MACQUARIE
University

Faculty of Science & Engineering

COMP8325 Applications of Artificial Intelligence for Cyber Security

Workshop Week 11 and 12

Learning outcomes for this session

The following content aims at exploring machine learning techniques for detection of malicious URLs and spam (unsolicited and unwanted) messages.

NOTE

- The responses to **Question 3** and **Question 4** are a part of your tutorial task for **Week 11** and **Week 12**, respectively.
- Download data and scripts from iLearn.

TASK 1. Spam-filtering

Definition. Spam Filter is a machine learning model that is used to detect unsolicited and unwanted messages and avert those messages from getting to a users's inbox.

In the following, we will detect SMS spam.

(1) **Spam-filtering Techniques.** The most challenging task in machine learning is to apply algorithm because we can not apply any random algorithm to any random data.

To apply a machine learning algorithm on some data, we need to explore the data. To this end, we do visualise the data to get obtain some understanding of the data. As the dataset, for this task, has two classes, we have two columns one is text or messages and other one is class spam or ham. There are some classification techniques like `kNN`, `Naive Bayes`, `logistic regression`, but after thinking intuitively we will use Näive Bayes Algorithm because it uses Bayes theorem to determine the probability that a give message is spam, given words in this message.

If S is the event of a given message being spam and w is a word in the message, we will classify it as spam with probability:

$$P(S|w) = \frac{P(w|S) * P(S)}{P(w|S) * P(S) + P(w|S') * P(S')} \tag{1}$$

(2) **Preprocessing and vectorisation.** First of all we need to preprocess our text data and convert it into vectors with the help of featurisation such as: **BOW(Bag of Words)**,**TF-IDF**, **Word2Vec**, **Avg TF-IDF Word2vec**. Näive Bayes uses probability of data so the data values should be positive

but in the case of **Word2vec** and **Avg TFIDF Word2vec** vectors may be positive or negative so we can not use this featurisation.

(3) **Feature Extraction.** After preprocessing we will split our data into train and test then convert into vectors with the help of BOW and TF-IDF.

(4) We will use Scikit library of python to implement Naive Bayes.

(5) **Laplace Smoothing.** It is used to smooth categorical data. Lets take an observation $x = (x_1, x_1, x_1, \ldots, x_d)$ from a multinomial distribution with N trials then,

$$clf = \frac{(x+\alpha)}{(N+\alpha * d)}. \tag{2}$$

So we need to figure out our best alpha value such that our area under ROC curve maximize. Then we will fit our train data for optimal alpha and predict for test data and summarise it into confusion matrix using heatmaps.

(6) **Question 1.** What is underfitting and overfitting?

(7) **Question 2.** How underfitting and overfitting related to high bias and high variance?

TASK 2. Detecting Malicious URLs

Definition. A malicious website[1] is a site that attempts to install malware (a general term for anything that will disrupt computer operation, gather your personal information or, in a worst-case scenario, gain total access to your machine) onto your device. This usually requires some action on your part, however, in the case of a drive-by download, the website will attempt to install software on your computer without asking for permission first.

In the following, we will create a model that can detect malicious websites. Website URL is used as a feature and 1D Convolutional Neural Network (CNN) is used as an algorithm for detection malicious websites. Model will be validated by holdout method.

(1) Install `tldextract`:

```
$ sudo pip install tldextract
```

(2) Import libraries:

[1] https://us.norton.com/internetsecurity-malware-what-are-malicious-websites.html

```
import numpy as np
import pandas as pd
import re
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
import gc
import random
import os
import pickle
import tensorflow as tf
from tensorflow.python.util import deprecation
from urllib.parse import urlparse
import tldextract
```

(3) Import some more libraries:

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras import models, layers, backend, metrics
from tensorflow.keras.callbacks import EarlyStopping
from keras.utils.vis_utils import plot_model
from PIL import Image
from sklearn.metrics import confusion_matrix, classification_report
```

(4) Set environment for further processing:

```
# set random seed
os.environ['PYTHONHASHSEED'] = '0'
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
np.random.seed(0)
random.seed(0)
tf.set_random_seed(0)
```

```
# other setup
%config InlineBackend.figure_format = 'retina'
pd.set_option('max_colwidth', 500)
deprecation._PRINT_DEPRECATION_WARNINGS = False
```

(5) Load data

```
# load data
data = pd.read_csv('./malicious_url_dataset.csv')
# shuffle data
data = data.sample(frac=1, random_state=0)
print(f'Data size: {data.shape}')
data.head()
```

(6) Split data for validation and testing.

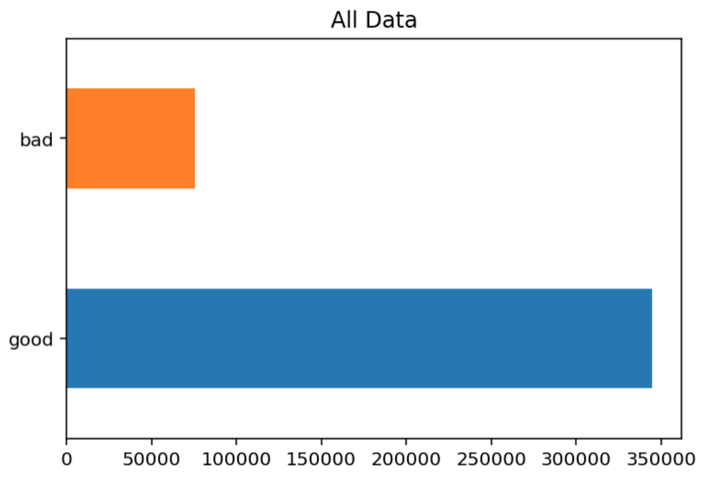
```
val_size = 0.2
train_data, val_data = train_test_split(data, test_size=val_size, s
tratify=data['label'], random_state=0)
print(f'Train shape: {train_data.shape}, Validation shape: {val_
data.shape}')
```

```
Train shape: (336371, 2), Validation shape: (84093, 2)
```

In this notebook the validation method used is the holdout method. The holdout method is a method that separates training and test data by 80% and 20%.

(7) Analyze data for different features:

```
data.label.value_counts().plot.barh()
plt.title('All Data')
plt.show()
```



(8) The next step we need to do tokenization on the URL so that it can be used as input to the CNN model

```python
tokenizer = Tokenizer(filters='', char_level=True, lower=False, oov
_token=1)
# fit only on training data
tokenizer.fit_on_texts(train_data['url'])
n_char = len(tokenizer.word_index.keys())
print(f'N Char: {n_char}')
```

```
N Char: 177
```

```python
train_seq = tokenizer.texts_to_sequences(train_data['url'])
val_seq = tokenizer.texts_to_sequences(val_data['url'])
print('Before tokenization: ')
print(train_data.iloc[0]['url'])
print('\nAfter tokenization: ')
print(train_seq[0])
```

```
Before tokenization:
melangemagazine.com/wp-content/plugins/telekom/vodafone_onlinerechn
ung

After tokenization:
[12, 2, 14, 4, 11, 20, 2, 12, 4, 20, 4, 43, 5, 11, 2, 13, 8, 3, 12,
6, 26, 17, 15, 8, 3, 11, 7, 2, 11, 7, 6, 17, 14, 19, 20, 5, 11, 9,
6, 7, 2, 14, 2, 28, 3, 12, 6, 30, 3, 16, 4, 25, 3, 11, 2, 29, 3, 11
, 14, 5, 11, 2, 10, 2, 8, 18, 11, 19, 11, 20]
```

(9) Training:

```python
train_seq = pad_sequences(train_seq, padding='post', maxlen=sequenc
e_length)
val_seq = pad_sequences(val_seq, padding='post', maxlen=sequence_le
ngth)
print('After padding: ')
print(train_seq[0])
```

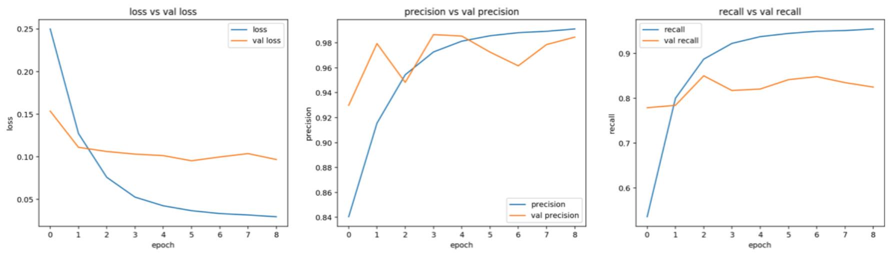
```
After padding:
[12  2 14  4 11 20  2 12  4 20  4 43  5 11  2 13  8  3 12  6 26 17
15  8
  3 11  7  2 11  7  6 17 14 19 20  5 11  9  6  7  2 14  2 28  3 12
6 30
  3 16  4 25  3 11  2 29  3 11 14  5 11  2 10  2  8 18 11 19 11 20
0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
```

```
print(f"Unique subdomain in Train data: {unique_value['subdomain
']}")
print(f"Unique domain in Train data: {unique_value['domain']}")
print(f"Unique domain suffix in Train data: {unique_value['domain
_suffix']}")
```

```
Unique subdomain in Train data: 23344
Unique domain in Train data: 98290
Unique domain suffix in Train data: 674
```

(10) Testing and Results:

```
plt.figure(figsize=(20, 5))
for index, key in enumerate(['loss', 'precision', 'recall']):
    plt.subplot(1, 3, index+1)
    plt.plot(history.history[key], label=key)
    plt.plot(history.history[f'val_{key}'], label=f'val {key}')
    plt.legend()
    plt.title(f'{key} vs val {key}')
    plt.ylabel(f'{key}')
    plt.xlabel('epoch')
```



(11) Validation

```
val_pred = model.predict(val_x)
val_pred = np.where(val_pred[:, 0] >= 0.5, 1, 0)
print(f'Validation Data:\n{val_data.label.value_counts()}')
print(f'\n\nConfusion Matrix:\n{confusion_matrix(val_y, val_pred)}
')
print(f'\n\nClassification Report:\n{classification_report(val_y,
val_pred)}')
```

```
Validation Data:
0    68964
1    15129
Name: label, dtype: int64


Confusion Matrix:
[[68796   168]
 [ 2762 12367]]
```

(12) **Question 3.** What does lower precision value indicates?

(13) **Question 4.** In contrast, what does higher value of precision indicates?

Tutorial Task. The response to **Question 3** is your tutorial task for **Week 11** while response to **Question 4** is a part of your tutorial task for **Week 12**. Submit your answers as online text (only numbers) via https://ilearn.mq.edu.au by Monday May 30th, 2023, 11:55pm. No extension will be granted.

——— End of Workshop Week 11 and 12 ———