UDACITY

< Return to Classroom

Plagiarism Detector

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Great work doing project implementation!! You did good on all necessary data preprocessing steps. This is very essential phase as model score eventually depends upon how well data is preprocessed before feeding it to model. If data is preprocessed efficiently then algorithm do their magic. You also learned and showcased AWS cloud skill to build, train and deploy ML model. Your test predictions looks great, nice strategy on experimenting with simplistic model first.

Congratulations for passing this project and All the best for other projects in this Nanodegree!!

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` .

Well done including all required files for submission!!

All the unit tests in project have passed.

Unit test shows passed for all cells.

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.

Well done mapping Category and Class columns to numeric representation.

```
# substitute categories by numeric values
df['Category'].replace(to_replace=replace_category_dict, inplace=True)

# create Class column based on the transformed Category column
df['Class'] = [1 if x > 0 else x for x in df['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.

complete_df correctly shows Text and Datatype column.

	File	Task	Category	Class	Text	Datatype
0	g0pA_taska.txt	a	0	0	inheritance is a basic concept of object orien...	train
1	g0pA_taskb.txt	b	3	1	pagerank is a link analysis algorithm used by ...	test
2	g0pA_taskc.txt	c	2	1	the vector space model also called term vector...	train
3	g0pA_taskd.txt	d	1	1	bayes theorem was names after rev thomas bayes...	train
4	g0pA_taske.txt	e	0	0	dynamic programming is an algorithm design tec...	train
5	g0pB_taska.txt	a	0	0	inheritance is a basic concept in object orien...	train
6	g0pB_taskb.txt	b	0	0	pagerank pr refers to both the concept and the...	train
7	g0pB_taskc.txt	c	3	1	vector space model is an algebraic model for r...	test
8	g0pB_taskd.txt	d	2	1	bayes theorem relates the conditional and marg...	train
9	g0pB_taske.txt	e	1	1	dynamic programming is a method for solving ma...	test

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.

Excellent!! Containment value is higher for highly plagiarized task and decreases as level of plagiarism is reduced.

```
Original category values:
[0, 3, 2, 1, 0]

3-gram containment values:
[0.009345794392523364, 0.9641025641025641, 0.6136363636363636, 0.15675675675675677, 0.031746031746031744]
```

Provide an answer to the question about containment feature calculation.

Good reasoning!! At this stage we haven't begun building our model yet we're in a preprocessing step that has to be done on both training and test data ,we're just extracting features out of our dataset, containment is calculated between a given answer text and its associated source text, so test and training data do not influence each other here.

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

Good work implementing lcs_norm_word using dynamic programming approach!!

```
LCS =  0.7407407407407407
Test passed!
```

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.

Good selection of ngram range.

```
# Define an ngram range
ngram_range = range(1,7)
```

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.

Good work creating train and test datasets!!

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

Good reasoning choosing ['c_1', 'c_5', 'lcs_word'] as the features represent least correlation and maximum variance.

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

Good reasoning justifying the selection of these features.

If the value is 1 then it suggest strong correlation and we should select feature with least correlation so that maximum variance is captured. Here feature c_1 and c_6 has least correlation of 0.87 so we can select these two feature. Good reasoning to select lcs_word as feature.

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.

Well done!! you can further use pd.DataFrame.dropna(axis=0) to drop any incomplete rows or empty rows.

```
def make_csv(x, y, filename, data_dir):
    '''Merges features and labels and converts them into one csv file with labels in the first column.
       :param x: Data features
       :param y: Data labels
       :param file_name: Name of csv file, ex. 'train.csv'
       :param data_dir: The directory where files will be saved
       '''
    # make data dir, if it does not exist
    if not os.path.exists(data_dir):
        os.makedirs(data_dir)


    # your code here
    df = pd.concat([pd.DataFrame(y), pd.DataFrame(x)], axis=1)
    df.to_csv(os.path.join(data_dir, filename), index=False, header=False)

    # nothing is returned, but a print statement indicates that the function has run
    print('Path created: '+str(data_dir)+'/'+str(filename))
```

Notebook 3: Data Upload

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

Well done uploading data to S3 cloud!!

```
# upload all data to S3
input_file=sagemaker_session.upload_data(data_dir,bucket,key_prefix=prefix)
```

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).

Good work defining SGDClassifier from sklearn library. Its best strategy to experiment with simplistic model first and eventually increase the complexity of model based on results. Complex model may also lead to overfitting.

```
## TODO: Define a model
model = SGDClassifier(max_iter=30)

## TODO: Train the model
model.fit(train_x, train_y)

## --- End of your code  --- ##
```

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

Well done creating estimator for sklearn model passing all necessary arguments!!

```
model= SKLearn(role=role,
                    instance_count=1,
                    instance_type='ml.c4.xlarge',
                    output_path=output_path,
                    entry_point='train.py',
                    source_dir="source_sklearn",
                    framework_version='0.23-1',
                    py_version='py3')
```

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

Well done fitting model with data!!

```
# Train your estimator on S3 training data
model.fit({'train':input_file})
```

Notebook 3: Deploying and Evaluating a Model

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

You have successfully deployed the model on 'ml.t2.medium'.

```
# deploy your model to create a predictor
predictor = model.deploy(initial_instance_count=1,instance_type='ml.t2.medium')
```

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.

Accuracy on test data is 88% for SGDClassifier model. Excellent!!

```
0.88

Predicted class labels:
[1 1 1 0 1 1 0 0 0 0 0 0 0 1 1 0 1 1 0 1 0 1 1 0 0]

True class labels:
[1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 0]
[[10  0]
 [ 3 12]]
```

Provide an answer to the two model-related questions.

Your selection of SGDClassifier model is justified. You have correctly evaluated false positives and false negatives!!

Notebook 3: Cleaning up Resources

Run the code to clean up your final model resources.

Model endpoints and S3 bucket is cleaned up after use!!

```
[ ]  # uncomment and fill in the line below!
     model.delete_endpoint()
```

```
The function delete_endpoint is a no-op in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
```

Deleting S3 bucket

When you are *completely* done with training and testing models, you can also delete your entire S3 bu training your model, you'll have to recreate your S3 bucket and upload your training data again.

```
[ ]  # deleting bucket, uncomment lines below

     bucket_to_delete = boto3.resource('s3').Bucket(bucket)
     bucket_to_delete.objects.all().delete()
```

here's reference to ensure all sagemaker resources are cleanup after use.
https://docs.aws.amazon.com/sagemaker/latest/dg/ex1-cleanup.html

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START