Amazon SageMaker

Table of Contents

[Sentiment Analysis 1](#_Toc120276611)

[Preparing the Environment 1](#_Toc120276612)

[Training 2](#_Toc120276613)

[Evaluate 2](#_Toc120276614)

[Obtaining and Viewing All Predictions 3](#_Toc120276615)

[Save and Load the Model 4](#_Toc120276616)

[Continuous Training 4](#_Toc120276617)

## Sentiment Analysis

This section is based on the quick start tutorial at:

<https://auto.gluon.ai/stable/tutorials/multimodal/text_prediction/beginner_text.html>

### Preparing the Environment

An error occurs when running the code for the tutorial. The error may occur because ipywidgets 8 conflicts with the other libraries.

To view your version of ipywidgets run the following code:

|  |
| --- |
| import ipywidgets  ipywidgets.\_\_version\_\_ |

To downgrade to a more stable version of ipywidgets, like version 7.0.0, run the following command and remember to restart the kernel when finished.

|  |
| --- |
| pip install ipywidgets==7.0.0 |

Running the code in this next step loads the Stanford Sentiment Treebank ([SST](https://nlp.stanford.edu/sentiment/)) dataset, which consists of movie reviews and their associated sentiment. Given a new movie review, the goal is to predict the sentiment reflected in the text (in this case a **binary classification**, where reviews are labeled as 1 if they convey a positive opinion and labeled as 0 otherwise). Let’s first load and look at the data, noting the labels are stored in a column called **label**.

The parquet format here is used but of course you could load this data from a csv file or other data source.

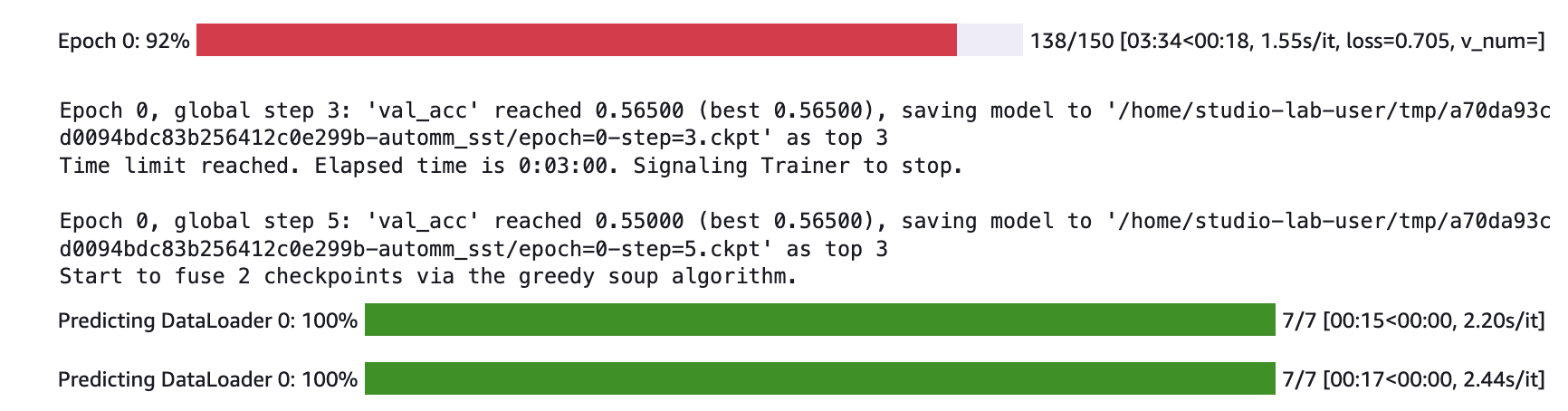
|  |
| --- |
| %matplotlib inline  import numpy as np  import warnings  import matplotlib.pyplot as plt  warnings.filterwarnings('ignore')  np.random.seed(123)  from autogluon.core.utils.loaders import load\_pd  train\_data = load\_pd.load('https://autogluon-text.s3-accelerate.amazonaws.com/glue/sst/train.parquet')  test\_data = load\_pd.load('https://autogluon-text.s3-accelerate.amazonaws.com/glue/sst/dev.parquet')  subsample\_size = 1000 # subsample data for faster demo, try setting this to larger values  train\_data = train\_data.sample(n=subsample\_size, random\_state=0)  train\_data.head(10) |

### Training

To ensure this tutorial runs quickly, we call fit() with a subset of 1000 training examples and limit its runtime to three minutes. To achieve reasonable performance in your applications, you are recommended to set much longer time\_limit (eg. 1 hour), or do not specify time\_limit at all (time\_limit=None).

|  |
| --- |
| from autogluon.multimodal import MultiModalPredictor  import uuid  model\_path = f"./tmp/{uuid.uuid4().hex}-automm\_sst"  predictor = MultiModalPredictor(label='label', eval\_metric='acc', path=model\_path)  predictor.fit(train\_data, time\_limit=180) |

Interestingly enough, the time limit of 60 seconds ran out and the model fitting was halted. The model likely is not very good since even the first epoch was not completed.



### Evaluate

After training, we can easily evaluate our predictor on separate test data formatted similarly to our training data.

|  |
| --- |
| test\_score = predictor.evaluate(test\_data, metrics=['acc', 'f1'])  print(test\_score) |

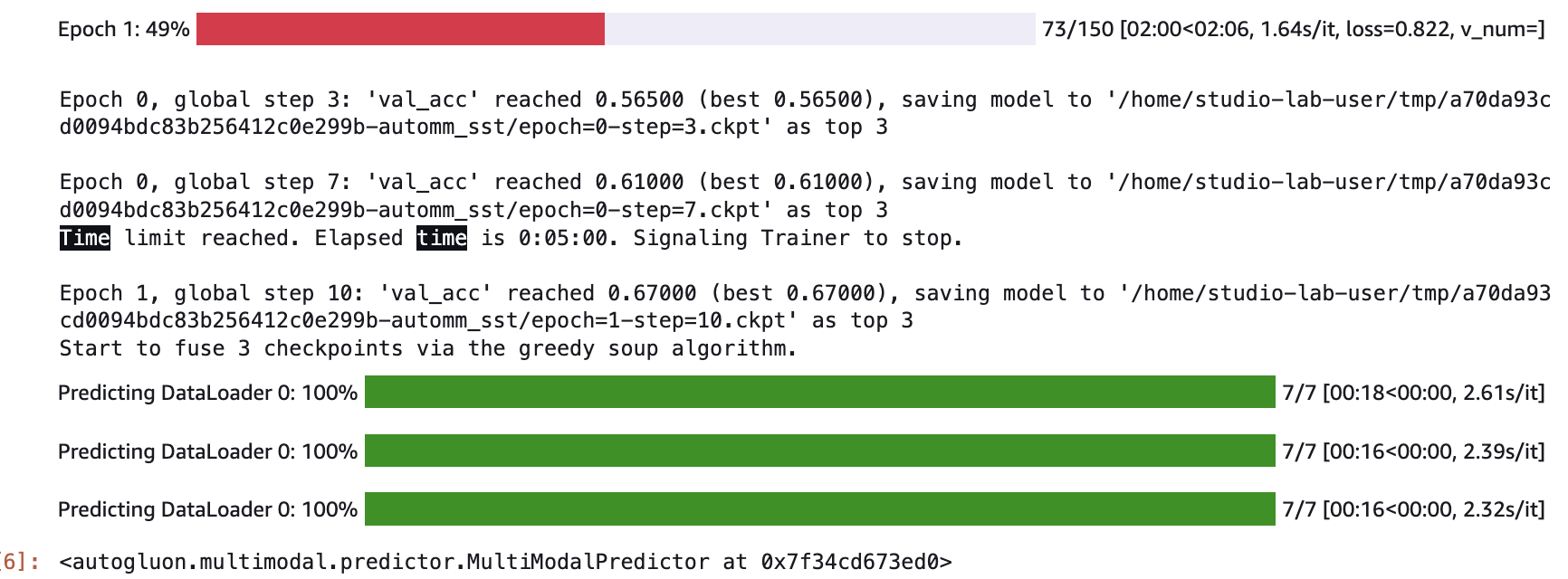
The prediction is poor:

|  |
| --- |
| {'acc': 0.5229357798165137, 'f1': 0.6799999999999999} |

Here we will extend the training to 5 minutes. Run this code to retrain the model with a longer duration.

|  |
| --- |
| predictor = MultiModalPredictor(label='label', eval\_metric='acc', path=model\_path)  predictor.fit(train\_data, time\_limit=5\*60) |

The model still does not finish training so more time could have been added. For the sake of saving time though I will leave it as is.



After the model retrains the second time, evaluate it again with this code:

|  |
| --- |
| test\_score = predictor.evaluate(test\_data, metrics=['acc', 'f1', 'recall', 'precision'])  print(test\_score) |

A significant improvement is evident however not all of the epochs were completed so likely more training time could have been allocated.

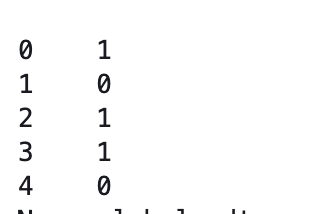
For the sake of saving time I will leave it as is since the accuracy is decent:

|  |
| --- |
| {'acc': 0.716743119266055, 'f1': 0.6754270696452037, 'recall': 0.5788288288288288, 'precision': 0.8107255520504731} |

### Obtaining and Viewing All Predictions

You could show predictions for the entire data set.

|  |
| --- |
| test\_predictions = predictor.predict(test\_data)  test\_predictions.head() |



To compare this with actual values run this code:

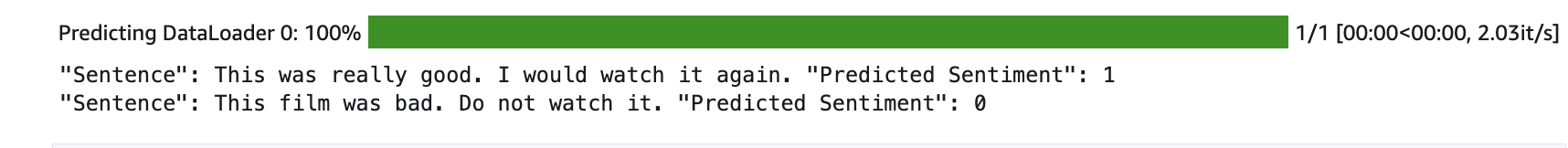
|  |
| --- |
| print(test\_data.head(5)) |

When comparing the actual values with the predictions we can see only one prediction is incorrect but the rest are accurate.



This code tests the model with new data:

|  |
| --- |
| sentence1 = "This was really good. I would watch it again."  sentence2 = "This film was bad. Do not watch it."  predictions = predictor.predict({'sentence': [sentence1, sentence2]})  print('"Sentence":', sentence1, '"Predicted Sentiment":', predictions[0])  print('"Sentence":', sentence2, '"Predicted Sentiment":', predictions[1]) |



Exercise 1 (4 marks)

Test the model with a sentence with your name in it. Make sure you say something nice about yourself. Then show a screenshot here. Show your code here:

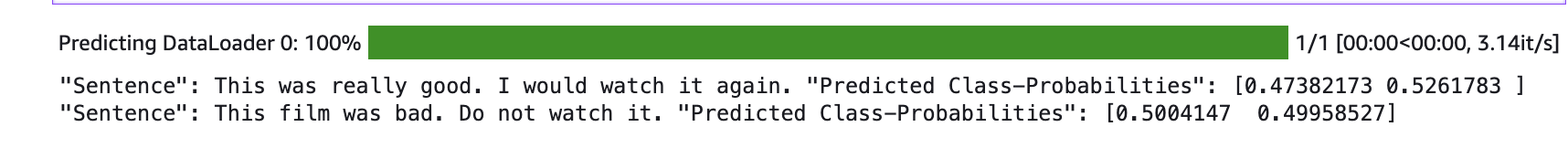
|  |
| --- |
|  |

Show your output here:

|  |
| --- |
|  |

It is also possible to show classification probabilities. I notice that the probabilities were not very distinct so it tells me that the model is currently not very good.

|  |
| --- |
| probs = predictor.predict\_proba({'sentence': [sentence1, sentence2]})  print('"Sentence":', sentence1, '"Predicted Class-Probabilities":', probs[0])  print('"Sentence":', sentence2, '"Predicted Class-Probabilities":', probs[1]) |



Exercise 2 (1 mark)

Show a screenshot of probabilities that appear for the sentence with your name in it.

|  |
| --- |
|  |

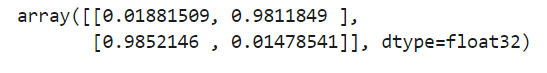
### Save and Load the Model

The trained predictor is automatically saved at the end of .**fit()**, and you can easily reload it.

You can also save the predictor to any location by calling **.save()**.

|  |
| --- |
| new\_model\_path = "mymodel"  predictor.save(new\_model\_path)  loaded\_predictor = MultiModalPredictor.load(new\_model\_path)  loaded\_predictor.predict\_proba({'sentence': [sentence1, sentence2]}) |

When making predictions with predict\_proba the probabilities are displayed.



### Continuous Training

It is also possible to resume training in another session. To do this, load the predictor, call .fit() again to continue training the same predictor with new data.

|  |
| --- |
| new\_predictor = MultiModalPredictor.load(new\_model\_path)  new\_predictor.fit(train\_data, time\_limit=30)  test\_score = new\_predictor.evaluate(test\_data, metrics=['acc', 'f1'])  print(test\_score) |