Assignment 4 Report

Yan Kang (yk2848), Ava Yang (ty2418), Zi Zhuang(zz2693)

Abstract:

Through this assignment, we have employed a CNN model in order to complete the sentiment analysis. Ahead of implementing the designed algorithm, preprocessing our existing dataset is a crucial step. The major tool we used for preprocessing provided raw data is from the last assignment - our self-written python library. By utilizing our written library, we are able to exclude unnecessary information and transform our raw data into workable strings with given sentiment scores.

Step 1 (Pre Processing ETL) :

First, we added a function named “load_embedding_dictionary” to our preprocessing library, so that it can load the dictionary word list from a zip file directly without unzipping it.

```python
import os
import zipfile

class embedding():
    def __init__(self,max_length_dictionary):
        self.max_length_dictionary=max_length_dictionary

    def load_embedding_dictionary(self,dictionary_path):


        self.embedding_dictionary = {}

        embeddings = []

        if ".zip/" in dictionary_path:
            archive_path = os.path.abspath(dictionary_path)

            split = archive_path.split(".zip/")

            archive_path = split[0] + ".zip/"
            path_inside = split[0]

            archive = zipfile.ZipFile(archive_path, "r")
            embeddings = archive.read(path_inside).decode("utf8").split("\n")

        else:

            embeddings = open(dictionary_path, "r", encoding="utf8").read().split("\n")

        for index, row in enumerate(embeddings):

            split = row.split(" ")

            if index == self.max_length_dictionary:
                return

            self.embedding_dictionary[split[0]] = index
```

Then we modified the dictionary: We added a zero vector at the top of the dictionary to match the pad sequence. We also added another line of vector below to indicate the unknown tag.

glove.twitter_aicloud.27B.25d

```
<pad> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
<unknown> 0.24634 -0.4477 -0.43908 -0.77769 -1.4601 -1.114 -0.41041 -1.0084 0.95986
```

At last, we created a zip archive containing the preprocessing library and the modified dictionary and uploaded it to S3.

Step 2 (Run the Pre Processing on the dataset) :

First, we shuffled and split the dataset into 3 different datasets “train”, “dev” and “eval” where “train” is 85% of the data, “dev” is 10% and “eval” is 5% of the data. And we uploaded all 3 datasets in 3 separate directories and ran crawlers on them.

	Name	Schedule	Status	Logs	Last runtime	Median runtime	Tables updated	Tables added
	test		Ready	Logs	1 min	1 min	0	1
	train		Ready	Logs	52 secs	52 secs	0	1
	val		Ready	Logs	1 min	1 min	0	1

We then created a glue job to preprocess our data. The python script is screenshotted below:

```python
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from pre_processing.pre_processing import PreProcessor

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])
Preprocessor=PreProcessor()
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
## @type: DataSource
## @args: [database = "model", table_name = "test_sample_csv", transformation_ctx = "datasource0"]
## @return: datasource0
## @inputs: []
datasource0 = glueContext.create_dynamic_frame.from_catalog(database = "model", table_name = "test_sample_csv",  transformation_ctx = "datasource0")
## @type: ApplyMapping
## @args: [mapping = [("sentiment", "long", "sentiment", "long"), ("tweet", "string", "tweet", "string")], transformation_ctx = "applymapping1"]
## @return: applymapping1
## @inputs: [frame = datasource0]
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("sentiment", "long", "sentiment", "long"), ("tweet", "string", "tweet", "string")], transformation_ctx = "applymapping1")
## @type: Map
## @args: [f = map_function, transformation_ctx = "mapping1"]
## @return: mapping1
## @inputs: [frame = applymapping1 ]
def map_function(dynamicRecord):
    tweet=dynamicRecord["tweet"]
    features=Preprocessor.pre_process_text(tweet)
    dynamicRecord["features"]=features
    return dynamicRecord
mapping1 = Map.apply(frame = applymapping1, f = map_function, transformation_ctx = "mapping1")

## @type: DataSink
## @args: [connection_type = "s3", connection_options = {"path": "s3://bucketkangtwitter/model/data"}, format = "json", transformation_ctx = "datasink2"]
## @return: datasink2
## @inputs: [frame = mapping1]
datasink2 = glueContext.write_dynamic_frame.from_options(frame = mapping1, connection_type = "s3", connection_options = {"path": "s3://bucketkangtwitter/model/data"}, format = "json", transformation_ctx = "datasink2")
job.commit()
```

Our proofs of successfulness are attached below:

	Run ID	Retry attempt	Run status	Error	Logs	Error logs	Glue version	Maximum capacity	Triggered by	Start time	End time	Execution time	Timeout	Delay	Job run input
	jr_6b8c9bd3dd65…	-	Succeeded		Logs		1.0	5		22 Feb…	22 Feb…	1 min	2880 mins		s3://aws-glue-te…

	Run ID	Retry attempt	Run status	Error	Logs	Error logs	Glue version	Maximum capacity	Triggered by	Start time	End time	Execution time	Timeout	Delay	Job run input
	jr_8de7e8d44141…	-	Succeeded		Logs		1.0	5		22 Feb…	22 Feb…	1 min	2880 mins		s3://aws-glue-te…

	Run ID	Retry attempt	Run status	Error	Logs	Error logs	Glue version	Maximum capacity	Triggered by	Start time	End time	Execution time	Timeout	Delay	Job run input
	jr_ad55184e4e86…	-	Succeeded		Logs		1.0	5		22 Feb…	22 Feb…	1 min	2880 mins		s3://aws-glue-te…

We hence have three separate processed data files.

Step 3 (CNN Modeling) :

Starting from this step, we implemented our written CNN model. We firstly created an embedding matrix with size 500000*25 and fed it into embedding layer's initial weights. We then ran through the model locally to obtain our built model.

You may look at our code on Github.

Our result is shown below:

```
model_training — -bash — 80×24
85/85 [==============================] - 9s 106ms/step - loss: 0.4562 - acc: 0.7
807 - val_loss: 0.5381 - val_acc: 0.7095
Epoch 5/10
85/85 [==============================] - 9s 107ms/step - loss: 0.4123 - acc: 0.8
089 - val_loss: 0.5377 - val_acc: 0.7244
Epoch 6/10
85/85 [==============================] - 9s 111ms/step - loss: 0.3571 - acc: 0.8
451 - val_loss: 0.5584 - val_acc: 0.7234
Epoch 7/10
85/85 [==============================] - 9s 108ms/step - loss: 0.3147 - acc: 0.8
653 - val_loss: 0.5987 - val_acc: 0.7363
Epoch 8/10
85/85 [==============================] - 9s 107ms/step - loss: 0.2617 - acc: 0.8
965 - val_loss: 0.6510 - val_acc: 0.7264
Epoch 9/10
85/85 [==============================] - 9s 109ms/step - loss: 0.2252 - acc: 0.9
142 - val_loss: 0.7194 - val_acc: 0.7124
Epoch 10/10
85/85 [==============================] - 9s 110ms/step - loss: 0.1792 - acc: 0.9
340 - val_loss: 0.7577 - val_acc: 0.7204
Test loss:0.7113824367523194
Test accuracy:0.7434343695640564
2020-02-23 17:22:48.023088: W tensorflow/python/util/util.cc:280] Sets are not c
urrently considered sequences, but this may change in the future, so consider av
```

Step 4 (SageMaker Jupyter Notebook) :

As requested, we ran through our code once again through SageMaker Notebook. To do this computation, some modifications of codes are performed. We first need to change our configuration file by changing the embedding path.

```
{
    "embeddings_dictionary_size": 500000,
    "embeddings_vector_size": 25,
    "padding_size": 20,
    "batch_size": 100,
    "embeddings_path": "s3://bucketkangtwitter/model/glove.twitter_aicloud.27B.25d.txt",
    "input_tensor_name": "embedding_input"
}
```

To run through the entire code, we used the terminal on SageMaker Jupyter. We modified our codes once again to give S3 access to Jupyter. Our result is shown below:

```
Call initializer instance with the dtype argument instead of passing it to the constructor
Starting training...
Epoch 1/10
85/85 [==============================] - 15s 181ms/step - loss: 0.6448 - acc: 0.6244 - val_loss: 0.5938 - val_acc: 0.6677
Epoch 2/10
85/85 [==============================] - 15s 179ms/step - loss: 0.5643 - acc: 0.7047 - val_loss: 0.5760 - val_acc: 0.6915
Epoch 3/10
85/85 [==============================] - 15s 180ms/step - loss: 0.5080 - acc: 0.7466 - val_loss: 0.5433 - val_acc: 0.7085
Epoch 4/10
85/85 [==============================] - 15s 179ms/step - loss: 0.4573 - acc: 0.7807 - val_loss: 0.5378 - val_acc: 0.7114
Epoch 5/10
85/85 [==============================] - 15s 175ms/step - loss: 0.4115 - acc: 0.8109 - val_loss: 0.5298 - val_acc: 0.7284
Epoch 6/10
85/85 [==============================] - 15s 173ms/step - loss: 0.3559 - acc: 0.8442 - val_loss: 0.5403 - val_acc: 0.7403
Epoch 7/10
85/85 [==============================] - 15s 173ms/step - loss: 0.3134 - acc: 0.8684 - val_loss: 0.5628 - val_acc: 0.7423
Epoch 8/10
85/85 [==============================] - 16s 185ms/step - loss: 0.2619 - acc: 0.8931 - val_loss: 0.6365 - val_acc: 0.7124
Epoch 9/10
85/85 [==============================] - 15s 171ms/step - loss: 0.2211 - acc: 0.9138 - val_loss: 0.6432 - val_acc: 0.7214
Epoch 10/10
85/85 [==============================] - 16s 185ms/step - loss: 0.1799 - acc: 0.9312 - val_loss: 0.6920 - val_acc: 0.7373
Test loss:0.6831464409828186
Test accuracy:0.7373737096786499
2020-02-24 00:09:15.937854: W tensorflow/python/util/util.cc:280] Sets are not currently considered sequences, but this may
 change in the future, so consider avoiding using them.
```