Amazon SageMaker Studio File Edit View Run Kernel Git Tabs Settings Help Feedback

/

Name	Last Modified
balanced	26 minutes ago
images	2 hours ago
src	43 minutes ago
C2_W1_Assignment.ipynb	11 minutes ago
feature_store_export.tsv	19 minutes ago

Terminal 1 | C2_W1_Assignment.ipynb | prepare_data.py

```python
########################################################################################################################################
########################################################## Import required modules #####################################################
########################################################################################################################################
from sklearn.model_selection import train_test_split
from sklearn.utils import resample
import functools
import multiprocessing

from datetime import datetime
from time import gmtime, strftime, sleep

import pandas as pd
import argparse
import subprocess
import sys
import os
import re
import collections
import json
import csv
import glob
from pathlib import Path
import time
import boto3

subprocess.check_call([sys.executable, "-m", "conda", "install", "-c", "pytorch", "pytorch==1.6.0", "-y"])

subprocess.check_call([sys.executable, "-m", "conda", "install", "-c", "conda-forge", "transformers==3.5.1", "-y"])
from transformers import RobertaTokenizer

subprocess.check_call([sys.executable, '-m', 'pip', 'install', 'sagemaker==2.35.0'])
import sagemaker

from sagemaker.session import Session
from sagemaker.feature_store.feature_group import FeatureGroup
from sagemaker.feature_store.feature_definition import (
    FeatureDefinition,
    FeatureTypeEnum,
)

########################################################################################################################################
######################################################## Setup environmental variables ################################################
########################################################################################################################################

region = os.environ['AWS_DEFAULT_REGION']
sts = boto3.Session(region_name=region).client(service_name='sts', region_name=region)
iam = boto3.Session(region_name=region).client(service_name='iam', region_name=region)
featurestore_runtime = boto3.Session(region_name=region).client(service_name='sagemaker-featurestore-runtime', region_name=region)
sm = boto3.Session(region_name=region).client(service_name='sagemaker', region_name=region)
```

1 1 Git: refreshing... Python Ln 188, Col 36 Spaces: 4 prepare_data.py

```python
assumed_role_name = assumed_role_arn.split('/')[-2]
get_role_response = iam.get_role(RoleName=assumed_role_name)
role = get_role_response['Role']['Arn']
bucket = sagemaker.Session().default_bucket()

sagemaker_session = sagemaker.Session(boto_session=boto3.Session(region_name=region),
                                      sagemaker_client=sm,
                                      sagemaker_featurestore_runtime_client=featurestore_runtime)

# List of sentiment classes: -1 - negative; 0 - neutral; 1 - positive
classes = [-1, 0, 1]

# Label IDs of the target class (sentiment) setup as a dictionary
classes_map = {
    -1: 0,
    0: 1,
    1: 2
}

# tokenization model
PRE_TRAINED_MODEL_NAME = 'roberta-base'

# create the tokenizer to use based on pre trained model
tokenizer = RobertaTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)

########################################################################################################################################
################################################################ Tools ###################################################################
########################################################################################################################################
# functions which can be considered as tools

def cast_object_to_string(data_frame):
    for label in data_frame.columns:
        if data_frame.dtypes[label] == 'object':
            data_frame[label] = data_frame[label].astype("str").astype("string")
    return data_frame


def wait_for_feature_group_creation_complete(feature_group):
    try:
        status = feature_group.describe().get("FeatureGroupStatus")
        print('Feature Group status: {}'.format(status))
        while status == "Creating":
            print("Waiting for Feature Group Creation")
            time.sleep(5)
```

Ln 188, Col 36 Spaces: 4 prepare_data.py

```python
        if status != "Created":
            print('Feature Group status: {}'.format(status))
            raise RuntimeError(f"Failed to create feature group {feature_group.name}")
        print(f"FeatureGroup {feature_group.name} successfully created.")
    except:
        print('No feature group created yet.')

def list_arg(raw_value):
    """argparse type for a list of strings"""
    return str(raw_value).split(',')

def to_sentiment(star_rating):
    if star_rating in {1, 2}: # negative
        return -1
    if star_rating == 3:      # neutral
        return 0
    if star_rating in {4, 5}: # positive
        return 1

########################################################################################################################################
################################################### Create or load Feature Group #######################################################
```

```
120 ##########################################################################################################################################
121
122 def create_or_load_feature_group(prefix, feature_group_name):
123
124     # Feature Definitions for the records
125     feature_definitions= [
126         FeatureDefinition(feature_name='review_id', feature_type=FeatureTypeEnum.STRING),
127         FeatureDefinition(feature_name='date', feature_type=FeatureTypeEnum.STRING),
128         FeatureDefinition(feature_name='sentiment', feature_type=FeatureTypeEnum.STRING),
129         FeatureDefinition(feature_name='label_id', feature_type=FeatureTypeEnum.STRING),
130         FeatureDefinition(feature_name='input_ids', feature_type=FeatureTypeEnum.STRING),
131         FeatureDefinition(feature_name='review_body', feature_type=FeatureTypeEnum.STRING),
132         FeatureDefinition(feature_name='split_type', feature_type=FeatureTypeEnum.STRING)
133     ]
134
135     # setup the Feature Group
136     feature_group = FeatureGroup(
137         name=feature_group_name,
138         feature_definitions=feature_definitions,
139         sagemaker_session=sagemaker_session
140     )
141
142     print('Feature Group: {}'.format(feature_group))
```

Ln 188, Col 36 Spaces: 4 prepare_data.py

```
145         print('Waiting for existing Feature Group to become available if it is being created by another instance in our cluster...')
146         wait_for_feature_group_creation_complete(feature_group)
147     except Exception as e:
148         print('Before CREATE FG wait exeption: {}'.format(e))
149
150     try:
151         record_identifier_feature_name = "review_id"
152         event_time_feature_name = "date"
153
154         print('Creating Feature Group with role {}...'.format(role))
155
156         # create Feature Group
157         feature_group.create(
158             s3_uri=f"s3://{bucket}/{prefix}",
159             record_identifier_name=record_identifier_feature_name,
160             event_time_feature_name=event_time_feature_name,
161             role_arn=role,
162             enable_online_store=False
163         )
164         print('Creating Feature Group. Completed.')
165
166         print('Waiting for new Feature Group to become available...')
167         wait_for_feature_group_creation_complete(feature_group)
168         print('Feature Group available.')
169
170         # the information about the Feature Group
171         feature_group.describe()
172
173     except Exception as e:
174         print('Exception: {}'.format(e))
175
176     return feature_group
177
178 ##########################################################################################################################################
179 ############################################################ Tokenization of the reviews ##################################################
180 ##########################################################################################################################################
181
182 # Convert the review into the BERT input ids using
183 def convert_to_bert_input_ids(review, max_seq_length):
184     encode_plus = tokenizer.encode_plus(
185           review,
186           add_special_tokens=True,
187           ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
188           max_length=max_seq_length, # Replace None
```

Ln 188, Col 36 Spaces: 4 prepare_data.py

```
191           padding='max_length',
192           return_attention_mask=True,
193           return_tensors='pt',
194           truncation=True
195     )
196
197     return encode_plus['input_ids'].flatten().tolist()
198
199 ##########################################################################################################################################
200 ############################################################ Parse input arguments ########################################################
201 ##########################################################################################################################################
202
203 def parse_args():
204     # Unlike SageMaker training jobs (which have `SM_HOSTS` and `SM_CURRENT_HOST` env vars), processing jobs to need to parse the resource config file
    directly
205     resconfig = {}
206     try:
207         with open('/opt/ml/config/resourceconfig.json', 'r') as cfgfile:
208             resconfig = json.load(cfgfile)
209     except FileNotFoundError:
210         print('/opt/ml/config/resourceconfig.json not found. current_host is unknown.')
211         pass # Ignore
212
213     # Local testing with CLI args
214     parser = argparse.ArgumentParser(description='Process')
215
216     parser.add_argument('--hosts', type=list_arg,
217         default=resconfig.get('hosts', ['unknown']),
218         help='Comma-separated list of host names running the job'
219     )
220     parser.add_argument('--current-host', type=str,
221         default=resconfig.get('current_host', 'unknown'),
222         help='Name of this host running the job'
223     )
224     parser.add_argument('--input-data', type=str,
225         default='/opt/ml/processing/input/data',
226     )
227     parser.add_argument('--output-data', type=str,
228         default='/opt/ml/processing/output',
229     )
230     parser.add_argument('--train-split-percentage', type=float,
231         default=0.90,
232     )
233     parser.add_argument('--validation-split-percentage', type=float,
```

Ln 188, Col 36 Spaces: 4 prepare_data.py

```
236     parser.add_argument('--test-split-percentage', type=float,
237         default=0.05,
238     )
239     parser.add_argument('--balance-dataset', type=eval,
240         default=True
241     )
242     parser.add_argument('--max-seq-length', type=int,
```

```
        default=128
    )
    parser.add_argument('--feature-store-offline-prefix', type=str,
        default=None,
    )
    parser.add_argument('--feature-group-name', type=str,
        default=None,
    )

    return parser.parse_args()

########################################################################################################################################################
######################################################## Processing functions ###############################################################
########################################################################################################################################################

def _preprocess_file(file,
                     balance_dataset,
                     max_seq_length,
                     prefix,
                     feature_group_name):

    print('file {}'.format(file))
    print('balance_dataset {}'.format(balance_dataset))
    print('max_seq_length {}'.format(max_seq_length))
    print('prefix {}'.format(prefix))
    print('feature_group_name {}'.format(feature_group_name))

    # Create a feature group
    # the Feature Group that was set in the main notebook cannot be passed here - it will be used later in the notebook for other purposes
    # you need to create a Feature Group with the same Feature Definitions within the processing job
    feature_group = create_or_load_feature_group(prefix, feature_group_name)

    filename_without_extension = Path(Path(file).stem).stem

    # read file
    df = pd.read_csv(file,
                     index_col=0)
```

Ln 188, Col 36 Spaces: 4 prepare_data.py

```
    df = df.dropna()
    df = df.reset_index(drop=True)
    print('Shape of dataframe {}'.format(df.shape))

    # convert star rating into sentiment
    df['sentiment'] = df['Rating'].apply(lambda star_rating: to_sentiment(star_rating=star_rating))
    print('Shape of dataframe with sentiment {}'.format(df.shape))

    # convert sentiment (-1, 0, 1) into label_id (0, 1, 2)
    df['label_id'] = df['sentiment'].apply(lambda sentiment: classes_map[sentiment])
    print('df[label_id] after using classes_map: {}'.format(df['label_id']))

    df['input_ids'] = df['Review Text'].apply(lambda review: convert_to_bert_input_ids(review, max_seq_length))
    print('df[input_ids] after calling convert_to_bert_input_ids: {}'.format(df['input_ids']))

    # convert the index into a review_id
    df.reset_index(inplace=True)
    df = df.rename(columns = {'index': 'review_id',
                              'Review Text': 'review_body'})

    # drop all columns except the following:
    df = df[['review_id', 'sentiment', 'label_id', 'input_ids', 'review_body']]
    df = df.reset_index(drop=True)

    print(df.head())

    print('Shape of dataframe after dropping columns {}'.format(df.shape))

    # balance the dataset by sentiment down to the minority class
    if balance_dataset:

        df_unbalanced_grouped_by = df.groupby('sentiment')
        df_balanced = df_unbalanced_grouped_by.apply(lambda x: x.sample(df_unbalanced_grouped_by.size().min()).reset_index(drop=True))
        print('Shape of balanced df: {}'.format(df_balanced.shape))

        print(df_balanced['sentiment'].head())

        df = df_balanced

    # adding timestamp as date column
    timestamp = datetime.now().strftime("%Y-%m-%dT%H:%M:%SZ")
    print(timestamp)

    df['date'] = timestamp
```

Ln 188, Col 36 Spaces: 4 prepare_data.py

```
    # split dataset
    print('Shape of dataframe before splitting {}'.format(df.shape))

    print('train split percentage {}'.format(args.train_split_percentage))
    print('validation split percentage {}'.format(args.validation_split_percentage))
    print('test split percentage {}'.format(args.test_split_percentage))

    holdout_percentage = 1.00 - args.train_split_percentage
    print('holdout percentage {}'.format(holdout_percentage))
    df_train, df_holdout = train_test_split(df,
                                            test_size=holdout_percentage,
                                            stratify=df['sentiment'])

    test_holdout_percentage = args.test_split_percentage / holdout_percentage
    print('test holdout percentage {}'.format(test_holdout_percentage))
    df_validation, df_test = train_test_split(df_holdout,
                                              test_size=test_holdout_percentage,
                                              stratify=df_holdout['sentiment'])

    df_train = df_train.reset_index(drop=True)
    df_validation = df_validation.reset_index(drop=True)
    df_test = df_test.reset_index(drop=True)

    print('Shape of train dataframe {}'.format(df_train.shape))
    print('Shape of validation dataframe {}'.format(df_validation.shape))
    print('Shape of test dataframe {}'.format(df_test.shape))

    train_data = '{}/sentiment/train'.format(args.output_data)
    validation_data = '{}/sentiment/validation'.format(args.output_data)
    test_data = '{}/sentiment/test'.format(args.output_data)

    ## write TSV Files
    df_train.to_csv('{}/part-{}-{}.tsv'.format(train_data, args.current_host, filename_without_extension), sep='\t', index=False)
    df_validation.to_csv('{}/part-{}-{}.tsv'.format(validation_data, args.current_host, filename_without_extension), sep='\t', index=False)
    df_test.to_csv('{}/part-{}-{}.tsv'.format(test_data, args.current_host, filename_without_extension), sep='\t', index=False)

    # dataframe
    df_train.head()
    df_validation.head()
```

```
    df_test.head()

    column_names = ['review_id', 'sentiment', 'date', 'label_id', 'input_ids', 'review_body']

    df_train_records = df_train[column_names]
```



```
    df_validation_records = df_validation[column_names]
    df_validation_records['split_type'] = 'validation'
    df_validation_records.head()

    df_test_records = df_test[column_names]
    df_test_records['split_type'] = 'test'
    df_test_records.head()

    # add record to feature store
    df_fs_train_records = cast_object_to_string(df_train_records)
    df_fs_validation_records = cast_object_to_string(df_validation_records)
    df_fs_test_records = cast_object_to_string(df_test_records)

    print('Ingesting features...')
    feature_group.ingest(
        data_frame=df_fs_train_records, max_workers=3, wait=True
    )
    feature_group.ingest(
        data_frame=df_fs_validation_records, max_workers=3, wait=True
    )
    feature_group.ingest(
        data_frame=df_fs_test_records, max_workers=3, wait=True
    )

    offline_store_status = None
    while offline_store_status != 'Active':
        try:
            offline_store_status = feature_group.describe()['OfflineStoreStatus']['Status']
        except:
            pass
        print('Offline store status: {}'.format(offline_store_status))
        sleep(15)
    print('...features ingested!')


def process(args):
    print('Current host: {}'.format(args.current_host))

    feature_group = create_or_load_feature_group(prefix=args.feature_store_offline_prefix,
                                                 feature_group_name=args.feature_group_name)

    feature_group.describe()
```



```
    validation_data = '{}/sentiment/validation'.format(args.output_data)
    test_data = '{}/sentiment/test'.format(args.output_data)

    # partial functions allow to derive a function with some parameters to a function with fewer parameters
    # and fixed values set for the more limited function
    # here 'preprocess_file' will be more limited function than '_preprocess_file' with fixed values for some parameters
    preprocess_file = functools.partial(_preprocess_file,
                                        balance_dataset=args.balance_dataset,
                                        max_seq_length=args.max_seq_length,
                                        prefix=args.feature_store_offline_prefix,
                                        feature_group_name=args.feature_group_name)

    input_files = glob.glob('{}/*.csv'.format(args.input_data))

    num_cpus = multiprocessing.cpu_count()
    print('num_cpus {}'.format(num_cpus))

    p = multiprocessing.Pool(num_cpus)
    p.map(preprocess_file, input_files)

    print('Listing contents of {}'.format(preprocessed_data))
    dirs_output = os.listdir(preprocessed_data)
    for file in dirs_output:
        print(file)

    print('Listing contents of {}'.format(train_data))
    dirs_output = os.listdir(train_data)
    for file in dirs_output:
        print(file)

    print('Listing contents of {}'.format(validation_data))
    dirs_output = os.listdir(validation_data)
    for file in dirs_output:
        print(file)

    print('Listing contents of {}'.format(test_data))
    dirs_output = os.listdir(test_data)
    for file in dirs_output:
        print(file)

    print('Complete')

##############################################################################################################################
############################################################## Main ##########################################################
```



```
if __name__ == "__main__":

    args = parse_args()
    print('Loaded arguments:')
    print(args)

    print('Environment variables:')
    print(os.environ)

    process(args)
```