Name -> Hariprakash S P

Batch -> Data Science & Machine Learning with Python

Project Name -> Anti Phishing Prediction

Group -> Own

Source code

```
#importing Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#import data
data0 = pd.read_csv('/content/4.phishing.csv')
data0.head()
data0.shape
data0.columns
data0.info()
data0.hist(bins = 50,figsize = (15,15))
plt.show()
#heatmap
plt.figure(figsize=(15,13))
sns.heatmap(data0.corr())
plt.show()


data0.describe()
data = data0.drop(['Domain'], axis = 1).copy()
data = data.sample(frac=1).reset_index(drop=True)
data.head()
# Sepratating & assigning features and target columns to X & y
y = data['Label']
X = data.drop('Label',axis=1)
X.shape, y.shape
```

```
# Splitting the dataset into train and test sets: 80-20 split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size =
0.2, random_state = 12)
X_train.shape, X_test.shape

#importing Libraries
import keras
from keras.layers import Input, Dense
from keras import regularizers
from keras.models import Model
from sklearn import metrics
input_dim = X_train.shape[1]
encoding_dim = input_dim

input_layer = Input(shape=(input_dim, ))
encoder = Dense(encoding_dim, activation="relu",

activity_regularizer=regularizers.l1(10e-4))(input_layer)
encoder = Dense(int(encoding_dim), activation="relu")(encoder)

encoder = Dense(int(encoding_dim-2), activation="relu")(encoder)
code = Dense(int(encoding_dim-4), activation='relu')(encoder)
decoder = Dense(int(encoding_dim-2), activation='relu')(code)

decoder = Dense(int(encoding_dim), activation='relu')(encoder)
decoder = Dense(input_dim, activation='relu')(decoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.summary()
#compiling the model
autoencoder.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])
```

```
#Training the model
history = autoencoder.fit(X_train, X_train, epochs=10, batch_size=64,
shuffle=True, validation_split=0.2)
acc_train_auto = autoencoder.evaluate(X_train, X_train)[1]
acc_test_auto = autoencoder.evaluate(X_test, X_test)[1]

print('\nAutoencoder: Accuracy on training Data: {:.3f}'
.format(acc_train_auto))
print('Autoencoder: Accuracy on test Data: {:.3f}'
.format(acc_test_auto))
```

Implementation

Anti_phishing_Detection

colab.research.google.com/drive/1cn57y-jiYIMYAwSwwgWxliUgSqaoJziy#scrollTo=9IOacNmkC-Jc

Anti_phishing_Detection.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk

Name -> Hariprakash S P

Batch -> Data Science & Machine Learning with Python

Project Name -> Anti Phishing Prediction

Group -> Own

[26]
```
#importing Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
#import data
data0 = pd.read_csv('/content/4.phishing.csv')
data0.head()
```

	Domain	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	Tiny_URL	Prefix/Suffix	DNS_
0	eevee.tv	0	0	0	4	0	0	0	0	
1	appleid.apple.com-sa.pm	0	0	0	1	0	0	0	1	
2	grandcup.xyz	0	0	0	0	0	0	0	0	
3	villa-azzurro.com	0	0	0	1	0	0	0	1	
4	mygpstrip.net	0	0	0	2	0	0	0	0	

[28]
```
data0.shape
```

```
(5000, 18)
```

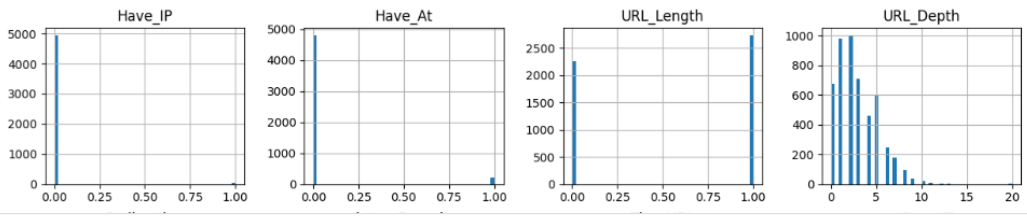
[29]
```
data0.columns
```

```
Index(['Domain', 'Have_IP', 'Have_At', 'URL_Length', 'URL_Depth',
       'Redirection', 'https_Domain', 'Tiny_URL', 'Prefix/Suffix',
       'DNS_Record', 'Web_Traffic', 'Domain_Age', 'Domain_End', 'iFrame',
       'Mouse_Over', 'Right_Click', 'Web_Forwards', 'Label'],
      dtype='object')
```

```
data0.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Domain         5000 non-null   object
 1   Have_IP        5000 non-null   int64
 2   Have_At        5000 non-null   int64
 3   URL_Length     5000 non-null   int64
 4   URL_Depth      5000 non-null   int64
 5   Redirection    5000 non-null   int64
 6   https_Domain   5000 non-null   int64
 7   Tiny_URL       5000 non-null   int64
 8   Prefix/Suffix  5000 non-null   int64
 9   DNS_Record     5000 non-null   int64
 10  Web_Traffic    5000 non-null   int64
 11  Domain_Age     5000 non-null   int64
 12  Domain_End     5000 non-null   int64
 13  iFrame         5000 non-null   int64
 14  Mouse_Over     5000 non-null   int64
 15  Right_Click    5000 non-null   int64
 16  Web_Forwards   5000 non-null   int64
 17  Label          5000 non-null   int64
dtypes: int64(17), object(1)
memory usage: 703.2+ KB
```

[31]
```
data0.hist(bins = 50,figsize = (15,15))
plt.show()
```



1s completed at 9:13 PM

Anti_phishing_Detection

colab.research.google.com/drive/1cn57y-jiYIMYAwSwwgWxliUgSqaoJziy#scrollTo=9IOacNmkC-Jc

Anti_phishing_Detection.ipynb

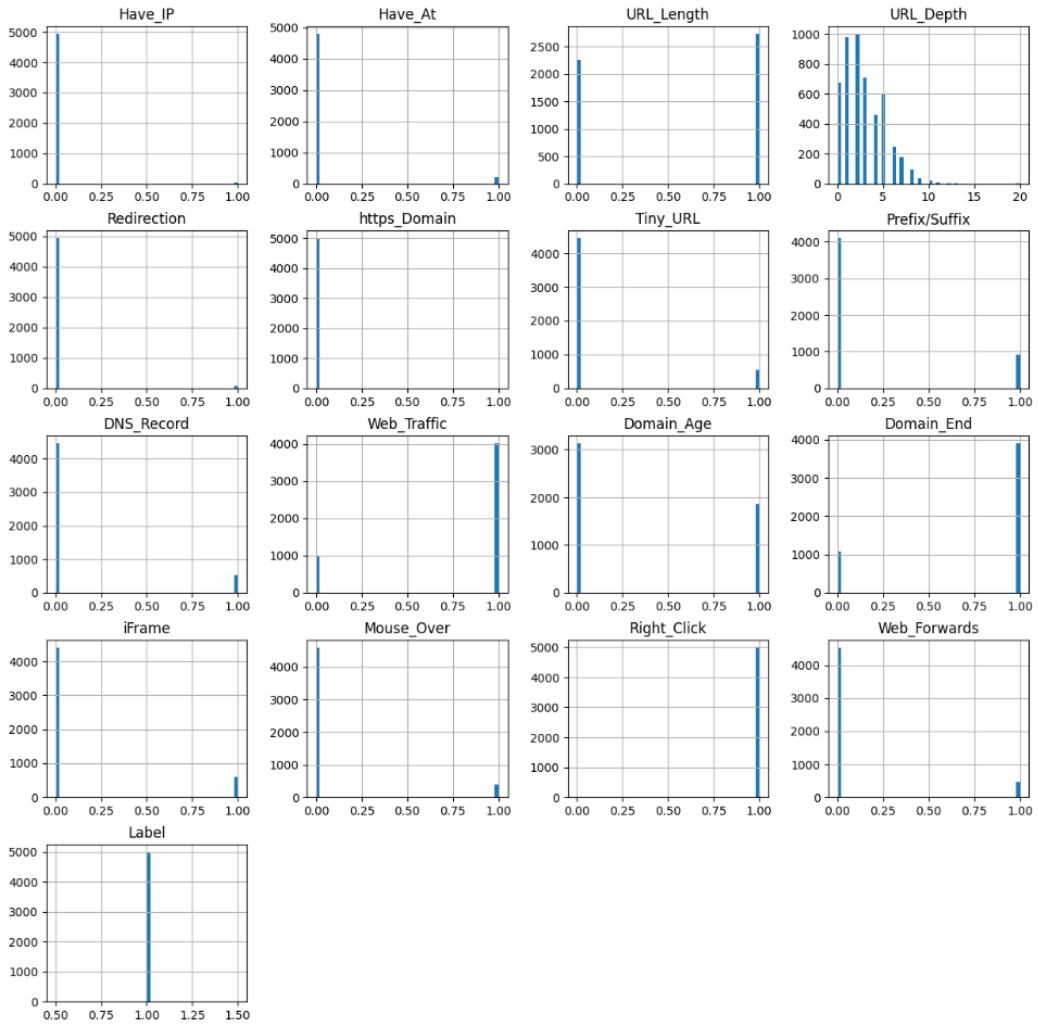
File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk

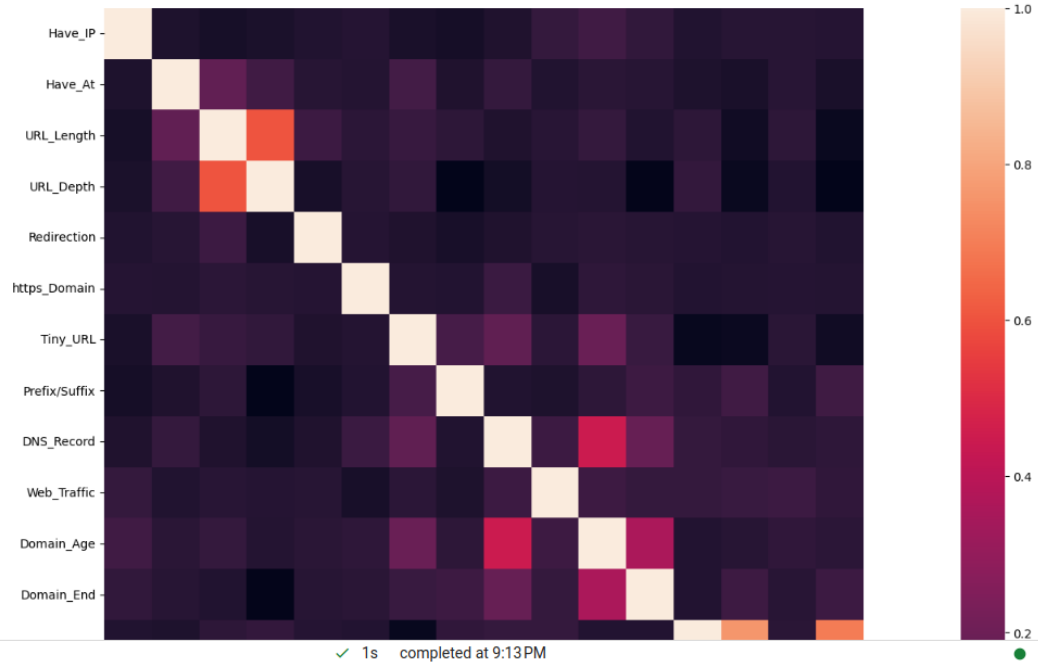
```
data0.hist(bins = 50,figsize = (15,15))
plt.show()
```



+ Code + Text

Add code cell
Ctrl+M B

```
[32] plt.figure(figsize=(15,13))
     sns.heatmap(data0.corr())
     plt.show()
```



1s completed at 9:13 PM

Anti_phishing_Detection

colab.research.google.com/drive/1cn57y-jiYIMYAwSwwgWxliUgSqaoJziy#scrollTo=9IOacNmkC-Jc

Anti_phishing_Detection.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk

[32]

[33]
```
data0.describe()
```

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	Tiny_URL	Prefix/Suffix	DNS_Recor
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.0000(
mean	0.011000	0.040200	0.546800	2.817200	0.013800	0.000400	0.111200	0.181200	0.1056(
std	0.104313	0.196448	0.497855	2.225213	0.116672	0.019998	0.314411	0.385222	0.3073!
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000(
25%	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.0000(
50%	0.000000	0.000000	1.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.0000(
75%	0.000000	0.000000	1.000000	4.000000	0.000000	0.000000	0.000000	0.000000	0.0000(
max	1.000000	1.000000	1.000000	20.000000	1.000000	1.000000	1.000000	1.000000	1.0000(

[34]
```
data = data0.drop(['Domain'], axis = 1).copy()
```

[35]
```
data.isnull().sum()
```

```
Have_IP          0
Have_At          0
URL_Length       0
URL_Depth        0
Redirection      0
https_Domain     0
Tiny_URL         0
Prefix/Suffix    0
DNS_Record       0
Web_Traffic      0
Domain_Age       0
Domain_End       0
iFrame           0
Mouse_Over       0
Right_Click      0
Web_Forwards     0
Label            0
dtype: int64
```

[36]
```
data = data.sample(frac=1).reset_index(drop=True)
data.head()
```

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	Tiny_URL	Prefix/Suffix	DNS_Record	Web_Traf
0	0	0	0	3	0	0	0	1	0	
1	0	1	1	5	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	
3	0	0	1	2	0	0	0	0	0	
4	0	0	0	2	0	0	0	0	0	

[37]
```
# Sepratating & assigning features and target columns to X & y
y = data['Label']
X = data.drop('Label',axis=1)
X.shape, y.shape
```

```
((5000, 16), (5000,))
```

[38]
```
# Splitting the dataset into train and test sets: 80-20 split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size = 0.2, random_state = 12)
X_train.shape, X_test.shape
```

```
((4000, 16), (1000, 16))
```

Autoencoder Neural Network

An auto encoder is a neural network that has the same number of input neurons as it does outputs. The hidden layers of the neural network will have fewer neurons than the input/output neurons. Because there are fewer neurons, the auto encoder must learn to encode the input to

1s completed at 9:13 PM

Anti_phishing_Detection

colab.research.google.com/drive/1cn57y-jiYIMYAwSwwgWxliUgSqaoJziy#scrollTo=9IOacNmkC-Jc

Anti_phishing_Detection.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk

Autoencoder Neural Network

An auto encoder is a neural network that has the same number of input neurons as it does outputs. The hidden layers of the neural network will have fewer neurons than the input/output neurons. Because there are fewer neurons, the auto-encoder must learn to encode the input to the fewer hidden neurons. The predictors (x) and output (y) are exactly the same in an auto encoder.

[43]
```
#importing Libraries
import keras
from keras.layers import Input, Dense
from keras import regularizers
from keras.models import Model
from sklearn import metrics
```

```
input_dim = X_train.shape[1]
encoding_dim = input_dim

input_layer = Input(shape=(input_dim, ))
encoder = Dense(encoding_dim, activation="relu",
                activity_regularizer=regularizers.l1(10e-4))(input_layer)
encoder = Dense(int(encoding_dim), activation="relu")(encoder)

encoder = Dense(int(encoding_dim-2), activation="relu")(encoder)
code = Dense(int(encoding_dim-4), activation='relu')(encoder)
decoder = Dense(int(encoding_dim-2), activation='relu')(code)

decoder = Dense(int(encoding_dim), activation='relu')(encoder)
decoder = Dense(input_dim, activation='relu')(decoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.summary()
```

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 16)]              0

 dense_28 (Dense)            (None, 16)                272

 dense_29 (Dense)            (None, 16)                272

 dense_30 (Dense)            (None, 14)                238

 dense_33 (Dense)            (None, 16)                240

 dense_34 (Dense)            (None, 16)                272

=================================================================
Total params: 1,294
Trainable params: 1,294
Non-trainable params: 0
_________________________________________________________________
```

[47]
```
#compiling the model
autoencoder.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

#Training the model
history = autoencoder.fit(X_train, X_train, epochs=10, batch_size=64, shuffle=True, validation_split=0.2)
```

```
Epoch 1/10
50/50 [==============================] - 1s 7ms/step - loss: 3.9442 - accuracy: 0.0284 - val_loss: 3.3880 - val_ac
Epoch 2/10
50/50 [==============================] - 0s 3ms/step - loss: 3.2061 - accuracy: 0.0256 - val_loss: 3.2945 - val_ac
Epoch 3/10
50/50 [==============================] - 0s 3ms/step - loss: 3.1260 - accuracy: 0.0081 - val_loss: 3.2345 - val_ac
Epoch 4/10
50/50 [==============================] - 0s 3ms/step - loss: 3.0863 - accuracy: 0.0044 - val_loss: 3.2204 - val_ac
Epoch 5/10
50/50 [==============================] - 0s 3ms/step - loss: 3.0689 - accuracy: 0.0034 - val_loss: 3.2019 - val_ac
Epoch 6/10
50/50 [==============================] - 0s 3ms/step - loss: 3.0545 - accuracy: 0.0034 - val_loss: 3.1894 - val_ac
Epoch 7/10
50/50 [==============================] - 0s 3ms/step - loss: 3.0391 - accuracy: 0.0034 - val_loss: 3.1730 - val_ac
Epoch 8/10
50/50 [==============================] - 0s 3ms/step - loss: 3.0298 - accuracy: 0.0034 - val_loss: 3.1684 - val_ac
Epoch 9/10
50/50 [==============================] - 0s 3ms/step - loss: 3.0220 - accuracy: 0.0034 - val_loss: 3.1594 - val_ac
Epoch 10/10
50/50 [==============================] - 0s 3ms/step - loss: 3.0144 - accuracy: 0.0034 - val_loss: 3.1479 - val_ac
```

[48]
```
acc_train_auto = autoencoder.evaluate(X_train, X_train)[1]
acc_test_auto = autoencoder.evaluate(X_test, X_test)[1]

print('\nAutoencoder: Accuracy on training Data: {:.3f}' .format(acc_train_auto))
print('Autoencoder: Accuracy on test Data: {:.3f}' .format(acc_test_auto))
```

```
125/125 [==============================] - 0s 2ms/step - loss: 3.0382 - accuracy: 0.0037
32/32 [==============================] - 0s 3ms/step - loss: 2.9828 - accuracy: 0.0000e+00

Autoencoder: Accuracy on training Data: 0.004
Autoencoder: Accuracy on test Data: 0.000
```

1s completed at 9:13 PM