to answer the following questions. Some functions which you may find useful are described at the end of the document.

1. We will look at using Naïve Bayes (NB) to do sentiment analysis – determining if some text has positive or negative sentiment. The file **simple-food-reviews.txt** contains 18 simple reviews for a restaurant. Each line of the file starts with either a 1 or a -1, representing a positive or negative label.

   (a) Train a NB model on a random 12 reviews, **with** Laplace smoothing. Test on the remaining 6. Report the confusion matrix of your results.

   (b) Train the model on the full review set. Generate your own positive and negative reviews (1 of each) that can confuse the NB. Why did they confuse it?

   (c) Repeat step 1(a) **without** Laplace smoothing. Report your results. What do you notice?

   (d) A common trick in text processing is to remove "stop words". These are short, commonly-occurring words, such as "the", "and", "is", "a". Try removing some simple stop words from the data. For this case remove all words of length 1 or 2. How does this affect classifier performance?

2. We will now look at a more challenging text-based classification problem, namely to classify a page from a Harry Potter book into which of the seven books the page was taken from. The books can be found in the folder "hp_books" and are text files where each page of a given book is a line in the text file. Note, all punctuation and capital letters have been removed from the file, so that only the words of the page remain to be used by our model.

   (a) Train an NB model using 80% of the data to train and the remaining 20% as test data. Use Laplace smoothing for your model. Report a confusion matrix of your results.

   Laplace smoothing is a simple way of avoiding 0 values in the class-conditional models (table of likelihoods). However, it may cause problems when many unique, infrequent words are added to the table (when multiplied together low likelihoods may still become 0 but too large a smoothing value will bias the model). In such a case even removing stopping words may not be enough. Thus, we will now smooth the table of likelihoods by adding a set value to each element of the table. The smoothing value used will now become a hyper-parameter for our algorithm, and so we will need to use a validation data set to find the correct value for the hyper-parameter.

   (b) Adapt your code to use 80% of the data to train, 10% of the data as validation data and the remaining 10% as test data. Train separate NB classifiers using the values $\{1 \times 10^{-1}, 1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times$

$10^{-4}, 1 \times 10^{-5}, 1 \times 10^{-6}\}$ to smooth the table of likelihoods. Train each model using the training data, and track its performance on the validation data. Which model gave the best accuracy on validation data? Does the choice of smoothing value have a big impact on the performance of the model?

(c) Use the model which achieved the best validation accuracy and test it using the test data set. Report a confusion matrix of the results, as well as the test accuracy of the model.

(d) Looking at the confusion matrix, which books would you say are most similar to each other (hint: look at which books are often confused with each other)? Do you think JK Rowling's writing style changed over time? Why else do you think certain books are more easily confused with each other?

3. The file **smalldigits.csv** contains 1797 8x8 images of hand written digits (modified from the sklearn digits dataset). Note the data here is low resolution and has been binarised for this example. Each row of the file has 65 elements: the first 64 are 0/1 values indicating the pixel value, and the last element is the class number (0-9). To visualise the nth digit, reshape the nth row into an 8x8 matrix. Train a NB classifier on a random 80% of the data, and use the remaining 20% to generate a confusion matrix showing its performance. Are these results reasonable? How might you improve them?

Potentially useful functions

(Note I will use capital letters to represent arrays and low-case letters represent scalars/variables):

- $np.array([])$: Creates an empty array.
- $np.zeros(x)$: Creates an array of length x of zeros
- $np.arange(x, y, z)$: Creates an array of values between x and y (not including y) using step size z.
  Eg: $np.arange(1, 7, 2) = [1, 3, 5]$
- $np.random.choice(X, y, replace = False)$: Returns a random subset of size y of elements from array X. $replace = False$ stops elements of X from being sampled more than once
- $np.delete(X, Y)$: returns an array identical to X except the values at the indices given in array Y are removed. So deletes elements of X found at the indices in Y
- $np.vstack([X, Y])$: Stacks array X on top of array Y
  Eg: if $X = [1, 2, 3]$ and $Y = [4, 5, 6]$ then $np.vstack([X, Y]) = [[1, 2, 3], [4, 5, 6]]$
- $np.full(x, y)$: returns an array of length y filled with the value x
- $np.random.shuffle(X)$: randomly re-orders array X (nothing is returned, array X itself is shuffled)
- $np.unique(X, return\_counts = True)$: returns an array of the unique values found in array X as well as an array showing the corresponding counts for each unique value.
- $np.append(X, Y)$: Adds the values found in array Y to array X but returns a flattened array (use vstack to create an array of arrays, append just creates one long array of scalars)
- $np.sum(X)$: Returns the sum of the elements in array X
- $np.prod(X)$: Returns the multiplication of all elements in array X
- $np.where(X\ with\ condition)$: Returns values of X where condition is True.
  Eg: $np.where([1, 2, 20, 21] > 10) = [20, 21]$
- $np.where(X\ with\ condition, True\_output, False\_output)$:Returns an array with the same length as X with $True\_output$ inserted where the condition is True and $False\_output$ inserted where the condition is false.
  Eg: $np.where([1, 2, 20, 21] > 10, True, False) = [False, False, True, True]$
- $np.in1d(X, Y)$: Returns an boolean array with "True" wherever a value in Y appears in X
- $np.invert(X)$: Inverts array X (True values become false, etc)
- $np.argmax(X)$: Returns the index where the maximum value in X occurs.
- $x.split(char)$: Splits a string x into separate arrays wherever a character $char$ is found.
  Eg: $a =$ "Hello World"
  $a.split($" "$) =$ ["Hello", "World"]