I have imported some libraries.

In [1]:

```
import gensim
import collections
import numpy as np
import random
import pandas as pd
from sklearn import tree,calibration,ensemble,linear_model, neural_network,metrics
import tensorflow as tf
import sklearn
import tensorflow_hub as hub
import tensorflow_text
```

To train the model, we'll need to associate a tag/number with each document of the training corpus. In our case, the tag is simply the zero-based line number.

In [2]:

```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

def read_corpus(fname, tokens_only=False):
    with open(fname, encoding="iso-8859-1") as f:
        for i, line in enumerate(f):
            tokens = gensim.utils.simple_preprocess(line)
            if tokens_only:
                yield tokens
            else:
                # For training data, add tags
                yield gensim.models.doc2vec.TaggedDocument(tokens, [i])


# Loading the Train, Test and Validation Data
train_corpus = list(read_corpus("./train.txt"))
test_corpus = list(read_corpus("./test.txt",tokens_only=True))
validation_corpus=list(read_corpus("./validation.txt",tokens_only=True))
```

Now, we have instantiated a Doc2Vec model with a vector size with 50 dimensions and iterating over the training corpus 40 times. We set the minimum word count to 2 in order to discard words with very few occurrences.

In [3]:

```
model = gensim.models.doc2vec.Doc2Vec(vector_size=50, min_count=2, epochs=40)
```

Building a vocbulary for training corpus:

In [4]:

```
model.build_vocab(train_corpus)
```

In [5]:

```
model.train(train_corpus, total_examples=model.corpus_count, epochs=model.epochs)
```

First we have inferred new vectors for each document of the training corpus, Then, we did same task for the test and validation set also.

In [6]:
```
Train = model.infer_vector(train_corpus[0].words)
for doc_id in range(len(train_corpus)-1):
    X1 = model.infer_vector(train_corpus[doc_id].words)
    Train = np.vstack((X1,Train))
Train.shape
```

Out[6]:
```
(963, 50)
```

In [7]:
```
Valid = model.infer_vector(validation_corpus[0])
for doc_id in range(len(validation_corpus)-1):
    X1 = model.infer_vector(validation_corpus[doc_id])
    Valid = np.vstack((X1,Valid))
Valid.shape
```

Out[7]:
```
(118, 50)
```

In [8]:
```
Test = model.infer_vector(test_corpus[0])
for doc_id in range(len(test_corpus)-1):
    X1 = model.infer_vector(test_corpus[doc_id])
    Test = np.vstack((X1,Test))
Test.shape
```

Out[8]:
```
(122, 50)
```

Extracting the labels for train, test and validation set

In [9]:
```
train = pd.read_csv('trainlabels.txt', header = None)
train_labels=train[0].values
train_labels.shape
```

Out[9]:
```
(963,)
```

In [10]:
```
test = pd.read_csv('testlabels.txt', header = None)
test_labels=test[0].values
test_labels.shape
```

Out[10]:
```
(122,)
```

In [11]:
```
valid = pd.read_csv('validationlabels.txt', header = None)
valid_labels=valid[0].values
valid_labels.shape
```

Out[11]:
```
(118,)
```

Rndom Forest Classifer

I have applied Random Forest Classifier and calibrated classifier to get the optimal accuracy.

In [12]:

```
clf1 = sklearn.ensemble.RandomForestClassifier(max_depth=5).fit(Train,train_labels)
```

In [13]:
```
clf_calib = sklearn.calibration.CalibratedClassifierCV(base_estimator=clf1, cv='prefit'
```

In [14]:
```
clf_calib.score(Test,test_labels)
```

Out[14]:
```
0.680327868852459
```

In [15]:
```
clf1.score(Test,test_labels)
```

Out[15]:
```
0.6475409836065574
```

After using the Classifier and Calibrated Classifier I got the Accuracy respectively

Using the precision, recall and f1-score to evaluate the performance of Random Forest Classifier 64.75% and 68.03%

In [16]:
```
Y1_pred = clf_calib.predict(Test)
[Pre,Rec,F1,Oc] = metrics.precision_recall_fscore_support(test_labels, Y1_pred,average=
print("Precision =", Pre, "Recall=", Rec, "F1Score=", F1 )
```

```
Precision = 0.6932669789227166 Recall= 0.680327868852459 F1Score= 0.6789294780775352
```

Logistic Regression

By following the Instruction I have used the Logistic Regression Classifier and Calibrated Classifier to get the Optimal Accuracy.

In [67]:
```
clf2 = sklearn.linear_model.LogisticRegression(penalty='l2').fit(Train,train_labels)
```

In [68]:
```
clf2_tuned = sklearn.calibration.CalibratedClassifierCV(base_estimator=clf2, cv='prefit
```

In [69]:
```
clf2_tuned.score(Test,test_labels)
```

Out[69]:
```
0.6147540983606558
```

In [70]:
```
clf2.score(Test,test_labels)
```

Out[70]:
```
0.6065573770491803
```

After using the Classifier and Calibrated Classifier I got the Accuracy respectively 60.66% and 61.48%.

Using the precision, recall and f1-score to evaluate the performance of Logistic Regression Classifier

In [21]:
```
y_pred = clf2_tuned.predict(Test)
[Pre,Rec,F1,Oc] = metrics.precision_recall_fscore_support(test_labels, y_pred,average='
print("Precision =", Pre, "Recall=", Rec, "F1Score=", F1 )
```

```
Precision = 0.6137763848708883 Recall= 0.6147540983606558 F1Score= 0.6138929512527441
```

Grid Search for Logistic Regression on Training Set for finding Best Hyperparameters

In [107…
```
# grid search logistic regression model on the sonar dataset
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
@ignore_warnings(category=ConvergenceWarning)
def my_function():
    # Code that triggers the warning
    model = LogisticRegression()
    # define evaluation
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    # define search space
    space = dict()
    space['solver'] = ['newton-cg', 'lbfgs', 'liblinear']
    space['penalty'] = ['none', 'l1', 'l2', 'elasticnet']
    space['C'] = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100]
    # define search
    search = GridSearchCV(model, space, scoring='accuracy', n_jobs=-1, cv=cv)
    # execute search
    result = search.fit(Train,train_labels)
    # summarize result
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)
```

```
Best Score: 0.710946449026346
Best Hyperparameters: {'C': 0.00200201735306445, 'penalty': 'l2', 'solver': 'liblinear'}
```

Grid Search for Logistic Regression to maximize recall

The hyperparameters I have tuned are: Penalty: l1 or l2 which species the norm used in the penalization. C: Inverse of regularization strength- smaller values of C specify stronger regularization.

In [114…
```
#Grid Search
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score
from sklearn.model_selection import GridSearchCV
clf = LogisticRegression()
grid_values = {'penalty': ['l1', 'l2'],'C':[0.001,.009,0.01,.09,1,5,10,25]}
grid_clf_acc = GridSearchCV(clf, param_grid = grid_values,scoring = 'recall')
grid_clf_acc.fit(Train,train_labels)

#Predict values based on new parameters
y_pred_acc = grid_clf_acc.predict(Test)

# New Model Evaluation metrics
print('Accuracy Score : ' + str(accuracy_score(test_labels,y_pred_acc)))
print('Precision Score : ' + str(precision_score(test_labels,y_pred_acc)))
print('Recall Score : ' + str(recall_score(test_labels,y_pred_acc)))
print('F1 Score : ' + str(f1_score(test_labels,y_pred_acc)))
```

```python
#Logistic Regression (Grid Search) Confusion matrix
confusion_matrix(test_labels,y_pred_acc)

print('Confusion Matrix : \n' + str(confusion_matrix(test_labels,y_pred_acc)))
```

```
Accuracy Score : 0.6147540983606558
Precision Score : 0.5641025641025641
Recall Score : 0.7719298245614035
F1 Score : 0.6518518518518518
Confusion Matrix :
[[31 34]
 [13 44]]
/Users/mahmed3/opt/anaconda3/lib/python3.9/site-packages/sklearn/model_selection/_valida
tion.py:372: FitFailedWarning:
40 fits failed out of a total of 80.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='ra
ise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
40 fits failed with the following error:
Traceback (most recent call last):
  File "/Users/mahmed3/opt/anaconda3/lib/python3.9/site-packages/sklearn/model_selectio
n/_validation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/Users/mahmed3/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_l
ogistic.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/Users/mahmed3/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_l
ogistic.py", line 447, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/Users/mahmed3/opt/anaconda3/lib/python3.9/site-packages/sklearn/model_selection/_searc
h.py:969: UserWarning: One or more of the test scores are non-finite: [  nan 0.834   nan
0.728   nan 0.726   nan 0.712   nan 0.708   nan 0.708
   nan 0.708   nan 0.708]
  warnings.warn(
```

Neural Net

In [123…

```python
Net = sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(5,), activation='tanh',m
Net.fit(Train,train_labels)
```

Out[123…

```
MLPClassifier(activation='tanh', hidden_layer_sizes=(5,), max_iter=2000)
```

In [124…

```python
Net_calib = sklearn.calibration.CalibratedClassifierCV(base_estimator=Net, cv='prefit')
```

In [125…

```python
Net_calib.score(Test,test_labels)
```

Out[125…

```
0.680327868852459
```

In [126…

```
Net.score(Test, test_labels)
```

Out[126…

```
0.6639344262295082
```

After using the Classifier and Calibrated Classifier I got the Accuracy respectively 66.39% and 68.03%

Using the precision, recall and f1-score to evaluate the performance of Neural Net Classifier

In [127…

```
Y_pred = Net_calib.predict(Test)
[Pre,Rec,F1,Oc] = metrics.precision_recall_fscore_support(test_labels, Y_pred,average='
print("Precision =", Pre, "Recall=", Rec, "F1Score=", F1 )
```

```
Precision = 0.6807209228377614 Recall= 0.680327868852459 F1Score= 0.6804787084155215
```

By observing the result, I can say that, the train dataset is not big enough according to the test and validation set. I got best accuracy for Randoforest and neural net also. I have tried to find the best parameter for Logistic Regression and maximize the recall. I have tried my best to tune the above mentioned Classifier parameter to get the higher accuracy.

I have also tried to implement the BERT(Bidirectional Encoder Representation from Transformers) on th dataset to get best result. The Implentation is given below.

Here I have applied BERT (Bidirectional Encoder Representation from Transformers) to classify fake reviews from real reviews by using the Given Dataset.

In [1]:
```
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_text
```

Loading the dataset

In [77]:
```
import pandas as pd
import numpy as np

train_corpus = pd.read_csv('train.txt',delimiter='	', header=None)
train_label = pd.read_csv('trainlabels.txt', header=None)

test_corpus=pd.read_csv('test.txt',delimiter='	', header=None)
test_label=pd.read_csv('testlabels.txt', header=None)


validation_corpus=pd.read_csv('validation.txt',delimiter='	', header=None)
validation_label=pd.read_csv('validationlabels.txt', header=None)
```

Now lets import BERT model and get embeding vectors for few sample statements.

In [28]:
```
bert_preprocess = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_preproce

bert_encoder = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_
```

The below function return us the encodings for the sentences. These encodings are return by pretrined BERT modelwhich I have downloaded fron tfhub website.

In [9]:
```
def get_sentence_embeding(sentences):
    preprocessed_text = bert_preprocess(sentences)
    return bert_encoder(preprocessed_text)['pooled_output']

get_sentence_embeding([
    "500$ discount. hurry up",
    "Bhavin, are you up for a volleybal game tomorrow?"]
)
```

Out[9]:
```
<tf.Tensor: shape=(2, 768), dtype=float32, numpy=
array([[-0.84351707, -0.5132726 , -0.888457  , ..., -0.74748856,
        -0.75314736,  0.91964495],
       [-0.87208354, -0.50543964, -0.9444667 , ..., -0.858475  ,
        -0.7174535 ,  0.8808297 ]], dtype=float32)>
```

Get embeding vectors for few sample words. Compare them using cosine similarity. Here cosine similarity is a measure how similar to vectors are. If two vectors are pointing in the same direction then that means the cosine similarity will be close to 1. if we have vectors where the angle is less the they are more similar and if we have vectors the angle is more then they are less similar.

In [10]:
```
e = get_sentence_embeding([
    "banana",
    "grapes",
    "mango",
    "jeff bezos",
    "elon musk",
    "bill gates"
]
)
```

We have six embeddings. Each embeddings have 768 vectors. Here above first three term similar (fruits) in terms of cosine similarity. Then, the last three terms also similar (name of persons) in terms of cosine similarity.

In [11]:
```
e
```

Out[11]:
```
<tf.Tensor: shape=(6, 768), dtype=float32, numpy=
array([[-0.7606918 , -0.14219381,  0.4960459 , ...,  0.42165315,
        -0.532214  ,  0.80312175],
       [-0.8602322 , -0.21242929,  0.4915691 , ...,  0.39798048,
        -0.6050629 ,  0.84471637],
       [-0.7128861 , -0.15463905,  0.38401678, ...,  0.35278738,
        -0.5099134 ,  0.73474085],
       [-0.8253347 , -0.35550576, -0.5906969 , ..., -0.01613727,
        -0.61417574,  0.87230295],
       [-0.7504135 , -0.2681262 , -0.26689687, ...,  0.02839373,
        -0.5938099 ,  0.7974989 ],
       [-0.7854437 , -0.29949665,  0.41027418, ...,  0.5222541 ,
        -0.4957355 ,  0.8150751 ]], dtype=float32)>
```

In [12]:
```
from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity([e[0]],[e[1]])
```

Out[12]:
```
array([[0.9911089]], dtype=float32)
```

Values near to 1 means they are similar. 0 means they are very different. Above I can use comparing "banana" vs "grapes" I get 0.99 similarity as they both are fruits.

In [13]:
```
cosine_similarity([e[0]],[e[3]])
```

Out[13]:
```
array([[0.84703845]], dtype=float32)
```

Comparing banana with jeff bezos you still get 0.84 but it is not as close as 0.99 that we got with grapes.

In [14]:
```
cosine_similarity([e[3]],[e[4]])
```

Out[14]:
```
array([[0.9872036]], dtype=float32)
```

Jeff bezos and Elon musk are more similar then Jeff bezos and banana as indicated above

We need to understand how the embeddings are calculated. They are calculated based on the training they did on wekipidia and google books where based on the context.

Here, I am using the preprocessing and encoding this two functions into BERT layer. Here I am creating model. There are two process to crate tensorflow model. One is sequential and one is functional. Here I used sequential model. First I will create a input layer and I have passed it to BERT preprocess. And that I supplied it to BERT encoder that I have downloaded. And we get output as a result. Now, I will create Neural Networks Layer. I created dropout layers because Dropout helps witn overfitting. 0.1% of Neuron just Drop.

Sigmoid activation function, sigmoid(x) = 1 / (1 + exp(-x)). Applies the sigmoid activation function. For small values (<-5), sigmoid returns a value close to zero, and for large values (>5) the result of the function gets close to 1.

In [50]:

```
# Bert layers
text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
preprocessed_text = bert_preprocess(text_input)
outputs = bert_encoder(preprocessed_text)

# Neural network layers
l = tf.keras.layers.Dropout(0.1, name="dropout")(outputs['pooled_output'])
l = tf.keras.layers.Dense(1, activation='sigmoid', name="output")(l)

# Use inputs and outputs to construct a final model
model = tf.keras.Model(inputs=[text_input], outputs = [l])
```

In [51]:

```
model.summary()
```

```
Model: "model_5"
________________________________________________________________________________________
__________
 Layer (type)                   Output Shape          Param #     Connected to
========================================================================================
==========
 text (InputLayer)              [(None,)]             0           []

 keras_layer_2 (KerasLayer)     {'input_word_ids':    0           ['text[0][0]']
                                (None, 128),
                                 'input_mask': (Non
                                e, 128),
                                 'input_type_ids':
                                (None, 128)}

 keras_layer_3 (KerasLayer)     {'sequence_output':   109482241   ['keras_layer_2[11]
[0]',
                                 (None, 128, 768),                 'keras_layer_2[11]
[1]',
                                 'pooled_output': (                'keras_layer_2[11]
[2]']
                                None, 768),
                                 'default': (None,
                                768),
                                 'encoder_outputs':
                                 [(None, 128, 768),
                                 (None, 128, 768),
                                 (None, 128, 768),
```

```
                                                   (None, 128, 768),
                                                   (None, 128, 768),
                                                   (None, 128, 768),
                                                   (None, 128, 768),
                                                   (None, 128, 768),
                                                   (None, 128, 768),
                                                   (None, 128, 768),
                                                   (None, 128, 768),
                                                   (None, 128, 768)]}

 dropout (Dropout)              (None, 768)          0           ['keras_layer_3[11][1
3]']

 output (Dense)                 (None, 1)            769         ['dropout[0][0]']

==========================================================================================
==========
Total params: 109,483,010
Trainable params: 769
Non-trainable params: 109,482,241
__________________________________________________________________________________________
__________
```

Here, we can see that trainable parameter is 769 because 1 is dense layer and dropout layer 768. And others are non trainable parameter nad these are coming from BERT. We don't worry about retraining them. The actual training is driven by loss function. Adam is a replacement optimization algorithm for stochastic gradient descent for training deep learning models. The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters.0

Train the model

In [52]:

```
METRICS = [
      tf.keras.metrics.BinaryAccuracy(name='accuracy'),
      tf.keras.metrics.Precision(name='precision'),
      tf.keras.metrics.Recall(name='recall')
]

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=METRICS)
```

In [74]:

```
model.fit(train_corpus,train_label, epochs=10)
```

```
Epoch 1/10
31/31 [==============================] - 135s 4s/step - loss: 0.7168 - accuracy: 0.5151
- precision: 0.5277 - recall: 0.6280
Epoch 2/10
31/31 [==============================] - 126s 4s/step - loss: 0.6912 - accuracy: 0.5441
- precision: 0.5592 - recall: 0.5760
Epoch 3/10
31/31 [==============================] - 127s 4s/step - loss: 0.6702 - accuracy: 0.5774
- precision: 0.5730 - recall: 0.7300
```

```
Epoch 4/10
31/31 [==============================] - 125s 4s/step - loss: 0.6607 - accuracy: 0.6106
- precision: 0.6173 - recall: 0.6580
Epoch 5/10
31/31 [==============================] - 127s 4s/step - loss: 0.6444 - accuracy: 0.6345
- precision: 0.6336 - recall: 0.7020
Epoch 6/10
31/31 [==============================] - 127s 4s/step - loss: 0.6249 - accuracy: 0.6739
- precision: 0.6748 - recall: 0.7180
Epoch 7/10
31/31 [==============================] - 127s 4s/step - loss: 0.6174 - accuracy: 0.6791
- precision: 0.6884 - recall: 0.6980
Epoch 8/10
31/31 [==============================] - 140s 5s/step - loss: 0.6221 - accuracy: 0.6625
- precision: 0.6476 - recall: 0.7680
Epoch 9/10
31/31 [==============================] - 133s 4s/step - loss: 0.6023 - accuracy: 0.6916
- precision: 0.6926 - recall: 0.7300
Epoch 10/10
31/31 [==============================] - 126s 4s/step - loss: 0.6050 - accuracy: 0.6656
- precision: 0.6712 - recall: 0.6980
```

Out[74]:

```
<keras.callbacks.History at 0x7f9704423cd0>
```

If we are training on imbalance dataset we should not relay on accuracy. When we do model evaluation. When we do model.predict. We get y predicted and it is two dimensional but flatten make it one dimensional.

On the Test Dataset

In [78]:

```
model.evaluate(test_corpus, test_label)
```

```
4/4 [==============================] - 17s 4s/step - loss: 0.5443 - accuracy: 0.7541 - p
recision: 0.7547 - recall: 0.7018
```

Out[78]:

```
[0.5443463325500488,
 0.7540983557701111,
 0.7547169923782349,
 0.7017543911933899]
```

In [79]:

```
y_predicted = model.predict(test_corpus)
y_predicted = y_predicted.flatten()
```

Since y_predicted is bunch of sigmoid values so if the value is greater than 0.5, put value 1 otherwise put value 0.

In [80]:

```
import numpy as np

y_predicted = np.where(y_predicted > 0.5, 1, 0)
y_predicted
```

Out[80]:

```
array([0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0,
       0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1])
```

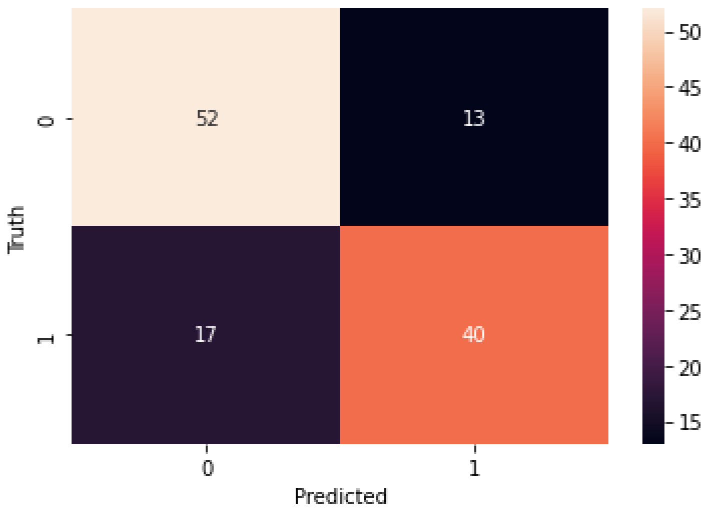
In [82]:
```
from sklearn.metrics import confusion_matrix, classification_report

cm = confusion_matrix(test_label, y_predicted)
cm
```

Out[82]:
```
array([[52, 13],
       [17, 40]])
```

In [83]:
```
from matplotlib import pyplot as plt
import seaborn as sn
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[83]:
```
Text(33.0, 0.5, 'Truth')
```



On the diagonal we have correct prediction. 52 times I have 0 as a truth and my Model predicted that to be 0. 40 times I have fake reviews and my model predicted to be fake reviews. On 17 Occasions I have fake review but model said it is not fake review. On 13 occassions I have real review but model told that it is not real.

In [84]:
```
print(classification_report(test_label, y_predicted))
```

```
              precision    recall  f1-score   support

           0       0.75      0.80      0.78        65
           1       0.75      0.70      0.73        57

    accuracy                           0.75       122
   macro avg       0.75      0.75      0.75       122
weighted avg       0.75      0.75      0.75       122
```

Then I have printed classification report.

Inference

In [85]:
```
reviews = [
    'Enter a chance to win $5000, hurry up, offer valid until march 31, 2021',
```

```
    'You are awarded a SiPix Digital Camera! call 09061221061 from landline. Delivery w
    'it to 80488. Your 500 free text messages are valid until 31 December 2005.',
    'Hey Sam, Are you coming for a cricket game tomorrow',
    "Why don't you wait 'til at least wednesday to see if you get your ."
]
model.predict(reviews)
```

Out[85]:

```
array([[0.40520084],
       [0.20399982],
       [0.2954064 ],
       [0.47395548],
       [0.34375012]], dtype=float32)
```

Validation Dataset

In [86]:

```
model.evaluate(validation_corpus, validation_label)
```

```
4/4 [==============================] - 16s 4s/step - loss: 0.5713 - accuracy: 0.7627 - p
recision: 0.7963 - recall: 0.7167
```

Out[86]:

```
[0.571272075176239, 0.7627118825912476, 0.7962962985038757, 0.7166666388511658]
```

In [87]:

```
Y_predicted = model.predict(validation_corpus)
Y_predicted = Y_predicted.flatten()
```

In [88]:

```
import numpy as np

Y_predicted = np.where(Y_predicted > 0.5, 1, 0)
Y_predicted
```

Out[88]:

```
array([1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0])
```

In [89]:

```
from sklearn.metrics import confusion_matrix, classification_report

CM = confusion_matrix(validation_label, Y_predicted)
CM
```

Out[89]:

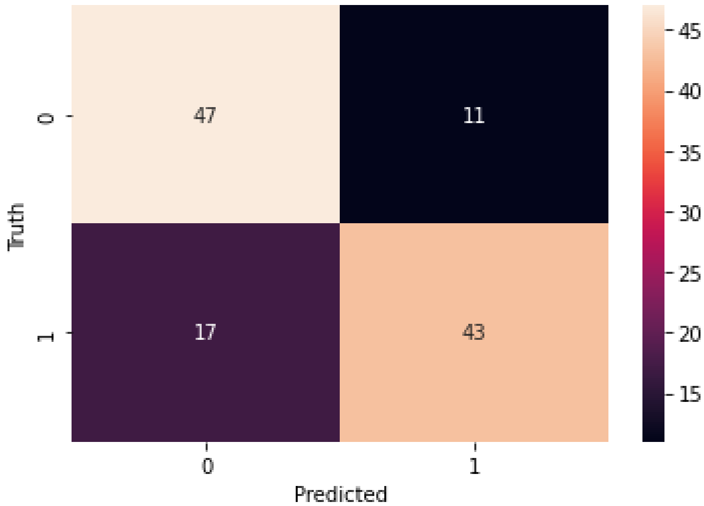
```
array([[47, 11],
       [17, 43]])
```

In [91]:

```
from matplotlib import pyplot as plt
import seaborn as sn
sn.heatmap(CM, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[91]:

```
Text(33.0, 0.5, 'Truth')
```



In [90]:
```
print(classification_report(validation_label, Y_predicted))
```

```
              precision    recall  f1-score   support

           0       0.73      0.81      0.77        58
           1       0.80      0.72      0.75        60

    accuracy                           0.76       118
   macro avg       0.77      0.76      0.76       118
weighted avg       0.77      0.76      0.76       118
```

In [ ]: