Feature Selection

We think the most useful and meaningful features about recognizing different languages are the frequency of the characters [1][2]. Because letters contribute a significant part in pronunciation and different cultures will create different words. Hence, letter frequency can be reliable to recognize the language.

Italian Letter Frequency

Letter	Frequency
E	11.49%
A	10.85%
I	10.18%
O	9.97%
N	7.02%
T	6.97%
R	6.19%
L	5.70%
S	5.48%
C	4.30%
D	3.39%
U	3.16%
P	2.96%
M	2.87%
V	1.75%
G	1.65%
H	1.43%
B	1.05%
F	1.01%
Z	0.85%
Q	0.45%
È	0.42%
À	0.15%
Ù	0.12%
Ò	0.11%
Ì	0.09%
É	0.06%

Dutch Letter Frequency

Letter	Frequency
E	19.31%
N	10.04%
A	7.76%
T	6.50%
O	5.88%
R	5.70%
D	5.48%
I	5.03%
S	3.91%
L	3.85%
G	3.17%
H	3.16%
K	2.83%
M	2.60%
V	2.27%
U	2.13%
W	1.74%
Z	1.62%
P	1.51%
B	1.38%
IJ	1.34%
C	1.31%
F	0.74%
J	0.51%
Y	0.06%
X	0.05%
Ä	0.03%
Ë	0.03%
Ü	0.02%
Ï	0.01%
Ö	0.01%
Q	0.01%
É	< 0.01 %
È	< 0.01 %

Difference

Letter	Frequency
E	7.82%
I	5.15%
O	4.09%
A	3.09%
N	3.02%
C	2.99%
K	2.83%
D	2.09%
L	1.85%
W	1.74%
H	1.73%
S	1.57%
G	1.52%
P	1.45%
U	1.03%
Z	0.77%
V	0.52%
J	0.51%
R	0.49%
T	0.47%
Q	0.44%
B	0.33%
F	0.27%
M	0.27%
Y	0.06%
X	0.05%

Additionally, to filter noise as much as possible and improve the performance of the algorithm, we only choose the top 6 letters (E, I, O, A, N, C) which have the highest frequency difference between Dutch and Italian to be the features. Also, we ignored the letters only appear in one language.

We also considered using average word length to recognize languages, but the distribution pattern of Dutch and Italian is almost the same. Hence, we didn't use it.

Data Gathering

We choose 3 Dutch text segments and 3 Italian text segments to be the examples. Each text segments are 20 words long and come from the wiki. And we try our best to avoid numbers, names, buildings, and markers to keep the examples including as many words as possible.

After getting the text examples, we use a simple algorithm to convert them to a truth table. Firstly, we count all the letters and calculate their frequency. Then, if the frequency is close to Dutch's, we give the true value. Otherwise, it will be a false value. Finally, we get the truth table.

Example	E	I	O	A	N	C	Language
X_1	T	F	T	T	F	T	T
X_2	T	T	T	T	T	T	T
X_3	T	T	T	F	T	T	T
X_4	F	F	T	F	F	F	F
X_5	F	F	F	T	T	F	F
X_6	F	T	F	F	F	F	F

Training Process

Decision Tree

The decision tree training process is the same as the decision tree algorithm in the lecture. We do a DFS order to expand the tree. Firstly, we will calculate the gain of all the attributes which aren't used by the tree. Then, choose the attributes which can provide the highest gain. Finally, append two child nodes and repeat the process again and again until the algorithm hitting the limit or there is no such child.

The training process will train the decision tree from depth limit 1 to 6. And choose the tree which has the highest accuracy. When two trees have the same accuracy, the algorithm will choose the tree with a higher limit.

Because the gain calculation process may be too long to print and DFS order may let the print result unclear. We only print the final accuracy of each depth limit.

```
Code output:
Build Decision Tree:
----Find best depth limit----
Acc:
1: 1.0
2: 0.6666666666666666
3: 0.8333333333333334
4: 0.6666666666666666
5: 0.6666666666666666
6: 1.0
Best Depth: 6
----Finish Decision Tree Training----
```

Adaboost

The algorithm is almost the same as the pseudo-code in the lecture. Because the original version will assign 0 to the example weight when there is no error for the example. Hence, we changed the function.

If example h[k] go wrong, w[j] = w[j] / 2 * (1 - error). To keep the weight be positive when the example doesn't have error.

Except that we didn't do any modification. Firstly, the algorithm will calculate the deafault weights for all the attributes. Then, it will go through several iterations to recalculate the weight depending on the result.

Our Adaboost training process will start from stump number 1 to 6. Because the algorithm has 6 attributes. When the stump number is smaller than 6, it will choose attributes having the highest gain in descending order. Then, the algorithm will choose the model having the highest accuracy. When multiple models have the same accuracy, it will choose the one that has more stumps.

```
Code output:
Adaboost:
----Find proper stump number----

a0 weight: 0.0
a1 weight: 0.0
a2 weight: 0.0
a3 weight: 0.0
a4 weight: 0.0
a5 weight: 1.0
trueSum: 1.0falseSum: 0.0
trueSum: 1.0falseSum: 0.0
trueSum: 1.0falseSum: 0.0
```

```
trueSum: 0.0falseSum: 1.0
trueSum: 0.0falseSum: 1.0
trueSum: 0.0falseSum: 1.0
stump number: 1 Acc: 1.0

a0 weight: 1.0
a1 weight: 0.0
a2 weight: 0.0
a3 weight: 0.0
a4 weight: 0.0
a5 weight: 1.0
trueSum: 2.0falseSum: 0.0
trueSum: 2.0falseSum: 0.0
trueSum: 2.0falseSum: 0.0
trueSum: 0.0falseSum: 2.0
trueSum: 0.0falseSum: 2.0
trueSum: 0.0falseSum: 2.0
stump number: 2 Acc: 1.0

a0 weight: 1.0
a1 weight: 0.0
a2 weight: 1.5677471079645748
a3 weight: 0.0
a4 weight: 0.0
a5 weight: 1.0
trueSum: 3.567747107964575      falseSum: 0.0
trueSum: 3.567747107964575      falseSum: 0.0
trueSum: 3.567747107964575      falseSum: 0.0
trueSum: 1.5677471079645748     falseSum: 2.0
trueSum: 0.0falseSum: 3.567747107964575
trueSum: 0.0falseSum: 3.567747107964575
stump number: 3 Acc: 1.0

a0 weight: 1.0
a1 weight: 0.0
a2 weight: 1.5677471079645748
a3 weight: 0.0
a4 weight: 1.5899318927965926
a5 weight: 1.0
trueSum: 3.567747107964575      falseSum: 1.5899318927965926
trueSum: 5.157679000761167      falseSum: 0.0
trueSum: 5.157679000761167      falseSum: 0.0
trueSum: 1.5677471079645748     falseSum: 3.5899318927965926
trueSum: 1.5899318927965926     falseSum: 3.567747107964575
```

```
trueSum: 0.0falseSum: 5.157679000761167
stump number: 4 Acc: 1.0

a0 weight: 1.0
a1 weight: 0.0
a2 weight: 1.5677471079645748
a3 weight: 1.7458501980808627
a4 weight: 1.5899318927965926
a5 weight: 1.0
trueSum: 5.313597306045438      falseSum: 1.5899318927965926
trueSum: 6.90352919884203       falseSum: 0.0
trueSum: 5.157679000761167      falseSum: 1.7458501980808627
trueSum: 1.5677471079645748     falseSum: 5.335782090877455
trueSum: 3.3357820908774554     falseSum: 3.567747107964575
trueSum: 0.0falseSum: 6.90352919884203
stump number: 5 Acc: 1.0

a0 weight: 1.0
a1 weight: 2.1151975416349145
a2 weight: 1.5677471079645748
a3 weight: 1.7458501980808627
a4 weight: 1.5899318927965926
a5 weight: 1.0
trueSum: 5.313597306045438      falseSum: 3.705129434431507
trueSum: 9.018726740476945      falseSum: 0.0
trueSum: 7.272876542396082      falseSum: 1.7458501980808627
trueSum: 1.5677471079645748     falseSum: 7.45097963251237
trueSum: 3.3357820908774554     falseSum: 5.68294464959949
trueSum: 2.1151975416349145     falseSum: 6.90352919884203
stump number: 6 Acc: 1.0
---- Finish ----
Highest Acc: 1.0 Best Stump Number: 6
Final Weights:
a0 weight: 1.0
a1 weight: 2.1151975416349145
a2 weight: 1.5677471079645748
a3 weight: 1.7458501980808627
a4 weight: 1.5899318927965926
a5 weight: 1.0
```

Result

Decision Tree:

The best model: depth limit 6

Accuracy: 100%

Adaboost:

The best model: stump number 6

Accuracy: 100%

Although, both models provide the same accuracy result. We believe the AdaBoost is more reliable than the decision tree version. Because decision tree more like a gamble and we didn’t have enough amount of high-quality data to train it. On the contrary, AdaBoost's idea is much more reasonable. It gives all the attributes different weights depending on the test result. Hence, we believe the AdaBoost model is much more reliable.

Reference

[1]: Alphabet and Character Frequency: Dutch (Nederlands), Stefan Trost, accessed 29 April 2021, https://www.sttmedia.com/characterfrequency-dutch

[2]: Alphabet and Character Frequency: Italian (Italiano), Stefan Trost, accessed 29 April 2021, https://www.sttmedia.com/characterfrequency-italian