



**SURYA GROUP OF INSTITUTIONS**

**NAAN MUDHALVAN**

**IBM-ARTIFICIAL INTELLIGENCE**

ABINASH R

422221104002

AI-Powered Spam Classifier

TEAM : 01



# **Build a Spam Classifier with Machine Learning Algorithm**

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

**Prerequisites:**

- Prepare your dataset using this notebook .
- Convert your dataset into trainable vectors in this notebook (Either way, this notebook will run this step for us).

**Running this notebook:**

- Recommended: Use Colab as it offers free GPUs for training models.

Launch this notebook here

- Fork the AI as an API repo and run

guides/spam-classifier/Spam\_Classifier\_with\_Keras.ipynb  
whenever you'd like.

```
!pip install boto3
```

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

### **Packages:**

```
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/Al-as-an-API/main/datasets/exports/spam-dataset.csv" -o "$DATASET_CSV_PATH"
```

```
!curl
```

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

**OutPut:**

% 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

Data Set:

```
df = pd.read_csv(DATASET_CSV_PATH)
```

```
df.head()
```

	label		text	source
0	ham	Go until jurong point, crazy.. Available only ...		uci-spam-sms
1	ham	Ok lar... Joking wif u oni...		uci-spam-sms
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...		uci-spam-sms
3	ham	U dun say so early hor... U c already then say...		uci-spam-sms
4	ham	Nah I don't think he goes to usf, he lives aro...		uci-spam-sms

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

BASE\_DIR is /

Random Index 2234

Found 9538 unique tokens.

Extract prepared training dataset results:

```
data = {}
```

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

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

### **Transform Extracted Dataset:**

```
X_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']
```

### **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
<hr/>		
spatial_dropout1d (SpatialDr	(None, 280, 128)	0
<hr/>		
lstm (LSTM)	(None, 196)	254800
<hr/>		
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-classifier-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-classifier-tokenizer.json'
```

```
TOKENIZER_EXPORT_PATH.write_text(tokenizer_as_json)
```

827612

## **Upload Model, Tokenizer, & Metadata to Object Storage:**

### **Object Storage options include:**

- AWS S3
- Linode Object Storage
- DigitalOcean Spaces

```
# 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)
```

```
,
```