COMP9318 Project: Predict Stress in English Words

Deadline: 23:59 20 May 2017 (Sat)

In this project, you need to build a classifier to predict the stresses for a list of English words.

1. Represent Pronunciations of English Words

1.1 Phonemes

We use the following symbols to represent phonemes of English words.

- Vowel phonemes: AA, AE, AH, AO, AW, AY, EH, ER, EY, IH, IY, OW, OY, UH, UW
- Consonant phonemes: P, B, CH, D, DH, F, G, HH, JH, K, L, M, N, NG, R, S, SH, T, TH, V, W, Y, Z, ZH

Vowel	IPA		consonant	IPA	consonant	IPA
AA	ɑ		P	p	S	s
AE	æ		B	b	SH	ʃ
AH	ə / ʌ		CH	tʃ	T	t
AO	ɔ		D	d	TH	θ
AW	aʊ		DH	ð	V	v
AY	aɪ		F	f	W	w
EH	ɛ		G	g	Y	j
ER	ɜːr		HH	h	Z	z
EY	eɪ		JH	dʒ	ZH	ʒ
IH	ɪ		K	k		
IY	i		L	l		
OW	oʊ		M	m		
OY	ɔɪ		N	n		
UH	ʊ		NG	ŋ		
UW	u		R	r		

Note: in this project, we follow the pronunciation of **American English**.

1.2 Stress

We use 0/1/2 followed by a *vowel* phoneme to indicate the stress:

- 0: No stress
- 1: Primary stress
- 2: Secondary stress

1.3 Word Stress Rules

We make the following assumptions in this project (they are true in most cases)

- A word **only have one** pronunciation (we do not consider words with multiple pronunciations)
- A word **must have one and only one** primary stress
- Only vowels are stressed

In addition, we **only consider words with less than 5 vowels** (i.e., words with 5 or more vowels have been removed from training and testing datasets).

1.4 Example

We take the word **university** (pronounced as [ˌjunəˈvɜrsəti]) as an example, its pronunciation is formed like:

Y UW2 N AH0 V ER1 S AH0 T IY0

2. File Format

2.1 Training Data

The training data contains 50,000 words. Each word (uppercase) follows by its pronunciation, formed like

```
Word:Pronunciation
```

For example, the line corresponding to word *university* should be:

```
UNIVERSITY:Y UW2 N AH0 V ER1 S AH0 T IY0
```

2.2 Testing Data

The testing data contains several lines, each line corresponds to a word.

The only difference compares to the training data is that, in the testing data, we have removed all the stress symbols (i.e., 0/1/2). And you need to predict them.

For example , the line corresponding to word *university* in the testing data should be:

```
UNIVERSITY:Y UW N AH V ER S AH T IY
```

2.3 Helper Functions

In order to make your life easier and avoid unnecessary bugs, we will offer a helper function to read training/testing data from files, and convert them into a list of strings. And the list will be passed as an argument to the training and testing functions. Please refer to 4.1.2 for execution example.

3. Your Task

3.1 Output the position of the primary stress

For each word in the testing data, you need to output the position of the primary stress. Since only vowels are stressed, we only count vowels. For example, you should output **3** for the word **university**, as the primary stress of the word university is on the 3rd vowel (i.e., **ER**).

Assume the testing data contains 5 words, whose primary stresses are on the 1st, 2nd, 1st, 3rd and 2nd vowel. Then your `test()` function should return a list of 5 integer numbers: `[1, 2, 1, 3, 2]`. In order to do that, you need to train a classifier using `train()` function.

3.2 train()

In order to successfully predict the stress, you need to train a classifier. You are required to implement a function named train(). Its two arguments are the training data (stored as a list of strings) and the output file path.

You need to dump the classifier and relevant data/tools (if there is any) into one single file. Hint: a easy (but ont the only) way of doing this is to use `pickle`.

In [1]:

```
def train(data, classifier_file):
    pass;
```

3.3 test()

You also need to implement a function named test(), which takes the test data as input and returns a list of integers which indicate the positions of the primary stress.

In [2]:

```
def test(data, classifier_file):
    pass;
```

3.4 Restrictions

- The **total** running time of training and testing **should not exceed 10 minutes** in the submission system. The system will force stop your program if it took more than 10 minutes, and you will receive 0 point for the programming part.
- You are encouraged to import classifiers from sklearn, but you can **only** import and use **classifiers that has been introduced during the lecture**. For example, you cannot import `sklearn.ensemble.RandomForestClassifier` as it is not covered in the lecture. We do this by assuming that all the students have the same background knowledge.

3.5 Report

You need to submit a report (in PDF format) which answers at least the following two questions:

- What features do you use in your classifier? Why are they important and what information do you expect them to capture?
- How do you experiment and improve your classifier?

4. Evaluation

4.1 Execution of the submitted code

Your submission will be tested automatically. Inorder to avoid unneccessary exceptions/errors, please make sure

- you have followed the instructions strickly
- you have tested your code before submission

4.1.1 Execution environment

We have pre-installed the following modules, you can only use these modules and the built-in modules/functions.

- python: 3.5.2
- pandas: 0.19.2
- numpy: 1.12.1
- scikit-learn: 0.18.1

4.1.2 Execution example

Note: we will only test `train()` and `test()`; **none** of the other functions in your `submission.py` will be called by us. So make sure you called them, if any, within your `train()` or `test()` function properly.

You can imagine that our test code looks similar to the following.

In [3]:

```
import helper
import submission

training_data = helper.read_data('./asset/training_data.txt')
classifier_path = './asset/classifier.dat'
submission.train(training_data, classifier_path)
```

In [4]:

```
test_data = helper.read_data('./asset/tiny_test.txt')
prediction = submission.test(test_data, classifier_path)
print(prediction)
```

```
[2, 1, 2, 1]
```

4.2 Evaluation

Your code will contribute 80% to your final score of the project, and your report will contribute the rest 20%.

After the execution, the output result of your submitted code will be compared to the ground truth. And the **marco averaged F_1 score** will be used to determine the score for the programming part.

The precision->score function is a step function:

- score = 80, if F_1 score $\geq T_1$
- score = 50, if $T_1 > F_1$ score $\geq T_2$
- score = 20, if $T_2 > F_1$ score $\geq T_3$
- score = 0, if F_1 score $< T_3$

T_i will be announced later.

4.2.1 Example

For example, in the above `tiny_test` case, the correct output should be:

```
[1, 1, 2, 1]
```

Therefore, the **marco averaged F_1 score** of the example result is 0.73.

Assume $T_1 = 0.85$ and $T_2 = 0.7$, then the score for the programming part will be 50.

In [ ]:

```
from sklearn.metrics import f1_score
ground_truth = [1, 1, 2, 1]
print(f1_score(ground_truth, prediction, average='macro'))
```

4.3 Bonus

The 20 best performed classifier will be rewarded by at most 20 bouns points. More specifically, the 1st place will get 20 points, the 2nd place will get 19 points, and so forth.

5. Submission

Similar to the labs, **you need to submit both `submission.py` and report to the online submission system.** You cannot submit other files.

The online system will try to execute your submitted code using data *sampled* from the test dataset that will be used for the final evaluation.

For example, *assume* the test dataset $\mathcal{D}$ for the final evaluation has 1000 words. Then each time when you submit your code, the system will randomly sample n words from $\mathcal{D}$, and use these n words to test your code.

You will be able to see the precision of your code on the n words. If your code cannot be correctly executed, then you will receive an error message.

5.1 Submission restrictions

Due to obvious reasons, we have the following restrictions. Please strictly follow them.

- Each student has only **10** chances to submit their *code* (no matter it can be correct executed or not). But you can submit report as many times as you want.
- The `submission.py` file should **not exceed 30KB**.
- The `report.pdf` file should **not exceed 10MB**.

5.2 Late penalty

- -20% for each day after the due date.

NOTE

- We will take the time of your last submission (no matter its the code or report submission) as your submission time.
- We will only store your last submitted code and last submitted report; There is no way we can use your earlier versions and we do not accept any file from you after the deadline.

In [ ]: