# Lab 3 Writeup:

Matthew Morrison

CSCI-331-02

msm8275

1. How to use the program:

- **python lab3.py train <examples> <features> <hypothesisOut> <learning-type>**
  - Used to read in labeled examples and train either a decision tree or adaboost hypotheses
    - examples: a file containing labeled examples that consist of the following -
      - label designating en for English or nl for Dutch
      - a "|" separating the label and words
      - a sequence of 15 words from a Wikipedia article
    - features: a file containing features to be trained against. The features should be a word or substring, separated line by line
    - hypothesisOut: the file name to write the new model to
    - learning-type: specifies the type of learning algorithm to run, either
      - "dt" for decision tree, or
      - "ada" for adaboost
- **python lab3.py predict <examples> <features> <hypothesis>**
  - Used to take in non-labeled examples and predict/classify each example as either English or Dutch
  - This function will print the predicted label on a newline
  - examples: a file containing non-labeled examples that consist of the following -
    - a sequence of 15 words from a Wikipedia article
  - features: a file containing features to be trained against. The features should be a word or substring, separated line by line
  - hypothesis: the file name to read the tree model from

2. Selection of Features

- My selection of the features consists of the most common words in the English and Dutch vocabulary that **do not intersect each other**. When investigating the most common Dutch words, I had to ignore words like "in", "is", "was", and more, as they would be confusing to train on since they show up for both languages. I also tried to eliminate some words that were substrings of words (the way it tests if the feature is in the string is by testing if it is a substring of it. Future improvements to test word for word will be added.), so words like "en" could not be used, as it was common to see that inside words of English strings. I also tried to include distinct words that were not really seen in the complement language. Trying to

follow language conventions, words like “zij” or “where”, sequences that were uncommon in the complement language, was preferred. With this in mind, I decided on 22 features, 8 English and 14 Dutch. I wanted to focus a little more on distinct Dutch features, as both have German dialects, but I noticed Dutch having more distinct features to train on.

3. Testing Results with Decision Tree Learning

- The decision tree algorithm is what we utilized in class. Using the set of features provided, it will:
  - Check the importance of each not-used feature by calculation of its entropy.
  - Whichever feature has the highest importance, add it as a node to the overall tree.
  - Each example can either have the feature (True) or not have it (False). These will be treated as the edges for the node.
  - For each example, split them up into either having the feature or not.
  - Recursively run the dt algorithm with the reduced examples and without the previously best feature
  - Repeat until it has either reached the max depth, or a base case has occurred, which can be:
    - every example has the same classification
    - no more features
    - no more examples
  - if it reaches a base case, look for the majority answer (their classifications) of the examples and create a leaf node.
- After tinkering with the max depth (labeled max_dt_depth), I came to the conclusion that the best depth was 22, the number of features. When messing with the parameter, it plateaued around 13, and going to the max number of features (up to 22) resulted in slight variation, with 22 actually having a slight edge. It would exponentially improve, and then plateau itself at the best possible depth that I found. This could be changed based on the number of training data and number of testing data, but I felt it was sufficient to display the efficiency of the decision tree learning algorithm. Originally, I had it at 13, but that was when I had set an edge case to when all of the importances were 0 to create a leaf node. After looking at the pseudocode again, this is not what should have happened, so it will continue exploring instead of saying it is an edge case.
- At its best possible depth, here are the testing results:
  - With 703 examples to test against with test.dat:
    - 671 Correct
    - 32 Incorrect
    - 95.45% correct overall
  - With 1101 examples to test against with test2.dat
    - 1039 Correct
    - 65 Incorrect
    - 94.37% correct overall

4. Testing Results with Adaboosting

- With the Adaboost algorithm, it did not see as much success in comparison with the dt algorithm.
- The Adaboost algorithm works as follows:
  - Until depth k, run the dt algorithm to a depth of 1 with **weighted** examples.
  - The examples originally are weighted at 1/len(examples) and get updated with each run of adaboost.
  - Using this single stump tree, check the number of correctly and incorrectly identified examples.
  - Using this, correct and normalize the weights of the examples
  - Calculate the weight of this stump (i.e. how much of a say it has)
  - Repeat until we reach the max designated depth.
- For this algorithm, I was only able to observe a max depth (labeled max_adaboost_depth) of 3. I noticed while testing that the first tree would have the feature of "the" used. This makes sense because it was seen in a vast majority of English examples. However, because adaboost does not remove already used features, by depth 4, "the" would again be the most important feature in the stump, which would inflate its importance and caused a distinct drop in accuracy (from 87.06% to 78.24%). Even with a massive increase in its depth, it would plateau at the 78.24% accuracy, so I opted to use a shorter depth to increase its overall accuracy. To solve this, I could add more distinct features in the list to test against. Also, as mentioned earlier, adjusting the implementation of the examples from just a single string to a list of words could help me add even more features to test with, but that can be implemented in the future.
- At its best possible depth, here are the testing results:
  - With 703 examples to test against:
    - 612 Correct
    - 91 Incorrect
    - 87.06% correct overall
  - With 1101 examples to test against with test2.dat
    - 959 Correct
    - 145 Incorrect
    - 87.10% correct overall

5. Other Comments

- I have added a few additional files that can help display the statistics with my test data:
  - test.dat: the first tested data from the analysis seen in the writeup
  - test2.dat: the second tested data from the analysis seen in the writeup
  - answers.dat: includes all the answers to all the test examples in test.dat.
  - answers2.dat: includes all the answers to add the test examples in

test2.dat.
    - **Uncommenting the lines in predict_dt and predict_ada functions will allow you to use the compare functions, which will print out the number of correctly and incorrectly identified examples**
- best_dt.model: the best decision tree model (this will be the same as best.model, but just to differentiate the two for comparison)
- best_ada.model: the best adaboost model