UDACITY

PROJECT

Build a Sign Language Recognizer

A part of the Artificial Intelligence Nanodegree and Specializations Program

PROJECT REVIEW

CODE REVIEW 4

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Hi Udacity Learner,

Impressive work! The project has all the requirements. The features of `my_model_selectors.py` and `my_recognizer.py` files are well-implemented and it works great. In addition, the answers to the questions in the submission are good and well organised. The submission perfectly addresses all the rubric points. Nice work on being extra and doing the optional challenge and improving the WER. Keep up the good work dear student!

PART 1: Data

1. **Student provides correct alternate feature sets: delta, polar, normalized, and custom.**
2. **Student passes unit tests.**
3. **Student provides a reasonable explanation for what custom set was chosen and why (Q1).**

1. Correct alternate feature sets for `delta`, `polar`, `normalized`, and `custom` have been provided. All implementations work accordingly.
2. All four features passed the unit test, excellent!
3. Very concise explanation for using both delta and normalization. The explanation is brief but comprehensive. Good job!

Awesome work on passing the requirements for the first part!

PART 2: Model Selection

1. **Student correctly implements CV, BIC, and DIC model selection techniques in "my_model_selectors.py".**
2. **Student code runs error-free in notebook, passes unit tests and code review of the algorithms.**
3. **Student provides a brief but thoughtful comparison of the selectors (Q2).**

1. The selectors have been implemented correctly i.e `BIC`, `DIC`, and `CV`.
2. Aside from having an error-free code in the notebook, the algorithms used in `my_model_selectors.py` also passed the unit tests.

```
ok
test_select_constant_interface (asl_test_model_selectors.TestSelectors) ... ok
test_select_cv_interface (asl_test_model_selectors.TestSelectors) ... ok
test_select_dic_interface (asl_test_model_selectors.TestSelectors) ... ok

----------------------------------------------------------------------
Ran 4 tests in 30.500s

OK
```

3. The comparison provided is concise. Also, nice work on citing the reference, the discussion was very clear.

Good job, all the requirements for the second part are complete. Well done!

PART 3: Recognizer

1. **Student implements a recognizer in "my_recognizer.py" which runs error-free in the notebook and passes all unit tests**

2. Student provides three examples of feature/selector combinations in the submission cells of the notebook.
3. Student code provides the correct words within <60% WER for at least one of the three examples student provided.
4. Student provides a summary of results and speculates on how to improve the WER.

1. The algorithm used in `my_recognizer.py` passed the unit tests as well the one in the notebook.

```
ok
test_recognize_probabilities_interface (asl_test_recognizer.TestRecognize)
0x000001D3D9F800F0>
ok

----------------------------------------------------------------------
Ran 2 tests in 33.684s

OK
```

2. Three examples were present i.e. `features_custom, features_polar, features_delta_2/SelectorBIC`, `features_polar/SelectorDIC`, `features_custom, features_polar, features_delta_2/SelectorCV`.
3. Very impressive work here! All combinations provided the correct words within <60% WER. Brilliant!
4. A concise summary of the results has been provided and with visualizations. Congratulations on figuring out that BIC performs best. Also, good job on providing ways on how to improve WER. Well done!

Excellent, all the specifications are met.

Notes

Here are some materials that can give ways on how to improve WER:

- The N-gram model can help in improving WER
- Implementing N-grams
- Computing N-grams

DOWNLOAD PROJECT

4 CODE REVIEW COMMENTS ❯

RETURN TO PATH

Student FAQ