UDACITY

< Return to Classroom

Plagiarism Detector

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Hello Udacity Student,
You have a good submission 👌. I saw your determination in completing this project which is such an admirable trait. Congratulations👏 on successful completion of this project. All the specifications have been met successfully. This demonstrate hard work and love for what you do. You showed great understanding of the course which is good. Keep it up and continue working hard. I look forward to your future submissions! Have a nice day.🏆

> I experienced many problems using the code provided in source_sklearn/train.py. It would be great if you can explain why I ran into these issues:

- import joblib failed
- default value of data-dir (environment variable) resulted in keyerror. I had to set the default value to None to make thing work.
- missing dependencies: I had to install s3sf in order to read the train data

I was not able to reproduce the errors you had. This could probably be with the updates made to the starter code as this was not the case when last I worked on the project. Nonetheless, you did a great job with the extra work to get thing moving.💪

All Required Files and Tests

The submission includes complete notebook files as `.ipynb`: "2_Plagiarism_Feature_Engineering" and "3_Training_a_Model". And the test and helper files are included: "problem_unittests.py", "helpers.py". The submission also includes a training directory `source_sklearn` OR `source_pytorch`.

Nice work submitting all the notebooks and project files as required for evaluation.👍

All the unit tests in project have passed.

Splendid! All the unit tests in have passed. 🎉

Notebook 2: DataFrame Pre-Processing

The function `numerical_dataframe` should be complete, reading in the original file_information.csv file and returning a DataFrame of information with a numerical `Category` column and new, `Class` column.

Good job with `numerical_dataframe()` function. It reads the original csv file and returned a DataFrame of information with a numerical `Category` column and new, `Class` column. 👏

There is no code requirement here, just make sure you run all required cells to create a `complete_df` that holds pre-processed file text data and `Datatype` information.

Nice! All required cells were run correctly to create a `complete_df` which holds the pre-processed file text data and Datatype information.

Notebook 2: Features Created

The function `calculate_containment` should be complete, taking in the necessary information and returning a single, normalized containment value for a given answer file.

Good job on the `calculate_containment` function as it gets a single, normalized containment value for every given answer file.👍

Pro Tips

The CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary.

- How to Prepare Text Data for Machine Learning with scikit-learn
- The difference between CountVectorizer token counts and TfidfTransformer
- Hacking Scikit-Learn's Vectorizers

Provide an answer to the question about containment feature calculation.

Well done! Your answer is reasonable for the containment feature calculation

The function `lcs_norm_word` should be complete, taking in two texts and returning a single, normalized LCS value.

Impressive implementation of `lcs_norm_word` to return a single, normalized LCS value, `LCS = 0.7407407407407407` 👏

Pro Tips

- Longest Common Subsequence Pseudo code
- Python Program for Longest Common Subsequence
- Rosetta code: Longest common subsequence

Define an n-gram range to calculate multiple containment features. Run the code to calculate one LCS feature, and create a DataFrame that holds all of these feature calculations.

Nice work defining the n-gram range and calculating all the containment features held in `all_features`. ⭐

Pro Tips

- Understanding the `ngram_range` argument in a CountVectorizer in sklearn
- Scikit learn ngram_range purpose in vectorizers

Notebook 2: Train and Test Files Created

Complete the function `train_test_data`. This should return only a *selection* of training and test features, and corresponding class labels.

Nice work with the function implementation to return only a selection of training and test features, and corresponding class labels.

Select at least three features to use in your final training and test data.

Nice here with the "good" features from the correlation matrix for your final training and test data. Good

choice.✔

Provide an answer that describes why you chose your final features.

Nice answer! Indeed, training and testing on those features with the least correlation will give more reliable results. 👏

Implement the `make_csv` function. The class labels for train/test data should be in the first column of the csv file; selected features in the rest of the columns. Run the rest of the cells to create `train.csv` and `test.csv` files.

Good work here, creating the `train.csv and test.csv` files correctly.

Comment

Good effort to use `.dropna(axis=0)` to drop any incomplete rows and avoid problem when dropping any empty rows especially when we concatenate labels and features.

Notebook 3: Data Upload

Upload the `train.csv` file to a specified directory in an S3 bucket.

`train.csv` is correctly uploaded into the data directory.👍

Notebook 3: Training a Custom Model

Complete at least *one* of the `train.py` files by instantiating a model, and training it in the main if statement. If you are using a custom PyTorch model, you will have to complete the `model.py` file, as well (you do not have to do so if you choose to use an imported sklearn model).

Well done completing the code in the `train.py` in `source_sklearn` directory. 👍

Pro Tips

- StackShare: PyTorch vs scikit-learn
- What are the differences in SciKit Learn, Keras, or Pytorch

Define a custom sklearn OR PyTorch estimator by passing in the required arguments.

Nice definition of your estimator. 👍

Pro Tips

- Choosing the right Estimator
- How to Choose Loss Functions
- SageMaker Estimators Documentation
- Linear Learner Algorithm Documentation

Fit your estimator (from the previous rubric item) to the training data you stored in S3.

Good job fitting your estimator to the training data as expected. ✔

Notebook 3: Deploying and Evaluating a Model

Deploy the model and create a `predictor` by specifying a deployment instance.

The `predictor` is created correctly with a deployment instance. Well done!

Pass test data to your deployed `predictor` and evaluate its performance by comparing its predictions to the true, class labels. Your model should get at least 90% test accuracy.

Nice Result! Your model acquired a perfect good accuracy of **96%** . 🎉

Provide an answer to the two model-related questions.

Nice response here!

Notebook 3: Cleaning up Resources

Run the code to clean up your final model resources.

Good work with the `clean up` of your final model resources. 💪

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START