Homomorphic Encryption Basic Working

In [1]:
```
# Import Dependencies
import phe as paillier
```

In [2]:
```
# Create Public and Private Keys
key_length = 1024
pub_key, private_key = paillier.generate_paillier_keypair(n_length=key_length)
```

In [3]:
```
pub_key
```

Out[3]:
```
<PaillierPublicKey 1a714f2e30>
```

In [4]:
```
private_key
```

Out[4]:
```
<PaillierPrivateKey for <PaillierPublicKey 1a714f2e30>>
```

In [5]:
```
# Encrypt an operation using Public Key
a = 10
print("a: ",a)

encrypted_a = pub_key.encrypt(a)
print("Encrypted a: ",encrypted_a)

print("Encrypted a Public Key: ", encrypted_a.public_key)
```

```
a:  10
Encrypted a:  <phe.paillier.EncryptedNumber object at 0x000001B85635D190>
Encrypted a Public Key:  <PaillierPublicKey 1a714f2e30>
```

In [6]:
```
# Encrypt another variable
b = 5
print("b: ", b)

encrypted_b = pub_key.encrypt(b)
print("Encrypted b: ", encrypted_b)

print("Encrypted b Public Key: ",encrypted_b.public_key)
```

```
b:  5
Encrypted b:  <phe.paillier.EncryptedNumber object at 0x000001B85527D130>
Encrypted b Public Key:  <PaillierPublicKey 1a714f2e30>
```

In [7]:
```
# Do an operation on Encrypted Variables
c = a + b
print("c: ", c)
```

```
c:  15
```

In [8]:
```
d = a * b
print("d: ",d)
```

```
d:  50
```

In [9]:
```
e = a - b

encrypted_e = pub_key.encrypt(e)
print("Encrypted e: ", encrypted_e)
```

```
Encrypted e:  <phe.paillier.EncryptedNumber object at 0x000001B855375C70>
```

In [10]:
```
# Decrypt the Encrypted Data
decrypted_e = private_key.decrypt(encrypted_e)
```

In [11]:
```
print("Decrypted e: ", decrypted_e)
```

```
Decrypted e:  5
```

Homomorphic Encryption for Machine Learning

Logistic Regression for Spam/Not Spam e-mail Classification.

For this problem we have two users:

USER-1

USER-2

AI Inc. makes a Machine Learning model that is trained on some email data for classification between Spam/Not Spam. Now, they want to take that model, encrypt it and send to USER-1 and USER-2 who will train the model on their data, fully Homomorphically Encrypted, and send the trained, a bit better model back to AI Inc.

In this process, AI Inc. get a better trained model every time without even looking at USER-1 or USER-2 data. This way AI Inc. can serve the customers better with a smart Machine Learning model and the USER has complete control of his/her data.

In [12]:
```
# Import Dependencies

import time
import os.path
from zipfile import ZipFile
from urllib.request import urlopen
from contextlib import contextmanager

import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer
```

In [33]:
```
import gensim
from gensim.models import word2vec, KeyedVectors
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
```

In [49]:
```
# Data Preprocessing
def preprocess_data():
    """
    Load the email dataset and Represent them as bag-of-words.
    Shuffle and split train/test.
    """

    print("Importing dataset...")
    #path = './dataset/enron1/ham/'
    path = "C:/Users/panka/Desktop/PROJECT MSC/EncriptedMachineLearning/enron1/ham/"
    ham1 = [open(path + f, 'r', errors='replace').read().strip(r"\n")
            for f in os.listdir(path) if os.path.isfile(path + f)]
    #path = './dataset/enron1/spam/'
    path = "C:/Users/panka/Desktop/PROJECT MSC/EncriptedMachineLearning/enron1/spam/"
    spam1 = [open(path + f, 'r', errors='replace').read().strip(r"\n")
             for f in os.listdir(path) if os.path.isfile(path + f)]
    #path = './dataset/enron2/ham/'
    path = "C:/Users/panka/Desktop/PROJECT MSC/EncriptedMachineLearning/enron2/ham/"
    ham2 = [open(path + f, 'r', errors='replace').read().strip(r"\n")
            for f in os.listdir(path) if os.path.isfile(path + f)]
    #path = './dataset/enron2/spam/'
    path = "C:/Users/panka/Desktop/PROJECT MSC/EncriptedMachineLearning/enron2/spam/"
    spam2 = [open(path + f, 'r', errors='replace').read().strip(r"\n")
             for f in os.listdir(path) if os.path.isfile(path + f)]

    # Merge and create labels
    emails = ham1 + spam1 + ham2 + spam2
    y = np.array([-1] * len(ham1) + [1] * len(spam1) +
                 [-1] * len(ham2) + [1] * len(spam2))

    # Words count, keep only frequent words
    # Minimum Document Word Frequency: 0.001
    count_vect = CountVectorizer(decode_error='replace',stop_words='english', min_df=0.001) # stop_words='english'
    X = count_vect.fit_transform(emails)

    print('Vocabulary size: %d' % X.shape[1])

    # Shuffle
    perm = np.random.permutation(X.shape[0])
    X, y = X[perm, :], y[perm]

    # Split train and test
    split = 500
    X_train, X_test = X[-split:, :], X[:-split, :]
    y_train, y_test = y[-split:], y[:-split]

    print("Labels in trainset are {:.2f} spam : {:.2f} ham".format(
        np.mean(y_train == 1), np.mean(y_train == -1)))

    return X_train, y_train, X_test, y_test
```

In [50]:
```
@contextmanager
def timer():
    """Helper for measuring runtime"""

    time0 = time.perf_counter()
    yield
    print('[elapsed time: %.2f s]' % (time.perf_counter() - time0))
```

In [51]:
```
class AI_Inc:
    """
    AI Inc. Trains a Logistic Regression model on plaintext data, encrypts the model for remote use by USER-1 and USER-2,
    decrypts encrypted scores using the paillier private key.
    """

    def __init__(self):
        self.model = LogisticRegression()

    # Generate Public and Private Key Pairs
    # Public Key is used to Encrypt the Data, Private Key to Decrypt
    def generate_paillier_keypair(self, n_length):
        self.pubkey, self.privkey = paillier.generate_paillier_keypair(n_length=n_length)

    # Train the Model
    def fit(self, X, y):
        self.model = self.model.fit(X, y)

    # Make Predictions for Email "Spam/Not Spam"
    def predict(self, X):
        return self.model.predict(X)

    # Encypt the Coefficients for the Logistic Regression Equation
    # Weights can tell about the data, so Encrypt them
    # Equation: y = mX + b
    def encrypt_weights(self):
        coef = self.model.coef_[0, :]
        encrypted_weights = [self.pubkey.encrypt(coef[i])
                             for i in range(coef.shape[0])]
        encrypted_intercept = self.pubkey.encrypt(self.model.intercept_[0])
        return encrypted_weights, encrypted_intercept

    # Decrypt the Scores for the Model
    def decrypt_scores(self, encrypted_scores):
        return [self.privkey.decrypt(s) for s in encrypted_scores]
```

In [52]:
```
# Now the USER-1 gets a trained model from AI Inc. and trains on its own data all using Homomorphic Encryption.
class User_1:
    """
    USER-1/USER-2 are given the encrypted model trained by AI Inc. and the public key.

    Scores local plaintext data with the encrypted model, but cannot decrypt
    the scores without the private key held by AI Inc..
    """

    def __init__(self, pubkey):
        self.pubkey = pubkey

    # Set Initial Values of Coefficients
    def set_weights(self, weights, intercept):
        self.weights = weights
        self.intercept = intercept

    # Compute the Prediction Scores for the Model all while being totally Encrypted.
    def encrypted_score(self, x):
        """Compute the score of `x` by multiplying with the encrypted model,
        which is a vector of `paillier.EncryptedNumber`"""
        score = self.intercept
        _, idx = x.nonzero()
        for i in idx:
            score += x[0, i] * self.weights[i]
        return score

    # Get the Evaluation Scores for the Model
    def encrypted_evaluate(self, X):
        return [self.encrypted_score(X[i, :]) for i in range(X.shape[0])]
```

In [53]:
```
# Get the Preprocessed Split Data
X_train, y_train, X_test, y_test = preprocess_data()
```

```
Importing dataset...
Vocabulary size: 7994
Labels in trainset are 0.28 spam : 0.72 ham
```

In [54]:
```
# Now firstly the AI Inc. Generates the Public and Private Keys
print("AI Inc.: Generating Paillier Public Private Keypair")
ai_inc = AI_Inc()
# NOTE: using smaller keys sizes wouldn't be cryptographically safe
ai_inc.generate_paillier_keypair(n_length=1024)
```

```
AI Inc.: Generating Paillier Public Private Keypair
```

In [55]:
```
print("AI Inc.: Training Initial Spam Classifier")
with timer() as t:
    ai_inc.fit(X_train, y_train)
```

```
AI Inc.: Training Initial Spam Classifier
[elapsed time: 0.43 s]
```

In [56]:
```
print("AI Inc.'s Classification on Test Data, what it would expect the performance to be on USER-1/2's data...")
with timer() as t:
    error = np.mean(ai_inc.predict(X_test) != y_test)
print("Error {:.3f}".format(error))
```

```
AI Inc.'s Classification on Test Data, what it would expect the performance to be on USER-1/2's data...
[elapsed time: 0.01 s]
Error 0.042
```

In [57]:
```
print("AI Inc.: Encrypting Trained Classifier before sending to USER-1/2")
with timer() as t:
    encrypted_weights, encrypted_intercept = ai_inc.encrypt_weights()
```

```
AI Inc.: Encrypting Trained Classifier before sending to USER-1/2
[elapsed time: 93.27 s]
```

In [58]:
```
# Confirming the Weights are Encrypted
print("Encrypted Weights: ", encrypted_weights)
print("Encrypted Intercept: ", encrypted_intercept)
```

```
>, <phe.paillier.EncryptedNumber object at 0x000001B8063B6AF0>, <phe.paillier.EncryptedNumber object at 0x000001B8063B6A00>, <phe.paillier.EncryptedNumber object at 0x000001B806
408DC0>, <phe.paillier.EncryptedNumber object at 0x000001B806408E20>, <phe.paillier.EncryptedNumber object at 0x000001B8064086D0>, <phe.paillier.EncryptedNumber object at 0x0000
01B806408EE0>, <phe.paillier.EncryptedNumber object at 0x000001B806408E80>, <phe.paillier.EncryptedNumber object at 0x000001B806408D30>, <phe.paillier.EncryptedNumber object at
0x000001B8064085E0>, <phe.paillier.EncryptedNumber object at 0x000001B806408490>, <phe.paillier.EncryptedNumber object at 0x000001B8064083A0>, <phe.paillier.EncryptedNumber obje
ct at 0x000001B8064084C0>, <phe.paillier.EncryptedNumber object at 0x000001B806408C70>, <phe.paillier.EncryptedNumber object at 0x000001B806408A00>, <phe.paillier.EncryptedNumbe
r object at 0x000001B806408CA0>, <phe.paillier.EncryptedNumber object at 0x000001B806408310>, <phe.paillier.EncryptedNumber object at 0x000001B806408670>, <phe.paillier.Encrypte
dNumber object at 0x000001B8064085B0>, <phe.paillier.EncryptedNumber object at 0x000001B8064086A0>, <phe.paillier.EncryptedNumber object at 0x000001B806408640>, <phe.paillier.En
cryptedNumber object at 0x000001B806408100>, <phe.paillier.EncryptedNumber object at 0x000001B806408460>, <phe.paillier.EncryptedNumber object at 0x000001B8064080D0>, <phe.paill
ier.EncryptedNumber object at 0x000001B806408280>, <phe.paillier.EncryptedNumber object at 0x000001B806408160>, <phe.paillier.EncryptedNumber object at 0x000001B806408340>, <ph
e.paillier.EncryptedNumber object at 0x000001B8064081C0>, <phe.paillier.EncryptedNumber object at 0x000001B806408250>, <phe.paillier.EncryptedNumber object at 0x000001B806408700
>, <phe.paillier.EncryptedNumber object at 0x000001B806408F40>, <phe.paillier.EncryptedNumber object at 0x000001B806408730>, <phe.paillier.EncryptedNumber object at 0x000001B806
408BB0>, <phe.paillier.EncryptedNumber object at 0x000001B806408B20>, <phe.paillier.EncryptedNumber object at 0x000001B8064088B0>, <phe.paillier.EncryptedNumber object at 0x0000
01B806408970>, <phe.paillier.EncryptedNumber object at 0x000001B8064088E0>, <phe.paillier.EncryptedNumber object at 0x000001B806408070>, <phe.paillier.EncryptedNumber object at
0x000001B806408F70>, <phe.paillier.EncryptedNumber object at 0x000001B8080C4D00>, <phe.paillier.EncryptedNumber object at 0x000001B8080C4700>, <phe.paillier.EncryptedNumber obje
ct at 0x000001B8080C4790>, <phe.paillier.EncryptedNumber object at 0x000001B8080C4670>, <phe.paillier.EncryptedNumber object at 0x000001B8080C4F10>, <phe.paillier.EncryptedNumbe
r object at 0x000001B8080C41F0>, <phe.paillier.EncryptedNumber object at 0x000001B8080C4370>, <phe.paillier.EncryptedNumber object at 0x000001B8080C42E0>, <phe.paillier.Encrypte
dNumber object at 0x000001B8080C4190>, <phe.paillier.EncryptedNumber object at 0x000001B8080C49A0>, <phe.paillier.EncryptedNumber object at 0x000001B8080C4160>, <phe.paillier.En
cryptedNumber object at 0x000001B8080C4250>, <phe.paillier.EncryptedNumber object at 0x000001B8080C4A90>, <phe.paillier.EncryptedNumber object at 0x000001B8080C4340>, <phe.paill
ier.EncryptedNumber object at 0x000001B8080C4EE0>, <phe.paillier.EncryptedNumber object at 0x000001B8080C4EB0>, <phe.paillier.EncryptedNumber object at 0x000001B8080C4E50>, <ph
```

Now, we have an encrypted trained model.

AI Inc. sends the trained model with it's weights encrypted [as weights can tell something about the data] and sends both the things to the USER-1 and USER2.

Now, USER-1 and USER-2 get the encrypted weights, the trained model and the public key to do some operations on their own dataset. This is called **Homomorphic Encryption**.

In [59]:
```
# USER-1 taking the encrypted model, weights and testing performance on it's own dataset
print("USER-1: Scoring on own data with AI Inc.'s Encrypted Classifier...")

# AI Inc sends the Public Keys to perform operations
user_1 = User_1(ai_inc.pubkey)

# USER-1 sets the model Hyperparameters to AI Inc.'s Hyperparameter values
user_1.set_weights(encrypted_weights, encrypted_intercept)

with timer() as t:
    encrypted_scores = user_1.encrypted_evaluate(X_test)
```

```
USER-1: Scoring on own data with AI Inc.'s Encrypted Classifier...
[elapsed time: 116.02 s]
```

In [60]:
```
# Making Sure the Score is Encrypted
print(encrypted_scores)
```

```
bject at 0x000001B807A4B970>, <phe.paillier.EncryptedNumber object at 0x000001B807A4B910>, <phe.paillier.EncryptedNumber object at 0x000001B807A4BBB0>, <phe.paillier.EncryptedNu
mber object at 0x000001B807A4B940>, <phe.paillier.EncryptedNumber object at 0x000001B807A4BBE0>, <phe.paillier.EncryptedNumber object at 0x000001B807A4BCA0>, <phe.paillier.Encry
ptedNumber object at 0x000001B807A4BB50>, <phe.paillier.EncryptedNumber object at 0x000001B807A4BD30>, <phe.paillier.EncryptedNumber object at 0x000001B807A4BC10>, <phe.paillie
r.EncryptedNumber object at 0x000001B807A4BD90>, <phe.paillier.EncryptedNumber object at 0x000001B807A4BE20>, <phe.paillier.EncryptedNumber object at 0x000001B807A4BDC0>, <phe.p
aillier.EncryptedNumber object at 0x000001B807A4BDF0>, <phe.paillier.EncryptedNumber object at 0x000001B807A4BFA0>, <phe.paillier.EncryptedNumber object at 0x000001B807A4BE80>,
<phe.paillier.EncryptedNumber object at 0x000001B807A4BF10>, <phe.paillier.EncryptedNumber object at 0x000001B807A53040>, <phe.paillier.EncryptedNumber object at 0x000001B807A533
0D0>, <phe.paillier.EncryptedNumber object at 0x000001B807A4BF40>, <phe.paillier.EncryptedNumber object at 0x000001B807A53220>, <phe.paillier.EncryptedNumber object at 0x000001B
807A53070>, <phe.paillier.EncryptedNumber object at 0x000001B807A53250>, <phe.paillier.EncryptedNumber object at 0x000001B807A53340>, <phe.paillier.EncryptedNumber object at 0x0
00001B807A533A0>, <phe.paillier.EncryptedNumber object at 0x000001B807A531F0>, <phe.paillier.EncryptedNumber object at 0x000001B807A53310>, <phe.paillier.EncryptedNumber object
at 0x000001B807A532B0>, <phe.paillier.EncryptedNumber object at 0x000001B807A53550>, <phe.paillier.EncryptedNumber object at 0x000001B807A534F0>, <phe.paillier.EncryptedNumber o
bject at 0x000001B807A53580>, <phe.paillier.EncryptedNumber object at 0x000001B807A53610>, <phe.paillier.EncryptedNumber object at 0x000001B807A53670>, <phe.paillier.EncryptedNu
mber object at 0x000001B807A53430>, <phe.paillier.EncryptedNumber object at 0x000001B807A536A0>, <phe.paillier.EncryptedNumber object at 0x000001B807A53760>, <phe.paillier.Encry
ptedNumber object at 0x000001B807A53640>, <phe.paillier.EncryptedNumber object at 0x000001B807A53880>, <phe.paillier.EncryptedNumber object at 0x000001B807A535E0>, <phe.paillie
r.EncryptedNumber object at 0x000001B807A53970>, <phe.paillier.EncryptedNumber object at 0x000001B807A53940>, <phe.paillier.EncryptedNumber object at 0x000001B807A53A00>, <phe.p
aillier.EncryptedNumber object at 0x000001B807A53910>, <phe.paillier.EncryptedNumber object at 0x000001B807A539A0>, <phe.paillier.EncryptedNumber object at 0x000001B807A53A60>,
<phe.paillier.EncryptedNumber object at 0x000001B807A53B50>, <phe.paillier.EncryptedNumber object at 0x000001B807A53C10>, <phe.paillier.EncryptedNumber object at 0x000001B807A53
B80>, <phe.paillier.EncryptedNumber object at 0x000001B807A53BE0>, <phe.paillier.EncryptedNumber object at 0x000001B807A53700>, <phe.paillier.EncryptedNumber object at 0x000001B
807A53D60>, <phe.paillier.EncryptedNumber object at 0x000001B807A53D00>, <phe.paillier.EncryptedNumber object at 0x000001B807A53D90>, <phe.paillier.EncryptedNumber object at 0x0
00001B807A53DF0>, <phe.paillier.EncryptedNumber object at 0x000001B807A53E50>, <phe.paillier.EncryptedNumber object at 0x000001B807A53F10>, <phe.paillier.EncryptedNumber object
at 0x000001B807A53EA0>, <phe.paillier.EncryptedNumber object at 0x000001B807A53F70>, <phe.paillier.EncryptedNumber object at 0x000001B807A53FD0>, <phe.paillier.EncryptedNumber o
```

Now USER has the option to train the model on it's own data and send the trained model to AI Inc.

In [61]:
```
print("AI Inc.: Decrypting USER-1/2's scores")

with timer() as t:
    scores = ai_inc.decrypt_scores(encrypted_scores)
    error = np.mean(np.sign(scores) != y_test)
print("Error {:.3f} -- this is not known to AI Inc., who does not possess the ground truth labels".format(error))
```

```
AI Inc.: Decrypting USER-1/2's scores
[elapsed time: 36.04 s]
Error 0.042 -- this is not known to AI Inc., who does not possess the ground truth labels
```

In [ ]: