Lab 2 Writeup

Monday, December 2, 2019 7:01 PM

Project Structure

The python script provided in the discussion post (scrape-data.py) was used to collect 15 word sentences and store them labelled in the file 'data.txt'. The Decision Tree model was stored in '***dtr.pickle'*** and the Adaboost model was stored in '***ada.pickle'***

To train models

- `python lab2.py train data.txt [dtr.pickle/ada.pickle] [dt/ada]`

To predict using the models

- `python lab2.py predict [dtr.pickle/ada.pickle] train.dat`

File used in background to build trees:

- Object class: Tree() in dtree.py
- Tree developing class: build_tree.py
- Analysis class: analysis.py
- CSV containing frequency of features sorted by ascending order
  - bigrams_nl.csv
  - bigrams_en.csv
  - letters_en.csv
  - letters_nl.csv

Training Process:

Decision Tree

The program starts in *lab2.py* by reading the data and creating a dataframe with all features calculated through use of functions defined in the file itself. Then, the code uses the *Tree*() class defined in *dtree.py* and uses the *build()* function to build the tree with the help of the features, labels and dataframe defined. This *build*() function again references the file *build_tree.py* which has means of creating the tree itself in the form of a dictionary, through using helper functions to facilitate entropy and information gain calculation.

Adaboost

The program starts in *lab2.py* in a similar fashion, created a dataframe and then referenced the *boosting.py* file to create a Boosting class, which in its rawest form contained alpha values and a list of decision stumps. This class referenced the *Tree* class in *dtree.py* to create decision stumps by further referencing the *build_decision_stump()* function in build_tree.py - a decision tree created by using the entropy rules and limited to a depth of one. For each model, the alpha values were recorded by storing the misclassification and predictions in an evaluation dataframe (*eval_df*) and has a built in *predict*() function to facilitate using the Adaboost model for predictions on test data.

Feature Selection

Before any feature selection, some research was conducted to identify differences between the features of the words that constituted either language. This blog post was useful in gathering a basic idea of the primary differences between how the words themselves were different in each language.

The features were represented as Boolean values in the dataframe that represented our knowledge base since it got a little complicated to work with numerical data and build a decision tree that worked with all kinds of data.

1) Dutch Bigrams:

`'nl_bigrams'`

An effective way to identify distinguishable features of a language would be to figure out popular letter combinations that exist throughout the language database. Through use of helpful software like the Frequency Analysis Tool. Text was collected through a python script and analyzed to reveal the most popular two letter combinations that occurred in the text. For the Dutch language, they were the substrings 'en', 'er' and 'de'. If the sentence contained at least two of these three character couples, the feature would be True, and otherwise False.

2) English Bigrams:

`'en_bigrams'`

Similarly, the same software was used to identify the most popular two letter combinations in English, which turned out to be 'th', 'he' and 'in'. Again, if two of the three substrings were identified, the feature would be True, and otherwise False.

3) Hyphenated Words:

`'hyphenated'`

The dutch contains a lot of compound words, most of which are hyphenated. Detection of a hyphen in the string would result in a True value, and otherwise False

4) Double Letters:

`'double_letters'`

The dutch language has a greater frequency of pairs of letters occuring in their sentences. Detection of such a case results in True, and otherwise false

5) Double Consonants:

`'double_consonants'`

Similarly, detection of double consonants signifies a greater affinity towards the dutch language.

6) Number of Vowels:

`'no_of_vowels'`

Upon analysis conducted on the training data, the number of vowels in english on average was 27, and in dutch was 29. Vowel frequency with less than 28 resulted in a True value, and otherwise false.

7) Long Words:

`'has_long_words'`

Long words are defined as words with length greater than 7 - very popular in the dutch language. More than long words resulted in a True value, and otherwise False.

8) Short Words:

```
'has_short_words'
```

Similarly, English has a lot shorted words, also defined as words with length less than or equal to 6. More than three short words resulted in a True value, and otherwise False.

9) Contains 'F':

```
'has_f'
```

The letter 'f' is used a lot more in English when compared to the Dutch langauge.

10) Contains 'Y':

```
'has_y'
```

The letter 'y' is also used a lot more in English when compared to Dutch

Decision Tree Implementation

First, a dataframe was used to read all the training data in the form of sentences and then store a set of attributes to define the sentence's features. To avoid complication, all features were decided to be Boolean values, which were used to construct a tree using the entropy to calculate the homogeneity of the sample data. By simply using the three end-conditions as mentioned in the textbook, the resulting decision tree provided an accuracy of around 80% on unseen testing data. The paramaters tweaked to achieve the best tree were :

- Length of long words: chose 7 from set of [6, 7, 8, 9, 10]
- Dutch bigrams threshold: Chose 2 from set of [1, 2, 3]
- English Bigram Threshold: Chose 2 from set of [1, 2, 3]
- Length of short words: Chose 5 from set of [5, 6, 7, 8]
- Number of vowels: Chose 28 from [27, 28, 29]

Challenges faced:

- Sample weight: Sample weight was not taken into consideration when first implementing the decision tree which resulted in errors that could have been tackled in lesser time
- Choosing features: Really had to think hard to come up with features dividing the two languages
- Overall accuracy: Could not achieve 100 % accuracy because of time limitation

Adaboost Implementation

The Adaboost implementation took a little more time considering the insertion of the sample weights into the entropy and information gain equation. Building a decision stump was fairly easy once it was clear how to select leaf values based on the weights for different target values. Time limitations did not allow tuning of hyperparameters but it was noticed **that the weights stopped changing all that much after the creation of the twelth or thirteenth decision stump**. To create these 1 depth decision trees, the class being used to create Decision trees was used but with a couple different functions in use.

Challenges faced:

- Incorporation of sample weights into entropy and information gain calculation
- Error rate calculation was fully working only after a few trial and error runs