Plagiarism Detector

Meets Specifications

Awesome you successfully used Amazon Sagemaker to train and deploy your model. Good luck. There are various other cloud services, each with competing capabilities and pricing, you should explore them too:
Azure DevOps and Google Cloud Platform offer many interesting capabilities including CI-CD.

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` .

Great job submitting all the necessary files. 👍

✓ All the unit tests in project have passed.

All the unit tests in the project 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.

Great, you correctly implemented `numerical_dataframe` function it returns a DataFrame of information with a numerical `Category` column and new, `Class` column.

And alternative pythonic way:

```
dict_1 = {'non': 0, 'heavy': 1, 'light': 2, 'cut': 3, 'orig': -1}
df['Category'] = [dict[c] for c in Category]
df['Class'] = [0 if c == 'non' else -1 if c == 'orig' else 1 for c in Category]
```

✓ 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.

Cool `complete_df` holds 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.

Great job getting the appropriate answer and source texts. You efficiently approached this in two main steps: creating a CountVectorizer to count n-grams and actualy calculating the intersection and normalized containment value.

Another cool way is

```
1.0 * sum(minimum(ngram_array[0],ngram_array[1]))/ngram_array[0].sum()
```

✓ Provide an answer to the question about containment feature calculation.

Perfect, the containment value depends only on the intersection between the answer_text and original Wikipedia text and on nothing else.

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

Awesome 😃 You handled each case—matching and non-matching words—very well with the help of dynamic programming.

Nice work using the structure of the matrix to grab the final LCS value in the last corner.

✓ 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.

Perfect implementation.

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.

✓ Select a few features to use in your final training and test data.

No choice is not always the best choice, but yes it works too 🙂

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

It is generally good to use one short and one long `c_n` to detect paraphrasing and copy-paste cases of plagiarism + `lcs_word`

✓ 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.

It may be useful to use `pd.DataFrame.dropna(axis=0)` to drop any incomplete rows.

Notebook 3: Data Upload

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

Good, use of Sagemaker `upload_data` function to upload all data neatly.

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).

Perfect

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

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

Notebook 3: Deploying and Evaluating a Model

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

Nice work deploying your model.

✓ 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.

Awesome in achieving 92% accuracy with `AdaBoostClassifier`

✓ Provide an answer to the two model-related questions.

Great work in searching, but to me, this appears brute force search, as you rightly mentioned in your comment. As you work with these algorithms, you will see that for this kind of problem, simple models like LinearSVC or MLPClassifier can also give more than 90% accuracy comfortably.

Notebook 3: Cleaning up Resources

✓ Run the code to clean up your final model resources.

Kindly ensure that all resources are deleted after use.

DOWNLOAD PROJECT

1 CODE REVIEW COMMENTS ›

RETURN TO PATH