

NAME:RAHULDASS S

EXAM NO:422221104028

COLLEGE CODE:4222

COLLEGE NAME:SURYA GROUP OF INSTITUTIONS



Build a Spam Classifier

With deep learning and AI, handling spam content has gotten easier and easier. Over time (and with the aid of direct user feedback) our spam classifier will rarely produce erroneous results.

This is the first part of a multi-part series covering how to:

- Build an AI Model (this one)
- Integrate a NoSQL Database (inference result storing)
- Deploy an AI Model into Production

```
!pip install boto3
```

```
# !pip install pandas tensorflow
```

```
import boto3
```

```
import os
```

```
import pathlib
```

```
import pandas as pd
```

```
import pickle

from tensorflow.keras.layers import Dense, Input

from tensorflow.keras.layers import Conv1D, MaxPooling1D, Embedding, LSTM, SpatialDropout1D

from tensorflow.keras.models import Model, Sequential

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
EXPORT_DIR = pathlib.Path('/datasets/exports/')
GUIDES_DIR = pathlib.Path("/guides/spam-classifier/")
DATASET_CSV_PATH = EXPORT_DIR / 'spam-dataset.csv'
TRAINING_DATA_PATH = EXPORT_DIR / 'spam-training-data.pkl'
PART_TWO_GUIDE_PATH = GUIDES_DIR / "2 - Convert Dataset into Vectors.ipynb"
```

Prepare Dataset

Creating a dataset rarely happens next to where you run the training. The below cells are a method for us to extract the needed data to perform training against.

```
!mkdir -p "$EXPORT_DIR"

!mkdir -p "$GUIDES_DIR"

!curl "https://raw.githubusercontent.com/codingforentrepreneurs/AI-as-an-API/main/datasets/exports/spam-dataset.csv" -o "$DATASET_CSV_PATH"

!curl "https://raw.githubusercontent.com/codingforentrepreneurs/AI-as-an-API/main/guides/spam-classifier/2%20-%20Convert%20Dataset%20into%20Vectors.ipynb" -o "$PART_TWO_GUIDE_PATH"
```

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload  Total  Spent  Left  Speed
100 729k 100 729k    0    0 1175k    0 --:--:-- --:--:-- --:--:-- 1173k
```

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload  Total  Spent  Left  Speed
100 15408 100 15408    0    0 40547    0 --:--:-- --:--:-- --:--:-- 40547
```

Let's review our extracted dataset which combines two different spam datasets as outlined in [this notebook](#).

```
df = pd.read_csv(DATASET_CSV_PATH)

df.head()
```

```
%run "$PART_TWO_GUIDE_PATH"
```

```
BASE_DIR is /
```

```
Random Index 2234
```

```
Found 9538 unique tokens.
```

```
data = {}
```

```
with open(TRAINING_DATA_PATH, 'rb') as f:
```

```
    data = pickle.load(f)
```

While the above code uses `pickle` to load in data, this data is actually exported via `pickle` when we execute the `%run` only a few steps ago. Since `pickle` can be unsafe to use from third-party downloaded data, we actually generate (again using `%run`) this pickle data and therefore is safe to use

Transform Extracted Dataset

```
CopyCopyX_test = data['X_test']
```

```
X_train = data['X_train']
```

```
y_test = data['y_test']
```

```
y_train = data['y_train']
```

```
labels_legend_inverted = data['labels_legend_inverted']
```

```
legend = data['legend']
```

```
max_sequence = data['max_sequence']
```

```
max_words = data['max_words']
```

```
tokenizer = data['tokenizer'] x_test = data['X_test']
```

Create our LSTM Model

```
embed_dim = 128
```

```
lstm_out = 196
```

```
model = Sequential()
```

```

model.add(Embedding(MAX_NUM_WORDS, embed_dim, input_length=X_train.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(lstm_out, dropout=0.3, recurrent_dropout=0.3))
model.add(Dense(2, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])
print(model.summary())

```

WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
embedding (Embedding)	(None, 280, 128)	35840

spatial_dropout1d (SpatialDr	(None, 280, 128)	0

lstm (LSTM)	(None, 196)	254800

dense (Dense)	(None, 2)	394
=====		

Total params: 291,034

Trainable params: 291,034

Non-trainable params: 0

None

batch_size = 32

epochs = 5

model.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size=batch_size, verbose=1, epochs=epochs)

Epoch 1/5

163/163 [=====] - 278s 2s/step - loss: 0.2675 - accuracy: 0.8958 - val_loss: 0.1621 - val_accuracy: 0.9446

Epoch 2/5

163/163 [=====] - 271s 2s/step - loss: 0.1256 - accuracy: 0.9577 - val_loss: 0.1075 - val_accuracy: 0.9664

Epoch 3/5

163/163 [=====] - 270s 2s/step - loss: 0.1087 - accuracy: 0.9619 - val_loss: 0.1113 - val_accuracy: 0.9610

Epoch 4/5

163/163 [=====] - 268s 2s/step - loss: 0.0961 - accuracy: 0.9650 - val_loss: 0.0910 - val_accuracy: 0.9703

Epoch 5/5

163/163 [=====] - 261s 2s/step - loss: 0.0904 - accuracy: 0.9681 - val_loss: 0.0969 - val_accuracy: 0.9653

<keras.callbacks.History at 0x7ff640316f50>

MODEL_EXPORT_PATH = EXPORT_DIR / 'spam-model.h5'

model.save(str(MODEL_EXPORT_PATH))

Predict new data

```
import numpy as np
```

```
def predict(text_str, max_words=280, max_sequence = 280, tokenizer=None):
```

```
    if not tokenizer:
```

```
        return None
```

```
    sequences = tokenizer.texts_to_sequences([text_str])
```

```
    x_input = pad_sequences(sequences, maxlen=max_sequence)
```

```
    y_output = model.predict(x_input)
```

```
    top_y_index = np.argmax(y_output)
```

```
    preds = y_output[top_y_index]
```

```
    labeled_preds = [{f'{labels_legend_inverted[str(i)]}': x} for i, x in enumerate(preds)]
```

```
    return labeled_preds
```

```
predict("Hello world", max_words=max_words, max_sequence=max_sequence,  
tokenizer=tokenizer)
```

```
{{'ham': 0.96744573}, {'spam': 0.032554302}}
```

Exporting Tokenizer & Metadata

```
import json

metadata = {
    "labels_legend_inverted": labels_legend_inverted,
    "legend": legend,
    "max_sequence": max_sequence,
    "max_words": max_words,
}

METADATA_EXPORT_PATH = EXPORT_DIR / 'spam-classifer-metadata.json'
METADATA_EXPORT_PATH.write_text(json.dumps(metadata, indent=4))

187

tokenizer_as_json = tokenizer.to_json()

TOKENIZER_EXPORT_PATH = EXPORT_DIR / 'spam-classifer-tokenizer.json'
TOKENIZER_EXPORT_PATH.write_text(tokenizer_as_json)

827612
```

We can

load `tokenizer_as_json` with `tensorflow.keras.preprocessing.text.tokenizer_from_json`.

Upload Model, Tokenizer, & Metadata to Object Storage

Object Storage options include:

- AWS S3
- Linode Object Storage
- DigitalOcean Spaces

All three of these options can use `boto3`.

AWS S3 Config

ACCESS_KEY = "<your_aws_iam_key_id>"

SECRET_KEY = "<your_aws_iam_secret_key>"

You should not have to set this

ENDPOINT = None

Your s3-bucket region

REGION = 'us-west-1'

BUCKET_NAME = '<your_s3_bucket_name>'

Linode Object Storage Config

ACCESS_KEY = "<your_linode_object_storage_access_key>"

SECRET_KEY = "<your_linode_object_storage_secret_key>"

Object Storage Endpoint URL

ENDPOINT = "https://cfe3.us-east-1.linodeobjects.com"

Object Storage Endpoint Region (also in your endpoint url)

REGION = 'us-east-1'

Set this to a valid slug (without a "/")

BUCKET_NAME = 'datasets'

DigitalOcean Spaces Config

ACCESS_KEY = "<your_do_spaces_access_key>"

SECRET_KEY = "<your_do_spaces_secret_key>"

Space Endpoint URL

ENDPOINT = "https://ai-cfe-1.nyc3.digitaloceanspaces.com"

Space Region (also in your endpoint url)

```
REGION = 'nyc3'
```

```
# Set this to a valid slug (without a "/" )
```

```
BUCKET_NAME = 'datasets'
```

Perform Upload with Boto3

```
os.environ["AWS_ACCESS_KEY_ID"] = ACCESS_KEY
```

```
os.environ["AWS_SECRET_ACCESS_KEY"] = SECRET_KEY
```

```
# Upload paths
```

```
MODEL_KEY_NAME = f"exports/spam-sms/{MODEL_EXPORT_PATH.name}"
```

```
TOKENIZER_KEY_NAME = f"exports/spam-sms/{TOKENIZER_EXPORT_PATH.name}"
```

```
METADATA_KEY_NAME = f"exports/spam-sms/{METADATA_EXPORT_PATH.name}"
```

```
session = boto3.session.Session()
```

```
client = session.client('s3', region_name=REGION, endpoint_url=ENDPOINT)
```

```
client.upload_file(str(MODEL_EXPORT_PATH), BUCKET_NAME, MODEL_KEY_NAME)
```

```
client.upload_file(str(TOKENIZER_EXPORT_PATH), BUCKET_NAME, TOKENIZER_KEY_NAME)
```

```
client.upload_file(str(METADATA_EXPORT_PATH), BUCKET_NAME, METADATA_KEY_NAME)
```

```
client.download_file(BUCKET_NAME, MODEL_KEY_NAME, pathlib.Path(MODEL_KEY_NAME).name)
```

```
client.download_file(BUCKET_NAME, TOKENIZER_KEY_NAME,  
pathlib.Path(TOKENIZER_KEY_NAME).name)
```

```
client.download_file(BUCKET_NAME, METADATA_KEY_NAME,  
pathlib.Path(METADATA_KEY_NAME).name)
```

conclusion :

By accurately identifying and filtering spam, individuals and organizations can focus on important emails and mitigate potential risks associated with malicious content.