Foundations of Intelligent Systems Lab 2

Christian Heinrich

Code Usage

In order to facilitate both training and testing relatively flexibly, I implemented a simple menu system, with three options to choose from, selected by typing in either “Train”, “Text” or “Exit” into the prompt. I chose to have the user actually type in a full word, rather than just a single character as it was slightly simpler to implement it that way and more importantly prevented the user from accidently retraining the System, which could potentially erase a good model in the process. The “Train” command will read in all the training data from the training folder, and use that to perform back propagation learning, and then save the found weights in a text file in a separate research folder. The training data is saved in the training folder in the form of 45 text files, 15 for each language, each labeled either D#, E#, or I#, where the # is the number of the example for the language indicated by the first letter, so D1 is the first example for Dutch, which E5 is the fifth example for English etc. The text segments for the training data are save as a single line, such that they can be read in simply and easily. These are read in when the Train command is given, and the network is retrained from scratch. I opted to iterate through the examples a set number of times (in this case 100 times) as this provided a good balance between speed of calculation vs. accuracy. For example, I initially tested with only 10 repeats instead of 100, and with that network at least one of my test cases for Dutch came back as Italian, which is an issue I’ve had with a couple of test cases across various models.

In addition to this training data which is read in automatically, I have my own Test Set, which is about a fifth the size of the training set, which is stored separate from the training set in its own folder, which won’t be read in automatically. Instead test data is entered in manually, as this allowed me to test more quickly than reading them in from text files, (this was also required by the assignment). In order to test a text segment, first give the Text command at the console menu prompt. There will then be a line asking you to input the desired text segment on the line bellow. Doing that and hitting enter will then return the networks evaluation of which language it is. The text segment entered should always be all on a single line, as otherwise the program will not properly register the input. (the training data is also stored as single long line of text.)

Feature Selection

I ended up employing several features in order to differentiate the various languages, and tested several more. Likely the two most key features, though one of them took up 5 inputs, were the letter repeat and jkwxy tests, as these appear to be the most identifiable elements of Dutch and Italian respectively. The letter repeat feature refers to one of the largest quirks with the Dutch language, in that it has an overwhelming tendency to have two of the same letter right next to each other, for example the double o in boom would be similar to this feature. The frequency of this feature appearing was high enough that it’d likely be core to any problem along these lines. The second feature, the jkwxy tests, was actually a single test until late in the development of the program, as I decided to split all the letters apart into their own input. These features are based on the fact that Italian only has a 21 letter alphabet, without any of these letters being native to the language, only appearing in imports from other languages. This made these features the perfect way to rule out Italian as a possible language for a given text. The final feature I selected was the IJ check, to see if these two letters were next to each other, as this is a very common element of Dutch. I didn’t use any real features to find if the text was written in English, however I opted to eliminate the other two options in order to determine if the text was English, as this was simpler than trying to select a Feature from English, due to the amount English has bled into other languages. I did attempt to use a feature meant to determine if the text was English, which was the Th feature, which checked if the letters T and h were next to each other as this is a common digraph in English. However, this ended up not working out, as the test actually seemed to be guiding the guess towards Italian, rather than English, so I opted to remove it. I also attempted to check if a word was beyond a certain length, in this case 10, under the idea that since Dutch employs compound nouns, Dutch would have a higher density of long words, however, that didn’t end working out, as the number of compound nouns was lower than anticipated and it was weighting towards both English and Italian, thus throwing the whole prediction off, hence I opted to disable it.

Overall, the jkwxy features combined with the letter repeat feature is an extremely effective way of determining which language the text is, as they are likely the most standout features of those languages.

Data Gathering

In order to gather data, I made extensive use of the random article feature on all three language’s Wikipedia. Every example from both my training and test sets are from random Wikipedia articles, as this provided ample sources for both training and test data. (Though oddly the Dutch Wikipedia overwhelmingly favored sending me to pages about various flora and fauna, which I avoided drawing on too much, as I wanted to try and get a wide variety of examples.) I opted to create a training set of about 15 examples for each language, as this seemed sufficient in this case to create an accurate model. I also gathered a set of nine test cases, to be sure that the model is accurate, though these test cases still have to be manually entered. I did avoid using anything which was overly related to another nation/language, especially any of the other languages being tested for. I also tried to avoid getting too much data about a single topic, as this also has the potential to overly bias the information. I also attempted to get a range of text input word lengths. Overall I believe the training data represents a fair spread of information.

Training Process

The training process first reads in the training data, then processes the text and stores the counts for the various inputs for each text inside of a list of Example objects, which store the numeric functions of the text, rather than having to store the entire text. The network has already been initialized prior to the training operation beginning, using the init() function of Network. The weights are stored in a WeightSet class, which has two 2d arrays of size NxN, where N is the number of nodes +1, one of which stores the connections between nodes ij, with a value of 1 indicating an edge between the nodes I and j, and the other holding the value of the weights. This is to prevent the possibility of having edges form between the input and output layers, skipping over the hidden layer. A this point the back propagation learning function is run, which will run for each example, and then repeat a hundred times, maintaining the same weights, in order to make sure the weights are as accurate as possible. Finally these weights are written out to the text file in order to allow the user to run tests using those weights.

Best Network

I believe the final network I have is the Best Network I’ve gotten, as it has had a fairly high success rate, with the only issue being some hiccups with a couple tests differentiating between Dutch and Italian. I feel a large part of this comes down to the removal of the word length test, as this proved to be ultimately more of a problem than a benefit, as well as fracturing what was once a single input (jkwxy value) into several inputs, as this allowed for a bit more nuance from the model, as well as increasing the success rate regarding judging Dutch text, as j and k are slightly more common in Dutch than either English and certainly Italian.