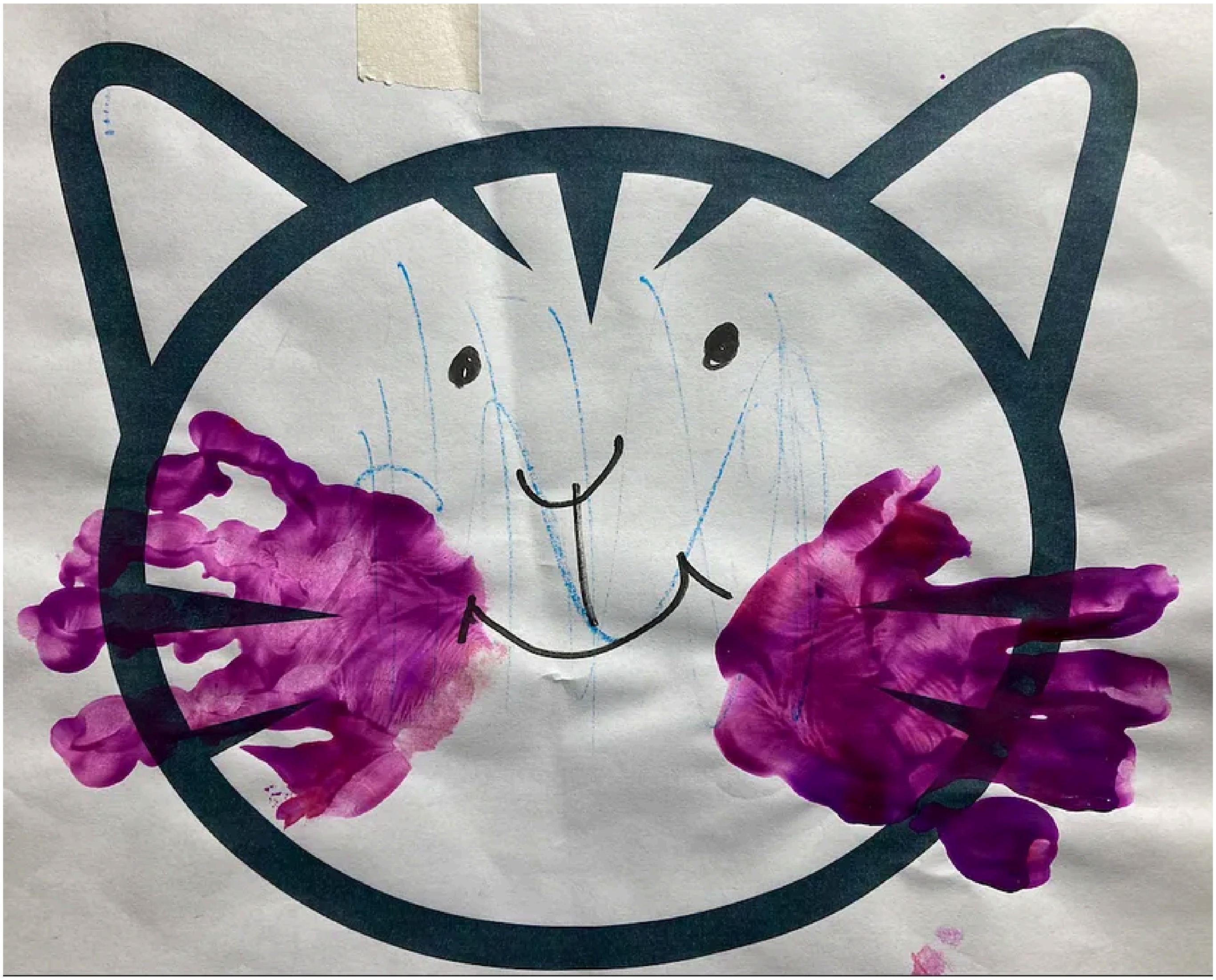
Train and deploy a text classification model with Spark NLP, BERT transfer learning, MLflow, and Databricks

Thomas Jaensch

Published in Towards Dev · 6 min read · Feb 6, 2023

9



Art my preschool daughter created

Step by step instructions how to train a binary text classification model via transfer learning on a pretrained **BERT** (Bidirectional Encoder Representations from Transformers) model and make batch predictions with the fine-grained model on new data

GitHub repo with complete notebook code **here**

Create a Databricks ML cluster with the Spark NLP library installed

Databricks comes with Machine Learning runtimes that have many libraries that one may want to use for anything ML already preinstalled. Unfortunately, Spark NLP isn't one of them and the process to set up a cluster with Spark NLP **is not** straightforward and easy but finicky and error prone. The exact instructions how to go about this can be found in the official documentation here, but you definitely need to edit the cluster's Spark config with additional "spark.jars.packages" AND install the same libraries via the cluster's "Libraries" tab.

Once you think you've configured your cluster correctly, start it up, create a Python Databricks notebook, attach it to the newly minted cluster, and run the following code:

```
spark.sparkContext.getConf().get('spark.jars.packages')
```

If everything works out, the cell above reports back one line with the jars installed. If you don't see anything, something isn't quite right just yet.

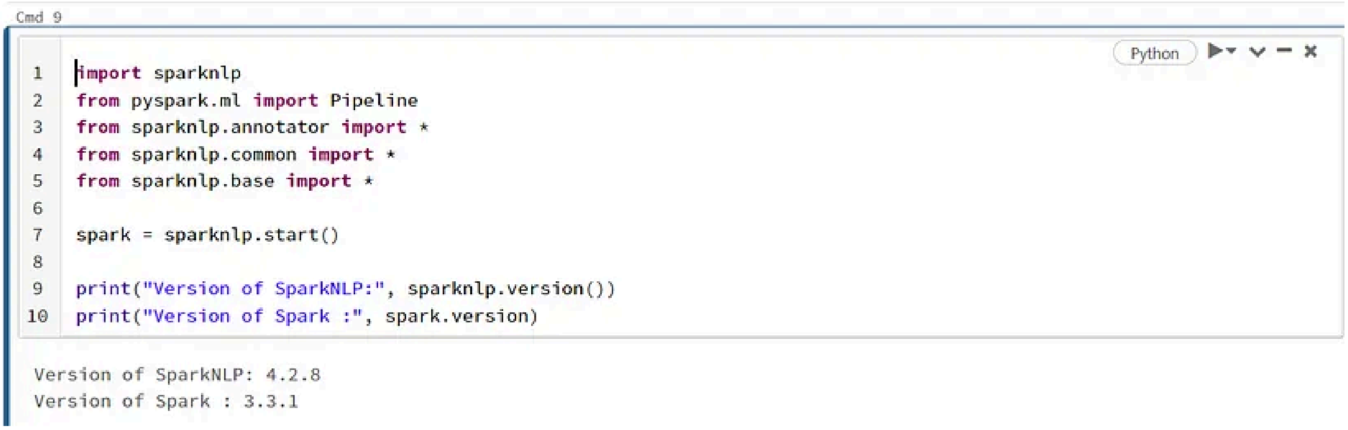
Import Spark NLP libraries and start Spark session

In the notebook, create a new cell and run the following code:

```
import sparknlp
from pyspark.ml import Pipeline
from sparknlp.annotator import *
from sparknlp.common import *
from sparknlp.base import *

spark = sparknlp.start()
```

```
print("Version of SparkNLP:", sparknlp.version())
print("Version of Spark :", spark.version)
```

Output should look something like this:



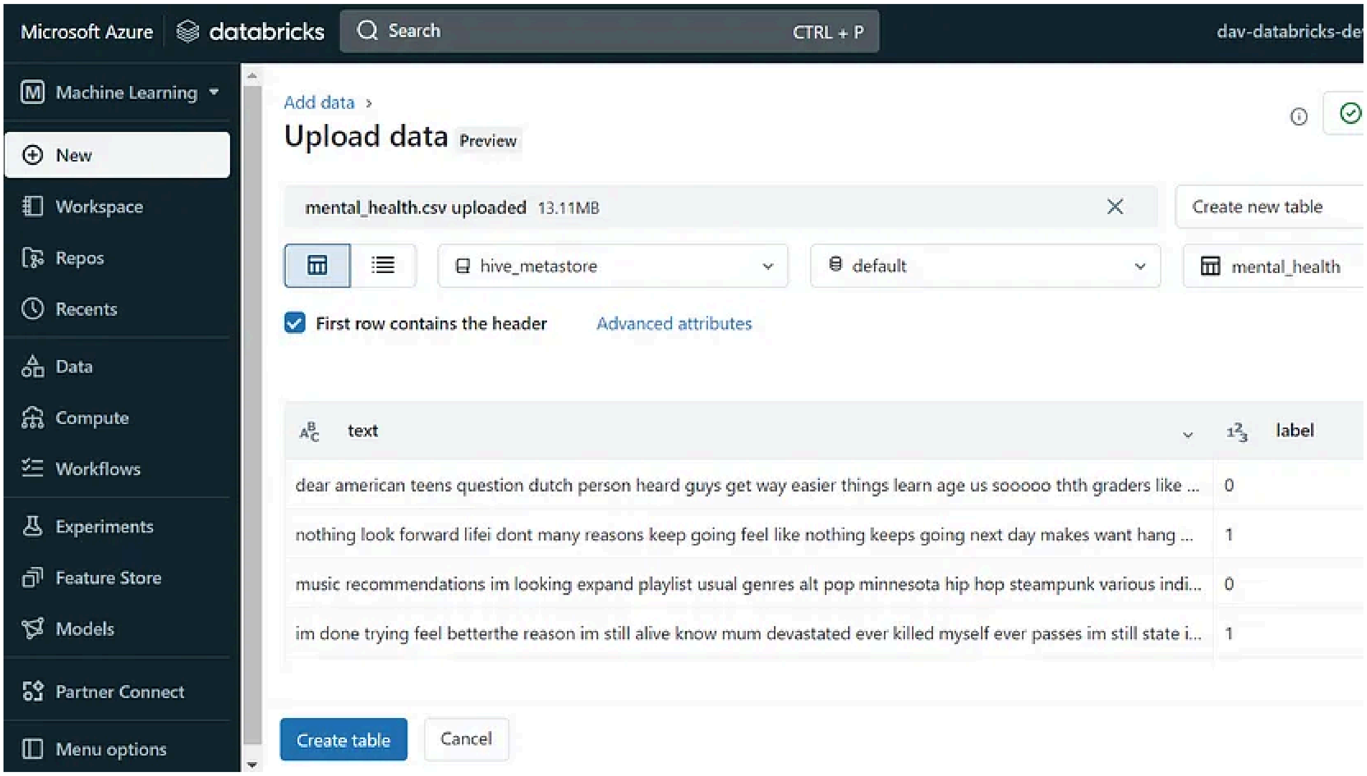
```
import sparknlp
from pyspark.ml import Pipeline
from sparknlp.annotator import *
from sparknlp.common import *
from sparknlp.base import *

spark = sparknlp.start()

print("Version of SparkNLP:", sparknlp.version())
print("Version of Spark :", spark.version)
```

```
Version of SparkNLP: 4.2.8
Version of Spark : 3.3.1
```

Load and transform the Kaggle dataset used for transfer learning on the BERT model

For this demo, I am going to use a Kaggle dataset called Mental Health Corpus that contains about 30K text snippets indicating if someone may suffer from anxiety, depression, and other mental health issues, or not.

From the Kaggle page, download the *archive.zip* file, unpack, and then create a table in the Databricks **default** database. Just select the unpacked CSV file and click “Create table”, like shown below:



Upload data Preview

mental_health.csv uploaded 13.11MB

Create new table

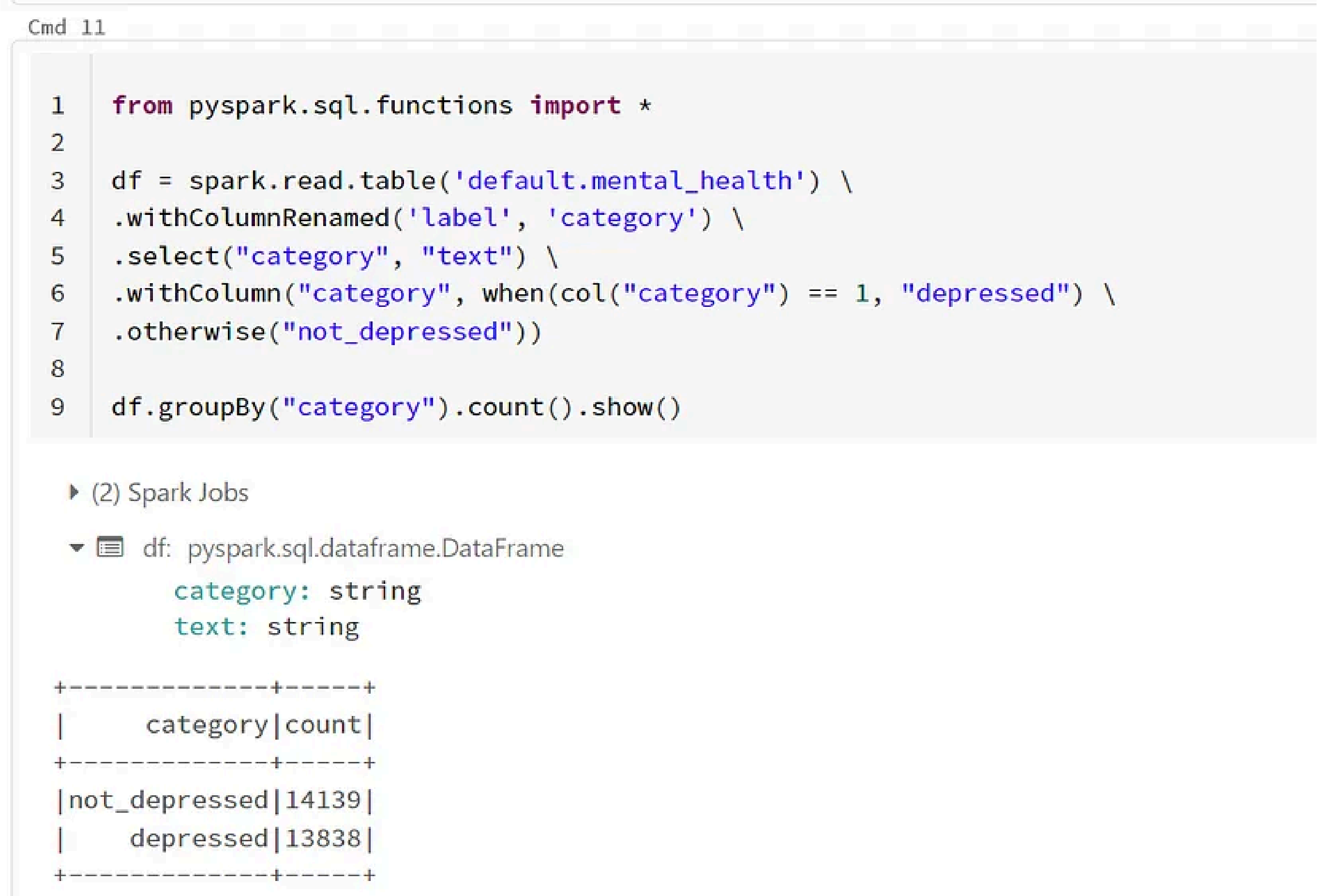
hive_metastore | default | mental_health

☑ First row contains the header   Advanced attributes

text	label
dear american teens question dutch person heard guys get way easier things learn age us sooooo thth graders like ...	0
nothing look forward lifei dont many reasons keep going feel like nothing keeps going next day makes want hang ...	1
music recommendations im looking expand playlist usual genres alt pop minnesota hip hop steampunk various indi...	0
im done trying feel betterthe reason im still alive know mum devastated ever killed myself ever passes im still state i...	1

In the notebook, in a new cell, load the contents of the previously created table into a Spark dataframe, do some ETL, and see the counts of both text columns:

```
from pyspark.sql.functions import *

df = spark.read.table('default.mental_health') \
.withColumnRenamed('label', 'category') \
.select("category", "text") \
.withColumn("category", when(col("category") == 1, "depressed") \
.otherwise("not_depressed"))

df.groupBy("category").count().show()
```

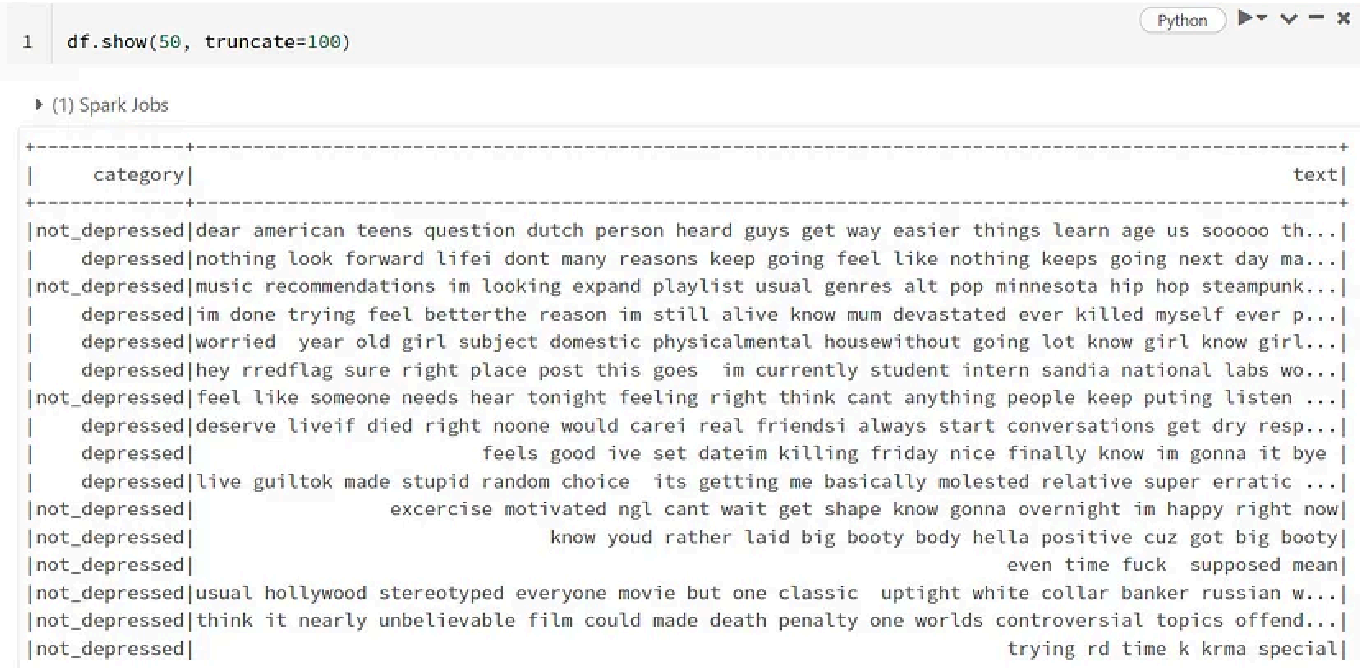


```python
from pyspark.sql.functions import *

df = spark.read.table('default.mental_health') \
.withColumnRenamed('label', 'category') \
.select("category", "text") \
.withColumn("category", when(col("category") == 1, "depressed") \
.otherwise("not_depressed"))

df.groupBy("category").count().show()
```

```
df:  pyspark.sql.dataframe.DataFrame
      category: string
      text: string

+-------------+-----+
|     category|count|
+-------------+-----+
|not_depressed|14139|
|    depressed|13838|
+-------------+-----+
```

Inspect a few of the text columns:

```python
df.show(50, truncate=100)
```



```
df.show(50, truncate=100)
```

```
+-------------+----------------------------------------------------------------------------------------------------+
|     category|                                                                                                text|
+-------------+----------------------------------------------------------------------------------------------------+
|not_depressed|dear american teens question dutch person heard guys get way easier things learn age us sooooo th...|
|    depressed|nothing look forward lifei dont many reasons keep going feel like nothing keeps going next day ma...|
|not_depressed|music recommendations im looking expand playlist usual genres alt pop minnesota hip hop steampunk...|
|    depressed|im done trying feel betterthe reason im still alive know mum devastated ever killed myself ever p...|
|    depressed|worried  year old girl subject domestic physicalmental housewithout going lot know girl know girl...|
|    depressed|hey rredflag sure right place post this goes  im currently student intern sandia national labs wo...|
|not_depressed|feel like someone needs hear tonight feeling right think cant anything people keep puting listen ...|
|    depressed|deserve liveif died right noone would carei real friendsi always start conversations get dry resp...|
|    depressed|                         feels good ive set dateim killing friday nice finally know im gonna it bye |
|    depressed|live guiltok made stupid random choice  its getting me basically molested relative super erratic ...|
|not_depressed|                 excercise motivated ngl cant wait get shape know gonna overnight im happy right now|
|not_depressed|                               know youd rather laid big booty body hella positive cuz got big booty|
|not_depressed|                                                                       even time fuck  supposed mean|
|not_depressed|usual hollywood stereotyped everyone movie but one classic  uptight white collar banker russian w...|
|not_depressed|think it nearly unbelievable film could made death penalty one worlds controversial topics offend...|
|not_depressed|                                                                       trying rd time k krma special|
```

Split the dataset into training and testing sets

```
train_text, test_text = df.randomSplit([0.8, 0.2], seed = 12345)
```

Start MLflow for experiment and model tracking

MLflow, a platform for the Machine Learning lifecycle, comes built in on Databricks ML runtimes and is already integrated with the Databricks ML workspace. To use it for this experiment, run the following in the next notebook cell:

```
import mlflow
from mlflow.models import Model, infer_signature, ModelSignature

mlflow_run = mlflow.start_run()
```

```
# Signature
signature = infer_signature(test_text)
```

Classification with BERT Sentence Embeddings

To avoid having to train a Machine Learning model from scratch which also probably wouldn't yield good results on a small dataset with only about 30K data points anyway, we are going to do transfer learning on an already pre-trained large language model, BERT LaBSE Sentence Embeddings, via Spark NLP. To be able to classify text, the BERT sentence embeddings will be combined with ClassifierDLApproach in a DocumentAssembler -> BertSentenceEmbeddings -> ClassifierDLApproach Spark NLP pipeline, like this:

```
document = DocumentAssembler()\
    .setInputCol("text")\
    .setOutputCol("document")

embeddings = BertSentenceEmbeddings\
    .pretrained("labse", "xx") \
    .setInputCols(["document"])\
    .setOutputCol("sentence_embeddings")

classsifierdl = ClassifierDLApproach()\
   .setInputCols(["sentence_embeddings"])\
   .setOutputCol("class")\
   .setLabelColumn("category")\
   .setMaxEpochs(10)\
   .setEnableOutputLogs(True)

nlp_pipeline_bert = Pipeline(
    stages=[document,
            embeddings,
            classsifierdl])
```

Time to train the model on the training dataset defined earlier:

```
classification_model_bert = nlp_pipeline_bert.fit(train_text)
```

Log the model in MLflow and build a reference to the model URI

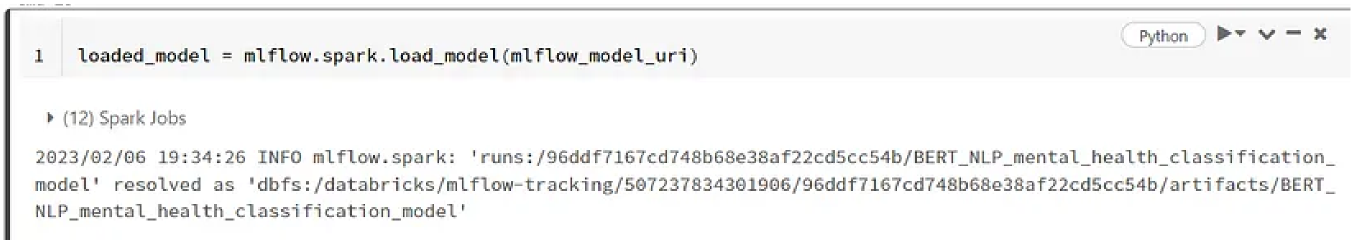
Once the model is trained, track it in Databricks Experiments via MLflow:

```
import pandas as pd

input_example = pd.DataFrame([{"index": 0, "text": "I'm lost in the darkness and

conda_env = {
    'channels': ['conda-forge'],
    'dependencies': [
        'python=3.9.5',
        {
            "pip": [
              'pyspark==3.1.2',
              'mlflow<3,>=2.1',
              'spark-nlp==4.2.8'
            ]
        }
    ],
    'name': 'mlflow-env'
}

model_name = "BERT_NLP_mental_health_classification_model"

mlflow.spark.log_model(classification_model_bert, model_name, conda_env=conda_en

mlflow.log_artifacts("com.johnsnowlabs.nlp:spark-nlp_2.12:4.2.8")

mlflow.end_run()

mlflow_model_uri = "runs:/{}/{}".format(mlflow_run.info.run_id, model_name)
```

```
display(mlflow_model_uri)
```

After the model has been logged, it can be loaded back into this or any other notebook with a suitable Databricks runtime from MLflow Experiment tracking and used for batch inference:

```
loaded_model = mlflow.spark.load_model(mlflow_model_uri)
```



```
loaded_model = mlflow.spark.load_model(mlflow_model_uri)
```

```
2023/02/06 19:34:26 INFO mlflow.spark: 'runs:/96ddf7167cd748b68e38af22cd5cc54b/BERT_NLP_mental_health_classification_
model' resolved as 'dbfs:/databricks/mlflow-tracking/507237834301906/96ddf7167cd748b68e38af22cd5cc54b/artifacts/BERT_
NLP_mental_health_classification_model'
```

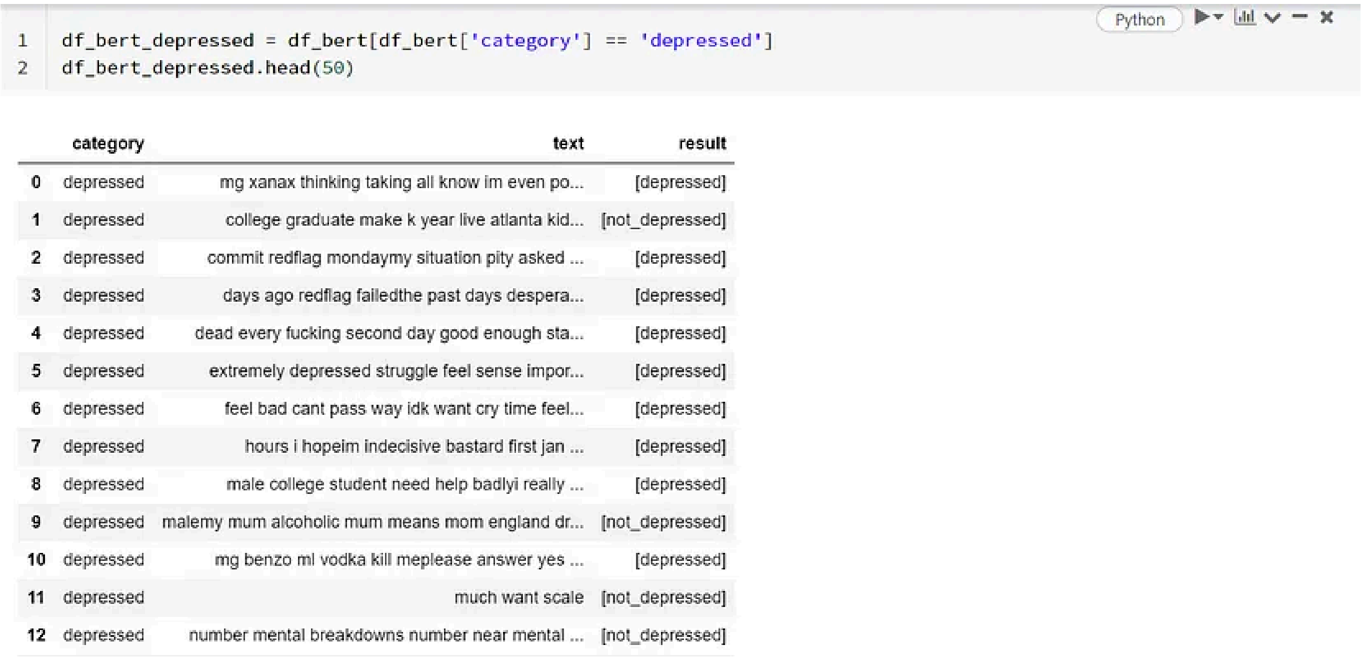
Run predictions on test dataset

Let’s see how this model trained on BERT sentence embeddings does on the test dataset:

```
df_bert = loaded_model.transform(test_text).select("category", "text", "class.re
```

View some ‘depressed’ predictions:

```
df_bert_depressed = df_bert[df_bert['category'] == 'depressed']
df_bert_depressed.head(50)
```

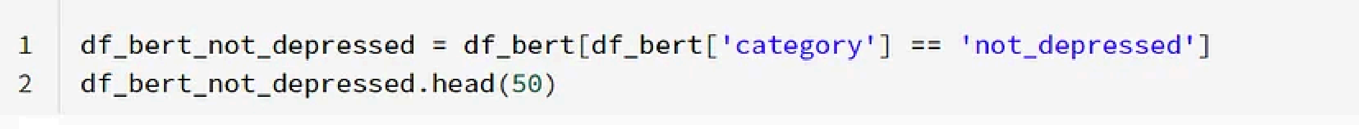


```
df_bert_depressed = df_bert[df_bert['category'] == 'depressed']
df_bert_depressed.head(50)
```

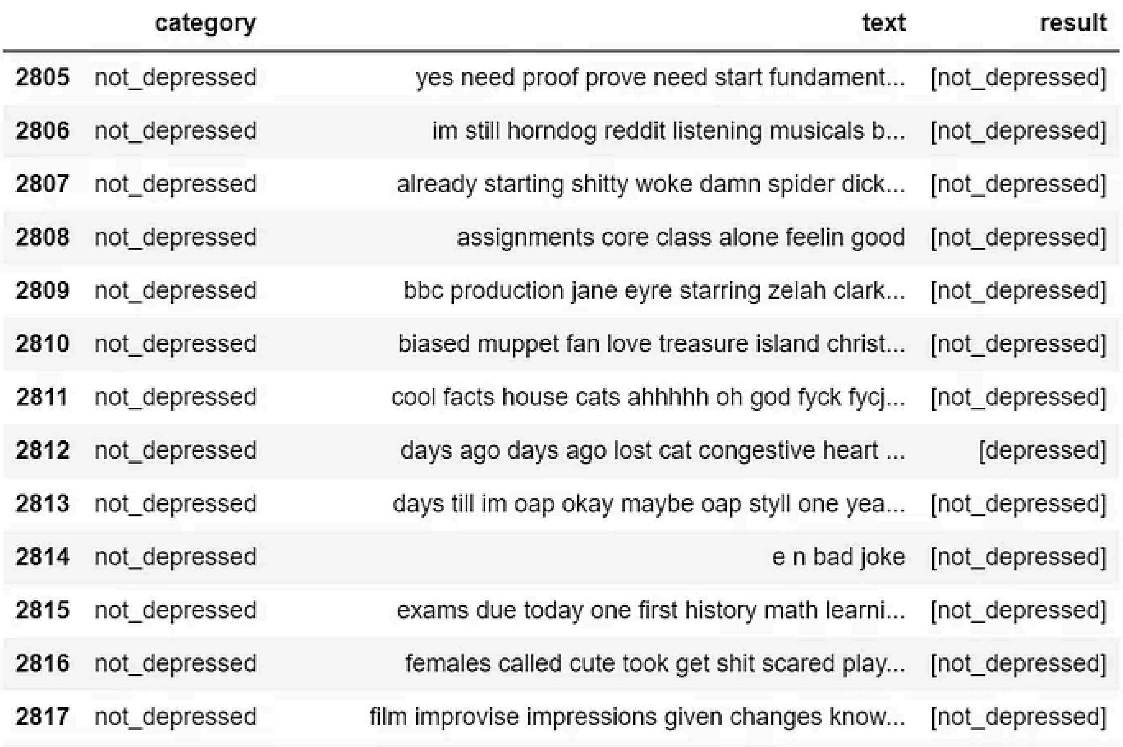
	category	text	result
0	depressed	mg xanax thinking taking all know im even po...	[depressed]
1	depressed	college graduate make k year live atlanta kid...	[not_depressed]
2	depressed	commit redflag mondaymy situation pity asked ...	[depressed]
3	depressed	days ago redflag failedthe past days despera...	[depressed]
4	depressed	dead every fucking second day good enough sta...	[depressed]
5	depressed	extremely depressed struggle feel sense impor...	[depressed]
6	depressed	feel bad cant pass way idk want cry time feel...	[depressed]
7	depressed	hours i hopeim indecisive bastard first jan ...	[depressed]
8	depressed	male college student need help badlyi really ...	[depressed]
9	depressed	malemy mum alcoholic mum means mom england dr...	[not_depressed]
10	depressed	mg benzo ml vodka kill meplease answer yes ...	[depressed]
11	depressed	much want scale	[not_depressed]
12	depressed	number mental breakdowns number near mental ...	[not_depressed]

View some ‘not_depressed’ predictions:

```
df_bert_not_depressed = df_bert[df_bert['category'] == 'not_depressed']
df_bert_not_depressed.head(50)
```



```
df_bert_not_depressed = df_bert[df_bert['category'] == 'not_depressed']
df_bert_not_depressed.head(50)
```

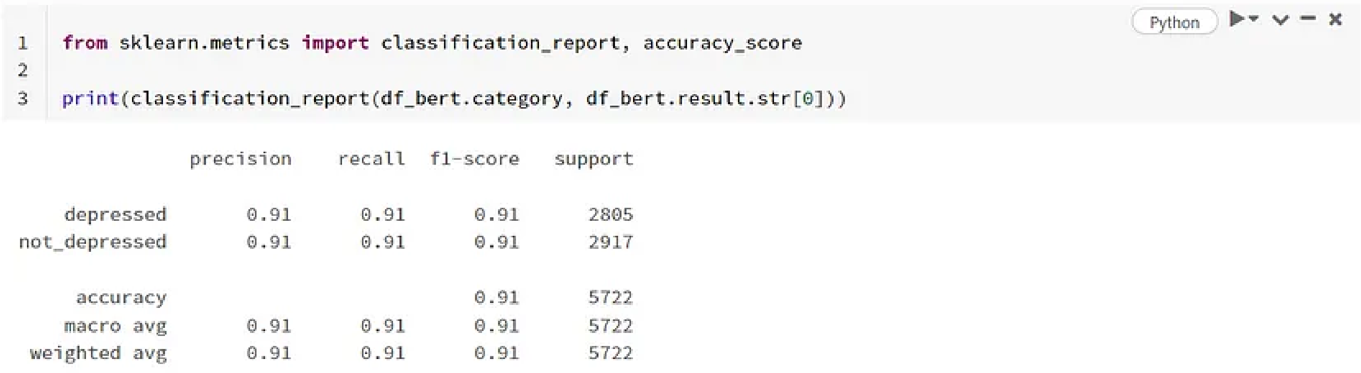


	category	text	result
2805	not_depressed	yes need proof prove need start fundament...	[not_depressed]
2806	not_depressed	im still horndog reddit listening musicals b...	[not_depressed]
2807	not_depressed	already starting shitty woke damn spider dick...	[not_depressed]
2808	not_depressed	assignments core class alone feelin good	[not_depressed]
2809	not_depressed	bbc production jane eyre starring zelah clark...	[not_depressed]
2810	not_depressed	biased muppet fan love treasure island christ...	[not_depressed]
2811	not_depressed	cool facts house cats ahhhhh oh god fyck fycj...	[not_depressed]
2812	not_depressed	days ago days ago lost cat congestive heart ...	[depressed]
2813	not_depressed	days till im oap okay maybe oap styll one yea...	[not_depressed]
2814	not_depressed	e n bad joke	[not_depressed]
2815	not_depressed	exams due today one first history math learni...	[not_depressed]
2816	not_depressed	females called cute took get shit scared play...	[not_depressed]
2817	not_depressed	film improvise impressions given changes know...	[not_depressed]

It seems like overall the model isn't all that bad at classifying the text snippets with the correct label but has trouble when the text is really short. On the other hand, how do you classify "much want scale" or "e n bad joke" in "depressed" or "not_depressed"…?

Let's create a Scikit-learn Classification report to see the overall accuracy on the entire test dataset:

```
from sklearn.metrics import classification_report, accuracy_score

print(classification_report(df_bert.category, df_bert.result.str[0]))
```



```python
from sklearn.metrics import classification_report, accuracy_score

print(classification_report(df_bert.category, df_bert.result.str[0]))
```

```
               precision    recall  f1-score   support

    depressed       0.91      0.91      0.91      2805
not_depressed       0.91      0.91      0.91      2917

     accuracy                           0.91      5722
    macro avg       0.91      0.91      0.91      5722
 weighted avg       0.91      0.91      0.91      5722
```

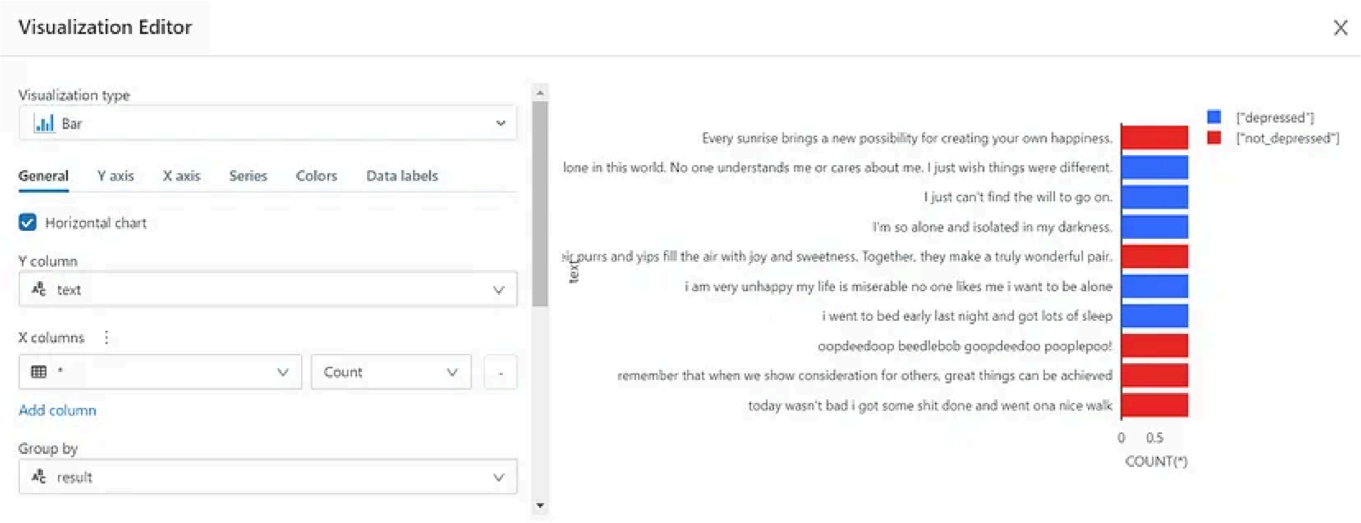
That comes down to 91% accuracy which means the model approximately classifies 9 out of 10 text snippets correctly from the test dataset. Regarding the small amount of code written, the size of the training set, and the overall effort put into creating this model that's a pretty decent outcome I would say.

Let's create a new dataframe with unlabeled text snippets to test the functionality of the model on previously unseen data that may also be somewhat different in style, etc. than the data in the training corpus:

```python
df_new = spark.createDataFrame([
    (1, "I'm so alone and isolated in my darkness."),
    (2, "remember that when we show consideration for others, great things can b
    (3, "oopdeedoop beedlebob goopdeedoo pooplepoo!"),
    (4, "i am very unhappy my life is miserable no one likes me i want to be alo
    (5, "i went to bed early last night and got lots of sleep"),
    (6, "today wasn't bad i got some shit done and went ona nice walk"),
    (7, "Kitties and doggies snuggle up together, giving each other the love and
    (8, "I feel like I'm all alone in this world. No one understands me or cares
    (9, "Every sunrise brings a new possibility for creating your own happiness.
    (10, "I just can't find the will to go on.")
]).toDF("id", "text")
```

See prediction results:

```
df = loaded_model.transform(df_new).select("text", "class.result").toPandas()
display(df)
```



Visualization Editor

Visualization type: Bar

General | Y axis | X axis | Series | Colors | Data labels

☑ Horizontal chart

Y column: text

X columns: * | Count

Add column

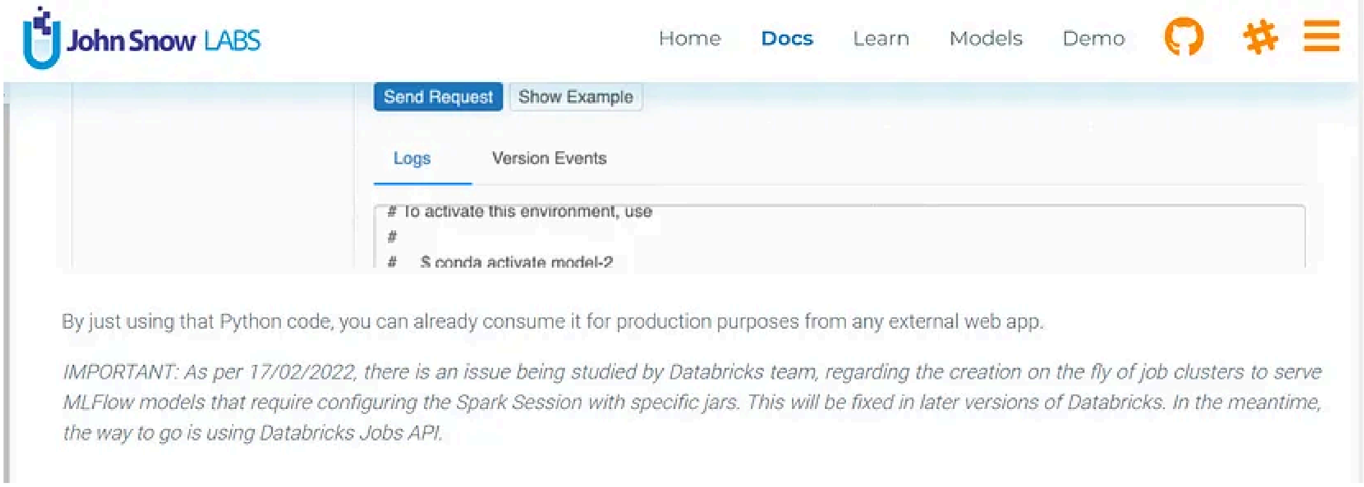
Group by: result

["depressed"]
["not_depressed"]
Every sunrise brings a new possibility for creating your own happiness.
lone in this world. No one understands me or cares about me. I just wish things were different.
I just can't find the will to go on.
I'm so alone and isolated in my darkness.
ir purrs and yips fill the air with joy and sweetness. Together, they make a truly wonderful pair.
i am very unhappy my life is miserable no one likes me i want to be alone
i went to bed early last night and got lots of sleep
oopdeedoop beedlebob goopdeedoo pooplepoo!
remember that when we show consideration for others, great things can be achieved
today wasn't bad i got some shit done and went ona nice walk
text
0
0.5
COUNT(*)

Well, that looks mostly right, no?

Bottom Line

Spark NLP definitely has a learning curve and is not easy to install correctly and without hiccups on a Databricks cluster, but once set up it is fairly straightforward to do Natural Language Processing ML using it and build models that exceed the playground stage with relatively little code. The integration with Databricks could definitely use improvement and simplification, and I also haven't been able to get the real-time model serving going with it because of what's mentioned here (I think):



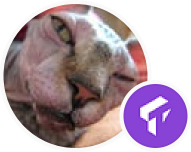
```
# To activate this environment, use
#
#     $ conda activate model-2
```

By just using that Python code, you can already consume it for production purposes from any external web app.

IMPORTANT: As per 17/02/2022, there is an issue being studied by Databricks team, regarding the creation on the fly of job clusters to serve MLFlow models that require configuring the Spark Session with specific jars. This will be fixed in later versions of Databricks. In the meantime, the way to go is using Databricks Jobs API.

Other than that, the possibilities seem endless and I will definitely look into this more if the need arises to build something NLP with Spark and/or Databricks.

NLP Machine Learning Mlflow Transfer Learning Databricks



Written by Thomas Jaensch

Edit profile

21 Followers · Writer for Towards Dev

WFH, staring at code