Text Classification in Spark NLP with Bert and Universal Sentence Encoders

Training a SOTA multi-class text classifier with Bert and Universal Sentence Encoders in Spark NLP with just a few lines of code in less than 10 min.

Veysel Kocaman Apr 12 · 11 min read

Photo by AbsolutVision on Unsplash

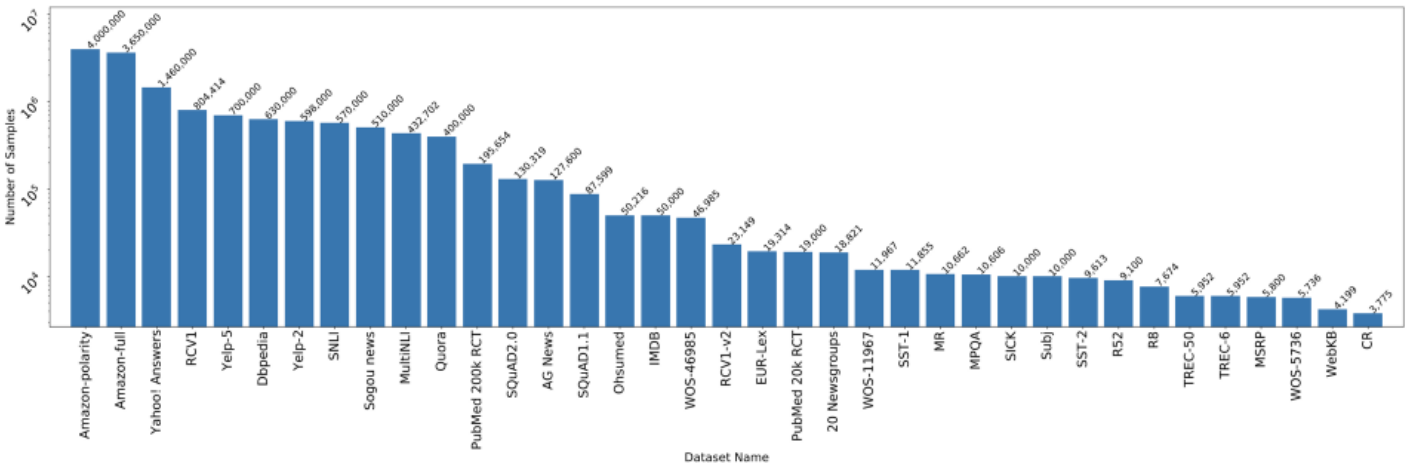
Natural language processing (NLP) is a key component in many data science systems that must understand or reason about a text. Common use cases include text classification, question answering, paraphrasing or summarising, sentiment analysis, natural language BI, language modeling, and disambiguation.

NLP is essential in a growing number of AI applications. Extracting accurate information from free text is a must if you are building a chatbot, searching through a patent database, matching patients to clinical trials, grading customer service or sales calls, extracting facts from financial reports or solving for any of these 44 use cases across 17 industries.

Text classification is one of the main tasks in modern NLP and it is the task of assigning a sentence or document an appropriate category. The categories depend on the chosen dataset and can range from topics.

Every text classification problem follows similar steps and is being solved with different algorithms. Let alone classical and popular Machine Learning classifiers like Random Forest or Logistic Regression, there are more than 150 deep learning frameworks proposed for various text classification problems.

There are several benchmark datasets being used in Text Classification problems and the latest benchmarks can be tracked on nlpprogress.com. Here is the basic statistics regarding these datasets.



Text classification benchmark datasets

A simple text classification application usually follows these steps:

- Text preprocessing & cleaning
- Feature engineering (creating handcrafted features from text)
- Feature vectorization (TfIDF, CountVectorizer, encoding) or embedding (word2vec, doc2vec, Bert, Elmo, sentence embeddings, etc.)
- Training a model with ML and DL algorithms.

Text classification in Spark NLP

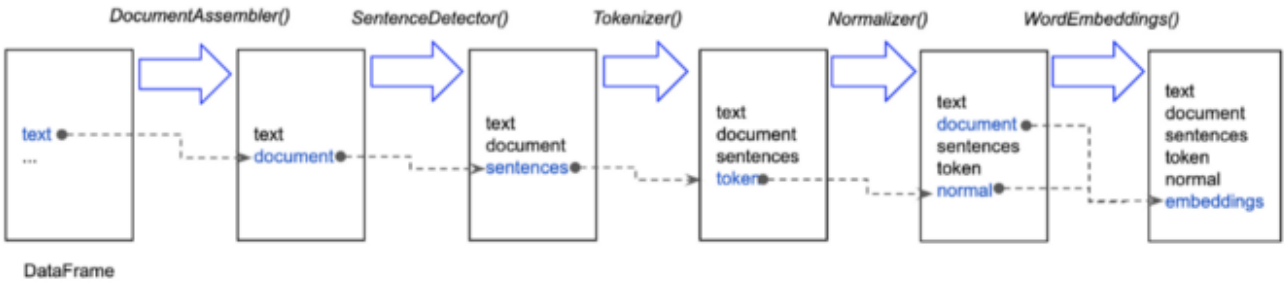
In this article, we will build a text classification model in Spark NLP using Universal Sentence Embeddings. Then we will compare this with other ML and DL approaches and text vectorization methods.

There are several text classification options in Spark NLP:

- Text preprocessing in Spark NLP and using ML algorithms from Spark ML
- Text preprocessing and word embeddings (Glove, Bert, Elmo) in Spark NLP and ML algorithms from Spark ML
- Text preprocessing and sentence embeddings (Universal Sentence Encoders) in Spark NLP and ML algorithms from Spark ML
- Text preprocessing and ClassifierDL (TensorFlow based DL architecture) in Spark NLP

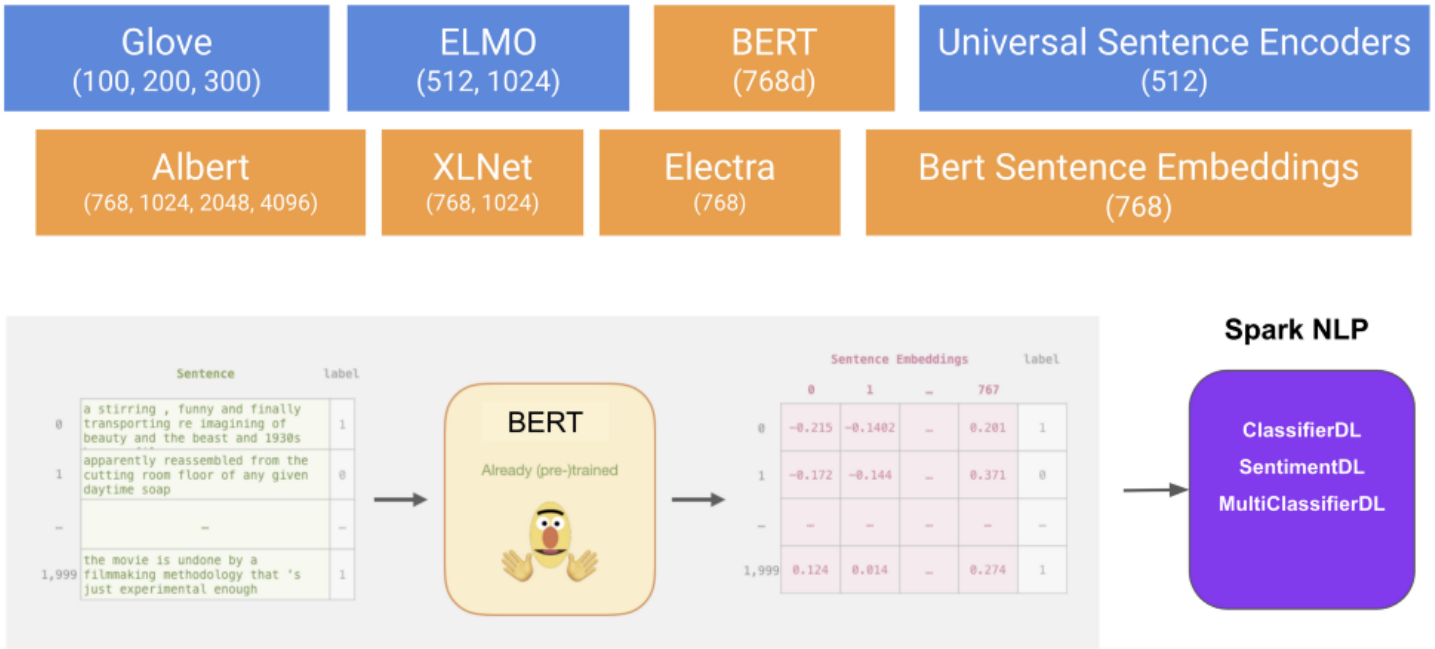
As discussed thoroughly in our seminal article about Spark NLP, all these text processing steps before `ClassifierDL` can be implemented in a pipeline that is specified as a sequence of stages, and each stage is either a Transformer or an Estimator. These stages are run in order, and the input DataFrame is transformed as it passes through each stage. That is, the data are passed through the fitted pipeline in order. Each stage's *transform()* method updates the dataset and passes it to the next stage. With the help of

Pipelines, we can ensure that training and test data go through identical feature processing steps.



Each annotator applied adds a new column to a data frame that is fed into the pipeline

Here are the available word/sentence embeddings and language models in Saprk NLP.



Spark NLP has 3 text classifiers that accept sentence embeddings

Universal Sentence Encoders (USE)

Before building any Deep Learning model in Natural Language Processing (NLP), text embedding plays a major role. The text embedding converts text (words or sentences) into a numerical vector.

> *Basically, the text embedding methods encode **words** and **sentences** in fixed-length dense vectors to drastically improve the processing of textual data. The idea is simple: Words that*

> *occur in the same contexts tend to have similar meanings.*

Techniques like Word2vec and Glove do that by converting a word to vector. Thus the corresponding vector of "cat" will be closer to "dog" than "eagle". But while embedding a sentence, along with words the context of the whole sentence needs to be captured in that vector. This is where the "Universal Sentence Encoder" comes into the picture.

The Universal Sentence Encoder encodes text into high dimensional vectors that can be used for text classification, semantic similarity, clustering, and other natural language tasks. The pre-trained Universal Sentence Encoder is publicly available in Tensorflow-hub . It comes with two variations i.e. one trained with **Transformer encoder** and the other trained with **Deep Averaging Network (DAN)**.

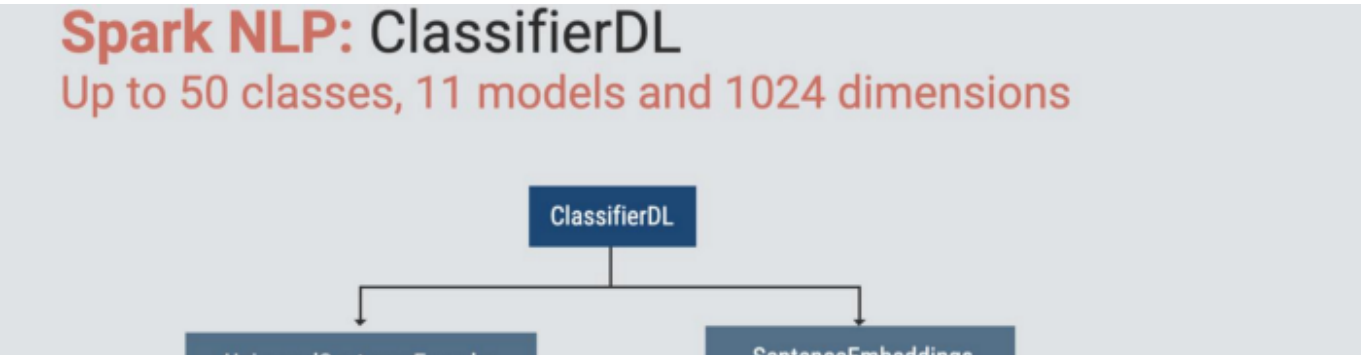
Spark NLP also use Tensorflow-hub version of USE that is wrapped in a way to get it run in the Spark environment. That is, you can just plug and play this embedding in Spark NLP and train a model in a distributed fashion.

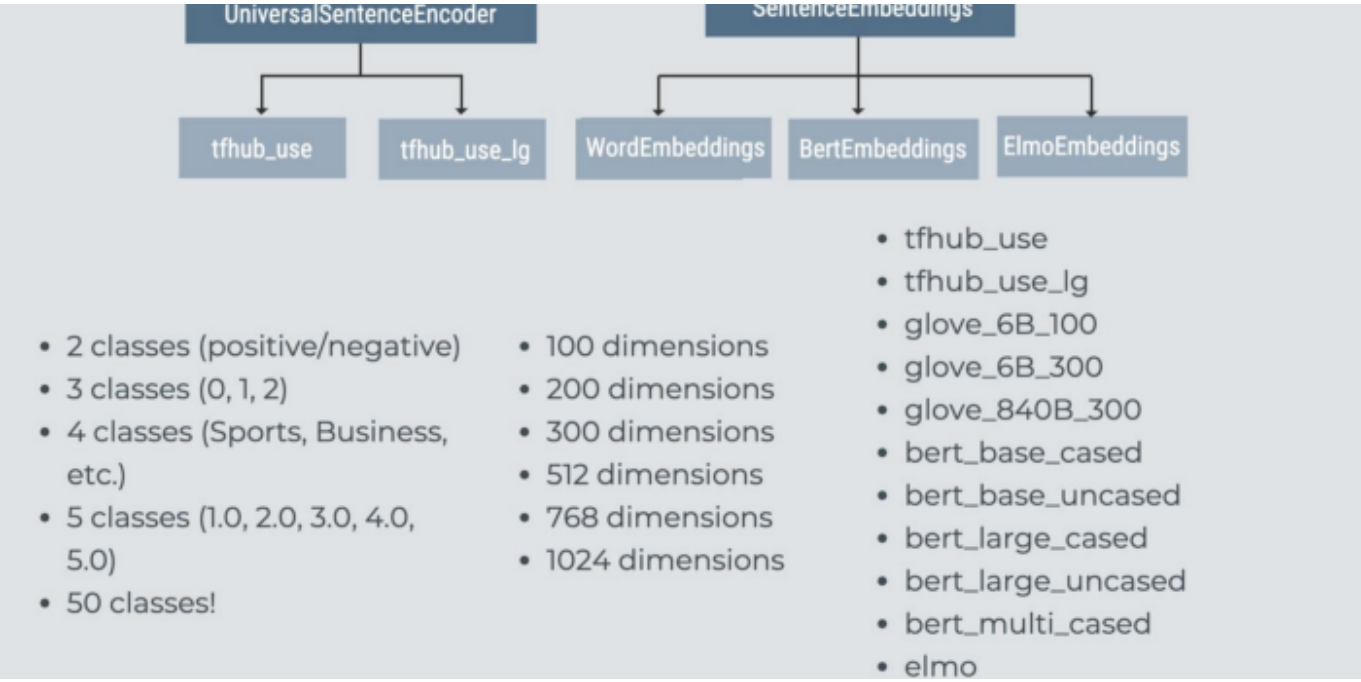
Instead of averaging the word embeddings of each word in a sentence to get a sentence embeddings, USE generates embeddings for the sentence with no further calculation.

Text Classification with ClassifierDL and USE in Spark NLP

In this article, we will use the AGNews dataset, one of the benchmark datasets in Text Classification tasks, to build a text classifier in Spark NLP using USE and **ClassifierDL** annotator, the latest classification module added to Spark NLP with version 2.4.4.

`ClassifierDL` is the very first multi-class text classifier in Spark NLP and it uses various text embeddings as an input for text classifications. The `ClassifierDL` annotator uses a deep learning model (DNNs) that is built inside TensorFlow and supports up to 50 classes.





That is, you can build a text classifier with `Bert`, `Elmo`, `Glove` and `Universal Sentence Encoders` in Spark NLP with this `ClassiferDL`.

Lets start coding !

Stating with loading necessary packages and starting a Spark session.

```
import sparknlp

spark = sparknlp.start()
# sparknlp.start(gpu=True) >> for training on GPU

from sparknlp.base import *
from sparknlp.annotator import *
from pyspark.ml import Pipeline
import pandas as pd

print("Spark NLP version", sparknlp.version())

print("Apache Spark version:", spark.version)

>> Spark NLP version 2.4.5
>> Apache Spark version: 2.4.4
```

We can then download `AGNews data set` from Github repo.

```
! wget https://raw.githubusercontent.com/JohnSnowLabs/spark-nlp-workshop/master/tutorials/Certification_Trainings/Public/data/news_category_train.csv

! wget https://raw.githubusercontent.com/JohnSnowLabs/spark-nlp-workshop/master/tutorials/Certification_Trainings/Public/data/news_category_test.csv

trainDataset = spark.read \
      .option("header", True) \
      .csv("news_category_train.csv")

trainDataset.show(10, truncate=50)

>>
+--------+--------------------------------------------------+
|category|                                       description|
+--------+--------------------------------------------------+
|Business| Short sellers, Wall Street's dwindling band of...|
|Business| Private investment firm Carlyle Group, which h...|
|Business| Soaring crude prices plus worries about the ec...|
|Business| Authorities have halted oil export flows from ...|
|Business| Tearaway world oil prices, toppling records an...|
|Business| Stocks ended slightly higher on Friday but sta...|
|Business| Assets of the nation's retail money market mut...|
|Business| Retail sales bounced back a bit in July, and n...|
|Business|" After earning a PH.D. in Sociology, Danny Baz...|
|Business| Short sellers, Wall Street's dwindling  band o...|
+--------+--------------------------------------------------+
only showing top 10 rows
```

`AGNews` dataset has 4 classes: `World`, `Sci/Tech`, `Sports`, `Business`

```
from pyspark.sql.functions import col

trainDataset.groupBy("category") \
    .count() \
    .orderBy(col("count").desc()) \
    .show()
>>
+--------+-----+
|category|count|
+--------+-----+
|   World|30000|
|Sci/Tech|30000|
|  Sports|30000|
|Business|30000|
+--------+-----+
```

```
testDataset = spark.read \
      .option("header", True) \
      .csv("news_category_test.csv")

testDataset.groupBy("category") \
    .count() \
    .orderBy(col("count").desc()) \
    .show()

>>
+--------+-----+
|category|count|
+--------+-----+
|Sci/Tech| 1900|
|  Sports| 1900|
|   World| 1900|
|Business| 1900|
+--------+-----+
```

Now we can feed this data frame to Spark NLP DocumentAssembler, which is the entry point to Spark NLP for any Spark datagram.

```
# actual content is inside description column

document = DocumentAssembler()\
    .setInputCol("description")\
    .setOutputCol("document")

# we can also use sentence detector here
# if we want to train on and get predictions for each sentence

# downloading pretrained embeddings
use = UniversalSentenceEncoder.pretrained()\
 .setInputCols(["document"])\
 .setOutputCol("sentence_embeddings")

# the classes/labels/categories are in category column

classsifierdl = ClassifierDLApproach()\
  .setInputCols(["sentence_embeddings"])\
  .setOutputCol("class")\
  .setLabelColumn("category")\
  .setMaxEpochs(5)\
  .setEnableOutputLogs(True)

use_clf_pipeline = Pipeline(
    stages = [
        document,
```

```
        use,
        classsifierdl
    ])
```

That's all. We take the dataset, feed into, and then get the sentence embeddings from `USE` and then train in `ClassifierDL`. With zero text preprocessing or cleaning!

Now let's start the training (fitting). We will train for 5 epochs using `.setMaxEpochs()` param in `ClassiferDL`. This will take around 10 min to finish in `Colab` environment.

```
use_pipelineModel = use_clf_pipeline.fit(trainDataset)
```

When you run this, Spark NLP will write the training logs to `annotator_logs` folder in your home directory. Here is how you can read the logs.

```
!cd ~/annotator_logs && ls -l
```

```
total 8
-rw-r--r-- 1 root root 533 Apr  7 17:14 ClassifierDLApproach_a0fddc9e970b.log
-rw-r--r-- 1 root root 976 Apr  7 16:59 ClassifierDLApproach_f222663dfb2c.log
```

```
!cat ~/annotator_logs/ClassifierDLApproach_a0fddc9e970b.log
```

```
Training started - total epochs: 5 - learning rate: 0.005 - batch size: 64 - training examples: 120000
Epoch 0/5 - 34.868680102%.2fs - loss: 1620.7466 - accuracy: 0.8803833 - batches: 1875
Epoch 1/5 - 35.627811455%.2fs - loss: 1604.4518 - accuracy: 0.8915333 - batches: 1875
Epoch 2/5 - 34.687788982%.2fs - loss: 1597.8773 - accuracy: 0.8966333 - batches: 1875
Epoch 3/5 - 34.629944711%.2fs - loss: 1593.4987 - accuracy: 0.900275 - batches: 1875
Epoch 4/5 - 34.714090256%.2fs - loss: 1590.3165 - accuracy: 0.90335834 - batches: 1875
```

As you can see, we achieved above **90% validation accuracy in less than 10 min** with no text preprocessing, which is usually the most time consuming and laborious step in any NLP modeling.

Now let's get the predictions on `test set`. We will use the test set that we downloaded above.

```
preds = clf_pipelineModel.transform(testDataset)

preds.select('category','description',"class.result").show(10, truncate=80)
```

```
+--------+--------------------------------------------------------------------------------+----------+
|category|                                                                     description|    result|
```

```
|category|                                                                                  description|    result|
+--------+-----------------------------------------------------------------------------------------+----------+
|Business|Unions representing workers at Turner   Newall say they are 'disappointed' af...|[Business]|
|Sci/Tech| TORONTO, Canada    A second team of rocketeers competing for the  #36;10 mil...|[Sci/Tech]|
|Sci/Tech| A company founded by a chemistry researcher at the University of Louisville ...|[Sci/Tech]|
|Sci/Tech| It's barely dawn when Mike Fitzpatrick starts his shift with a blur of color...|  [Sports]|
|Sci/Tech| Southern California's smog fighting agency went after emissions of the bovin...|   [World]|
|Sci/Tech|"The British Department for Education and Skills (DfES) recently launched a "...|[Sci/Tech]|
|Sci/Tech|"confessed author of the Netsky and Sasser viruses, is responsible for 70 per...|[Sci/Tech]|
|Sci/Tech|\\FOAF/LOAF  and bloom filters have a lot of interesting properties for socia...|[Sci/Tech]|
|Sci/Tech|"Wiltshire Police warns about ""phishing"" after its fraud squad chief was ta...|[Sci/Tech]|
|Sci/Tech|In its first two years, the UK's dedicated card fraud unit, has recovered 36,...|[Sci/Tech]|
+--------+-----------------------------------------------------------------------------------------+----------+
only showing top 10 rows
```

Here is how we can get the test metrics through `classification_report` in `sklearn` library.

```
from sklearn.metrics import classification_report, accuracy_score

df = pipelineModel.transform(testDataset).select('category','description',"class.result").toPandas()

df['result'] = df['result'].apply(lambda x: x[0])

print(classification_report(df.category, df.result))
print(accuracy_score(df.category, df.result))
```

```
              precision    recall  f1-score   support

    Business       0.85      0.84      0.85      8911
    Sci/Tech       0.85      0.87      0.86      8973
      Sports       0.95      0.98      0.97      9063
       World       0.92      0.88      0.90      9008

    accuracy                           0.89     35955
   macro avg       0.89      0.89      0.89     35955
weighted avg       0.89      0.89      0.89     35955

0.8930329578639966
```

We achieved **89.3% test set accuracy**! Looks nice!

Text Classification with Bert Sentence Embeddings in Spark NLP

(After 2.6.0 version, Spark NLP introduced `BertSentenceEmbeddings` annotator and more than 30 pretrained sentence embeddings models for Electra and Bert, in various sizes.)

	H=128	H=256	H=512	H=768
L=2	2/128 (BERT-Tiny)	2/256	2/512	2/768
L=4	4/128	4/256 (BERT-Mini)	4/512 (BERT-Small)	4/768
L=6	6/128	6/256	6/512	6/768
L=8	8/128	8/256	8/512 (BERT-Medium)	8/768

L=10	10/128	10/256	10/512	10/768
L=12	12/128	12/256	12/512	**12/768 (BERT-Base)**

Naming conventions in Spark NLP Bert models. L indicates which pooling layer is used while producing the embeddings and H mean the dimension of the embeddings returned. See the full list here.

Now let's use `BertSentenceEmbeddings` to train a model on the same dataset.

```
# actual content is inside description column
document = DocumentAssembler()\
    .setInputCol("description")\
    .setOutputCol("document")

bert_sent = BertSentenceEmbeddings.pretrained('sent_small_bert_L8_512')\
 .setInputCols(["document"])\
 .setOutputCol("sentence_embeddings")

# the classes/labels/categories are in category column
classsifierdl = ClassifierDLApproach()\
  .setInputCols(["sentence_embeddings"])\
  .setOutputCol("class")\
  .setLabelColumn("category")\
  .setMaxEpochs(20)\
  .setEnableOutputLogs(True)

bert_sent_clf_pipeline = Pipeline(
    stages = [
        document,
        bert_sent,
        classsifierdl
    ])
```

```
sent_small_bert_L8_512 download started this may take some time.
Approximate size to download 149.1 MB
[OK!]
```

Using a small Bert Sentence Embeddings instead of USE

We just load a small Bert sentence embeddings with `L8` and `512` dimension and use that instead of `USE`. As you can see it is almost 8 times smaller the size of `USE` with the power of Bert. And here what we get after the training.

```
              precision    recall  f1-score   support

    Business       0.84      0.86      0.85      1859
    Sci/Tech       0.89      0.84      0.86      2004
      Sports       0.97      0.95      0.96      1946
       World       0.87      0.92      0.89      1791

    accuracy                           0.89      7600
   macro avg       0.89      0.89      0.89      7600
weighted avg       0.89      0.89      0.89      7600
```

Almost the same metrics as what we get with `USE` . It's like AutoML on scale with a few lines of code !

Text Classification with text preprocessing in Spark NLP using Bert and Glove embeddings

As it is the case in any text classification problem, there are a bunch of useful text preprocessing techniques including lemmatization, stemming, spell checking and stopwords removal, and nearly all of the NLP libraries in Python have the tools to apply these techniques ***except spell checking***. *At the moment, the Spark NLP library is the only available NLP library out there to have a spell checking capability out of the box.*

Let's apply these steps in a Spark NLP pipeline and then train a text classifier with Glove word embeddings. We will at first apply several text preprocessing steps (normalize by just keeping the alphabetical letters, removing stopwords and then lemma)and then get word embeddings per token (lemma of a token) and then average the word embeddings in each sentence to get a sentence embeddings per row.

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

tokenizer = Tokenizer() \
  .setInputCols(["document"]) \
  .setOutputCol("token")

normalizer = Normalizer() \
    .setInputCols(["token"]) \
    .setOutputCol("normalized")

stopwords_cleaner = StopWordsCleaner()\
      .setInputCols("normalized")\
      .setOutputCol("cleanTokens")\
      .setCaseSensitive(False)

lemma = LemmatizerModel.pretrained('lemma_antbnc') \
    .setInputCols(["cleanTokens"]) \
    .setOutputCol("lemma")
```

```
word_embeddings = WordEmbeddingsModel().pretrained() \
 .setInputCols(["document",'lemma'])\
 .setOutputCol("embeddings")\
 .setCaseSensitive(False)

embeddingsSentence = SentenceEmbeddings() \
      .setInputCols(["document", "embeddings"]) \
      .setOutputCol("sentence_embeddings") \
      .setPoolingStrategy("AVERAGE")

classsifierdl = ClassifierDLApproach()\
  .setInputCols(["sentence_embeddings"])\
  .setOutputCol("class")\
  .setLabelColumn("target")\
  .setMaxEpochs(5)\
  .setEnableOutputLogs(True)

clf_pipeline = Pipeline(
    stages=[document_assembler,
            tokenizer,
            normalizer,
            stopwords_cleaner,
            lemma,
            word_embeddings,
            embeddingsSentence,
            classsifierdl])
```

Regarding all these text preprocessing tools in Spark NLP and more, you can find detailed instructions and code samples in this Colab notebook.

Then we can fit on the train set.

```
clf_pipelineModel = clf_pipeline.fit(trainDataset)
```

and get the test metrics.

```
df = clf_pipelineModel.transform(testDataset).select('category','description',"class.result").toPandas()

df['result'] = df['result'].apply(lambda x: x[0])
```

```
print(classification_report(df.category, df.result))

print(accuracy_score(df.category, df.result))
```

```
              precision    recall  f1-score   support

    Business       0.85      0.82      0.83      8911
    Sci/Tech       0.81      0.89      0.85      8973
      Sports       0.95      0.97      0.96      9063
       World       0.92      0.86      0.89      9008

    accuracy                           0.88     35955
   macro avg       0.88      0.88      0.88     35955
weighted avg       0.88      0.88      0.88     35955

0.8809066889167014
```

Now we have **88% test set accuracy**! Even after all these text cleaning steps, we couldn't beat `Universal Sentence Embeddings + ClassifierDL` :-) This is mainly due to the fact that `USE` is usually performs better on raw text compared to the cleaned version as it is already trained on raw sentences and when we apply text preprocessing we introduce some `noise` that was not seen while the `USE` is being trained and the sentence consistency was compromised while cleaning.

In order to train the same classifier with `BERT` , we can replace `glove_embeddings` stage with `bert_embeddings` stage in the same pipeline we built above. You can find more information about how we implement and leverage `Bert` in Spark NLP at this link.

```
word_embeddings = BertEmbeddings\
    .pretrained('bert_base_cased', 'en') \
    .setInputCols(["document",'lemma'])\
    .setOutputCol("embeddings")\

# we can also use Elmo embeddings instead.

word_embeddings = ElmoEmbeddings\
      .pretrained('elmo', 'en')\
      .setInputCols(["document",'lemma'])\
      .setOutputCol("embeddings")
```

Faster inference with LightPipeline

As discussed thoroughly in one of our previous articles, `LightPipelines` are Spark NLP specific Pipelines, equivalent to Spark ML Pipeline, but meant to deal with smaller

amounts of data. They're useful working with small datasets, debugging results, or when running either training or prediction from an API that serves one-off requests.

`Spark NLP LightPipelines` are Spark ML pipelines converted into a single machine but the multi-threaded task, becoming more than 10x times faster for smaller amounts of data (small is relative, but 50k sentences are roughly a good maximum). To use them, we simply plug in a trained (fitted) pipeline and then annotate a plain text. We don't even need to convert the input text to DataFrame in order to feed it into a pipeline that's accepting DataFrame as an input in the first place. This feature would be quite useful when it comes to getting a prediction for a few lines of text from a trained ML model.

LightPipelines are easy to create and also save you from dealing with Spark Datasets. They are also very fast and, while working only on the driver node, they execute parallel computation. Let's see how it applies to our case described above:

```
light_model = LightPipeline(clf_pipelineModel)

text="Euro 2020 and the Copa America have both been moved to the
summer of 2021 due to the coronavirus outbreak."

light_model.annotate(text)['class'][0]

>> "Sports"
```

You can also save this trained model to your disk and then use later in another Spark pipeline with `ClassifierDLModel.load()`.

```
clf_pipelineModel.stages
```

```
[DocumentAssembler_a4224cf73186,
 REGEX_TOKENIZER_18b626cc3e31,
 NORMALIZER_4a634222fa41,
 StopWordsCleaner_80e52b887898,
 LEMMATIZER_c62ad8f355f9,
 WORD_EMBEDDINGS_MODEL_48cffc8b9a76,
 SentenceEmbeddings_b7c9b012a4f3,
 ClassifierDLModel_e4e44f848b98]
```

```
clf_pipelineModel.stages[-1].write().overwrite()\
        .save('myClassifierDL_20200412')
```

Conclusion

In this article, we trained a multi-class text classification model in Spark NLP using popular word embeddings and Universal Sentence Encoders, and then achieved a decent model accuracy in less than 10 min train time. The entire code can be found at this Github repo (Colab compatible). We also prepared another notebook to cover nearly all the possible Text Classification combinations in Spark NLP and Spark ML (CV, TfIdf, Glove, Bert, Elmo, USE, LR, RF, ClassiferDL, DocClassifier).

We also started to give online Spark NLP trainings for both public and enterprise (healthcare) versions. Here is the link to all the public Colab notebooks that would take you from zero to hero in a few hours step by step.

John Snow Labs will be organizing virtual Spark NLP trainings and here is the link to the next training:

https://events.johnsnowlabs.com/online-training-spark-nlp

We hope that you already read the previous articles on our official Medium page, joined our slack channel, and started to play with Spark NLP. Here are the links for the other articles. Don't forget to follow our page and stay tuned!

Introduction to Spark NLP: Foundations and Basic Components (Part-I)

Introduction to: Spark NLP: Installation and Getting Started (Part-II)

Spark NLP 101 : Document Assembler

Spark NLP 101: LightPipeline

Named Entity Recognition (NER) with BERT in Spark NLP

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

spark = sparknlp.start()

trainDataset = spark.read.option("header", True) \
                    .csv("news_category_train.csv")

document = DocumentAssembler()\
    .setInputCol("description")\
    .setOutputCol("document")

use = UniversalSentenceEncoder.pretrained()\
 .setInputCols(["document"])\
 .setOutputCol("sentence_embeddings")

classsifierdl = ClassifierDLApproach()\
  .setInputCols(["sentence_embeddings"])\
  .setOutputCol("class")\
  .setLabelColumn("category")\
  .setMaxEpochs(5)\
  .setEnableOutputLogs(True)

use_clf_pipeline = Pipeline(
    stages = [
        document,
        use,
        classsifierdl
    ])

clf_model = use_clf_pipeline.fit(trainDataset)

light_model = LightPipeline(clf_model)

text="Euro 2020 and the Copa America have both been moved
to the summer of 2021 due to the coronavirus outbreak."

light_model.annotate(text)['class'][0]

>> "Sports"
```

The entire pipeline for text classification with USE in Spark NLP

Spark NLP Text Classification TensorFlow Bert