Information Security Term Project

Loan Approval Prediction

18102087 Yeongju Lee
19102083 Hanseung Kim
20102122 Hyoan Jeong

Table of Contents

Introduction of Our Application

- **Motivation**
  - ➢ Personal information must be protected
  - ➢ In many case, personal information may be dealt with by third parties with providers' agreement, for a certain reasons like to get the services we need
  - ➢ Especially, **individual's financial status** is personal information which is reluctant to exposed to others
- **Topic:** Loan approval classifier
- **Used dataset :** Loan Approval Prediction Dataset
  - ➢ (https://www.kaggle.com/datasets/sonalisingh1411/loan-approval-prediction?resource=download&select=Training+Dataset.csv)
- **Used algorithm: Logistic Regression**
  - ➢ When we receive clients' personal information, we assess whether to get loan approve or not

Prior Plan and Challenges

- **We planned to make logistic model without using scikit learn.**
- **Architecture of our algorithm until progress report**
  - ➢ sigmoid() : to compute sigmoid value from prediction value
  - ➢ fit() : to reset the beta values based on the loss function to find the optimized beta of the model
  - ➢ predict() : to do prediction
  - ➢ _loss() : to determine the output of loss function for each round
- **Challenges**
  - ➢ Limitation of Heaan's method
    - \- Matrix structure
    - \- Inner Product
  - ➢ Array-based processing

```python
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```python
def predict(self, X):
    if self.fit_intercept:
        X = self._add_intercept(X)

    # 예측값 계산
    y_pred = sigmoid(np.dot(X, self.theta))

    # 0과 1로 변환
    return np.round(y_pred)

def _loss(self, X, y):
    y_pred = sigmoid(np.dot(X, self.theta))
    loss = -(y * np.log(y_pred) + (1 - y) * np.log(1 - y_pred)).mean()
    return loss
```

```python
def fit(self, X, y):
    if self.fit_intercept:
        X = self._add_intercept(X)

    #(200*10) * (10*1)
    # betas 초기화
    self.theta = np.zeros(X.shape[1]) #열열

    for i in range(self.num_iter):
        # 예측값 계산
        y_pred = sigmoid(np.dot(X, self.theta))

        # 오차 계산
        error = y_pred - y

        # 경사 하강법
        gradient = np.dot(X.T, error)

        # 학습률 조정
        self.lr *= (1/(1 + 0.001*i))

        self.theta -= self.lr * gradient

        if self.verbose and i % 10000 == 0:
            loss = self._loss(X, y)
            print(f'iteration {i}, loss {loss}')
```

Prior Plan and Changes

- **Understanding the existing code**
  - ➢ Heean performs array-based operations
  - ➢ Variables in existing code: ctxt_beta, ctxt_beta0m msg_mask, …
  - ➢ Heean's method
    - ■ left_lotate & right_lotate, mult

ctxt_tmp	b1x11	b1x12	b2x21	b2x22	b3x31	b3x32	b4x41	b4x42	b5x51	b5x52	b6x61	b6x62	b7x71	b7x72	b8x81	b8x82	b01	b02
n = 2																		
I = 0, n*2**(2-i) = 8																		
ctxt_rot	b5x51	b5x52	b6x61	b6x62	b7x71	b7x72	b8x81	b8x82	b01	b02	b1x11							
ctxt_tmp	b1x11 + b5x51	b1x12 + b5x52	b2x21 + b6x61	b2x22 + b6x62	b3x31 + b7x71	b3x32 + b7x72	b4x41 + b8x81	b4x42 + b8x82	b5x51 + b01	b5x52 + b02	b6x61 + b1x11							
I = 1, 2*2**(2-1) = 4																		
ctxt_rot	b3x31 + b7x71	b3x32 + b7x72	b4x41 + b8x81	b4x42 + b8x82	b5x51 + b01	b5x52 + b02	b6x61 + b1x11	b6x62 + b1x12	b7x71 + b2x21	b7x72 + b2x22	b8x81 + b3x31							
ctxt_tmp	b1x11 + b5x51 + b3x31 + b7x71	b1x12 + b5x52 + b3x32 + b7x72	b2x21 + b6x61 + b4x41 + b8x81	b2x22 + b6x62 + b4x42 + b8x82	b3x31 + b7x71 + b5x51 + b01	b3x32 + b7x72 + b5x52 + b02	b4x41 + b8x81 + b6x61 + b1x11	b4x42 + b8x82 + b6x62 + b1x12	b5x51 + b01 + b7x71 + b2x21	b5x52 + b02 + b7x72 + b2x22	b6x61 + b1x11 + b8x81 + b3x31	b3x32	+ b4x41					

msg_mask	1	1	0	0
ctxt_tmp	b1x11 + b2x21 + b3x31 + b4x41 + b5x51 + b6x61 + b7x71 + b8x81 + b01	b1x12 + b2x22 + b3x32 + b4x42 + b5x52 + b6x62 + b7x72 + b8x82 + b02	0	0

```
ctxt_beta0 = heaan.Ciphertext(context)
eval.left_rotate(ctxt_beta, 8*n, ctxt_beta0)
```

```
msg_mask = heaan.Message(log_slots)
for i in range(n):
    msg_mask[i] = 1
eval.mult(ctxt_tmp, msg_mask, ctxt_tmp)
```

```
for i in range(3):
    eval.left_rotate(ctxt_tmp, n*2**(2-i), ctxt_rot)
    eval.add(ctxt_tmp, ctxt_rot, ctxt_tmp)
eval.add(ctxt_tmp, ctxt_beta0, ctxt_tmp)
```

Preprocessing

- **Preprocessing**
  - **Drop null** value
  - Convert ‘Y(loan approved)’ to 1 and ‘N(not approved)’ to 0
  - Split the dataset into test & training set to prevent data leakage
  - Conduct **get_dummies** for categorical features and **scaling** for numerical feature

```
x_train_cat = x_train[['Gender','Married','Dependents','Education','Self_Employed','Property_Area']]
x_train_num = x_train[['ApplicantIncome',	'CoapplicantIncome',	'LoanAmount',	'Loan_Amount_Term',	'Credit_History']]

x_train_dummies = pd.get_dummies(x_train_cat)

scaler = StandardScaler()
scaler.fit(x_train_num)
x_train_scaled = scaler.transform(x_train_num)
```

Trying to enhance the accuracy of model

- **Developing the model with entire dataset**

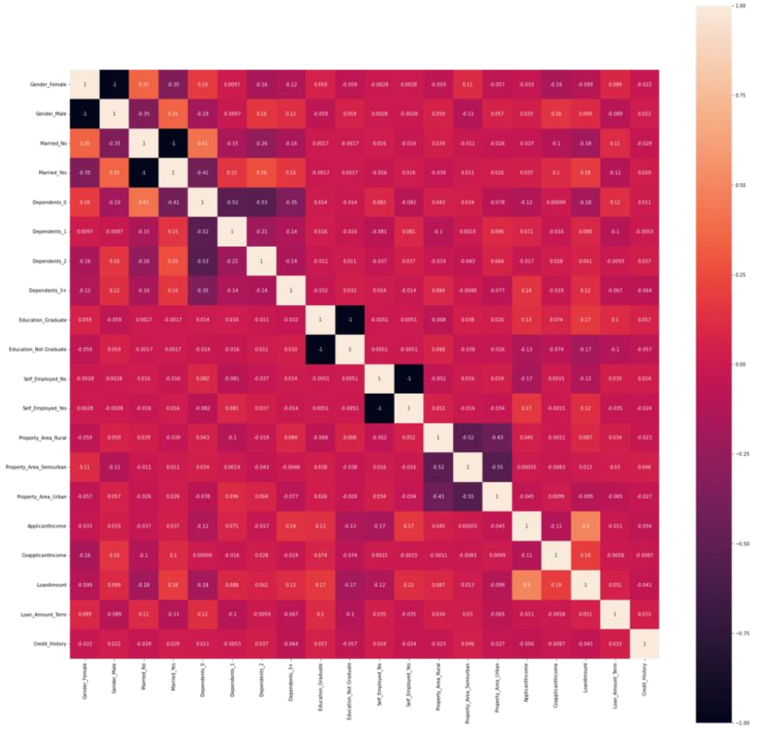
```
X[0] = list(x_train_trans['Gender_Female'].values)
X[1] = list(x_train_trans['Gender_Male'].values)
X[2] = list(x_train_trans['Married_No'].values)
X[3] = list(x_train_trans['Married_Yes'].values)
X[4] = list(x_train_trans['Dependents_0'].values)
X[5] = list(x_train_trans['Dependents_1'].values)
X[6] = list(x_train_trans['Dependents_2'].values)
X[7] = list(x_train_trans['Dependents_3+'].values)
X[8] = list(x_train_trans['Education_Graduate'].values)
X[9] = list(x_train_trans['Education_Not Graduate'].values)
X[10] = list(x_train_trans['Self_Employed_No'].values)
X[11] = list(x_train_trans['Self_Employed_Yes'].values)
X[12] = list(x_train_trans['Property_Area_Rural'].values)
X[13] = list(x_train_trans['Property_Area_Semiurban'].values)
X[14] = list(x_train_trans['Property_Area_Urban'].values)
X[15] = list(x_train_trans['ApplicantIncome'].values)
X[16] = list(x_train_trans['CoapplicantIncome'].values)
X[17] = list(x_train_trans['LoanAmount'].values)
X[18] = list(x_train_trans['Loan_Amount_Term'].values)
X[19] = list(x_train_trans['Credit_History'].values)
```

```
accuracy : 0.7083333333333334
```

- It is regarded as not sufficient
- So, we got through the feature selection stage to enhance the performance

Trying to enhance the accuracy of model

- **Conducting correlation analysis(Correlation matrix)**



- Colored with white or black
  : indicates high correlationship

- Drop the columns with high correlationship

- 4 pairs of features are highly correlated

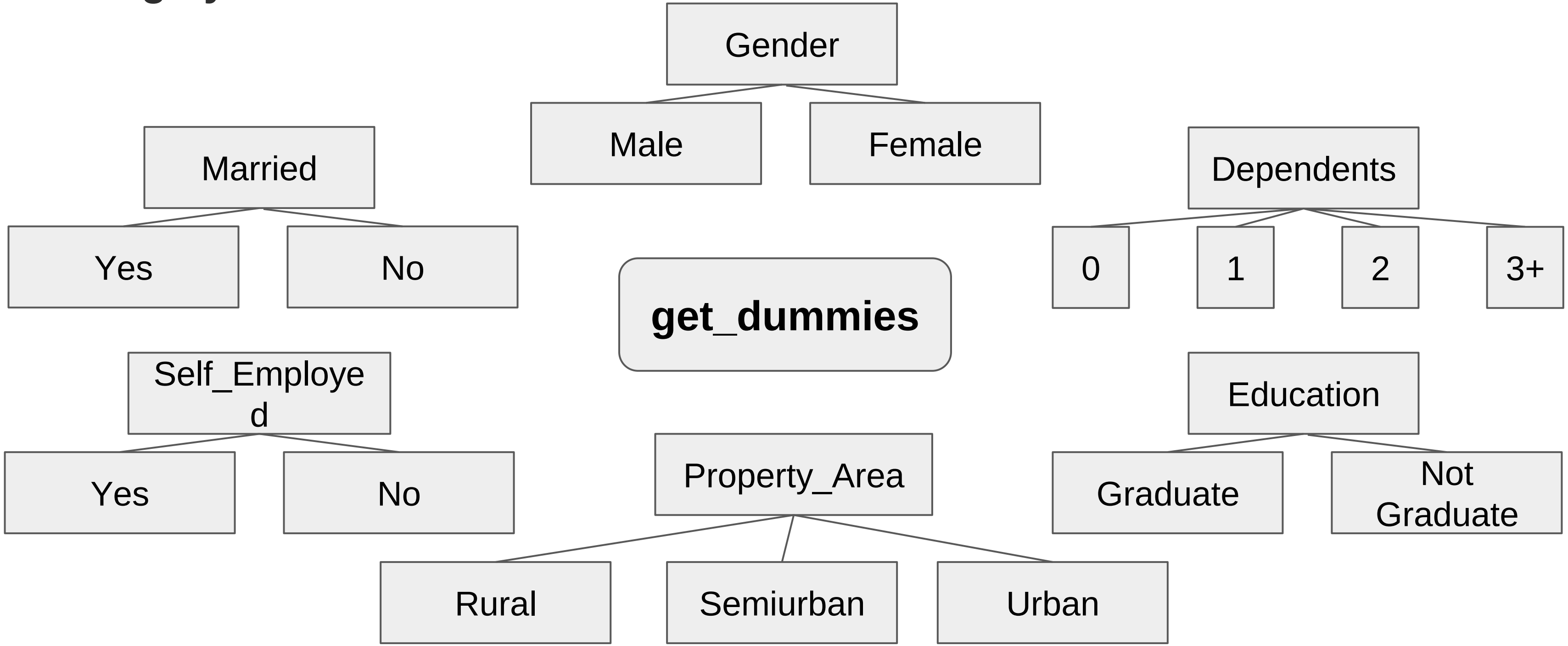
Trying to enhance the accuracy of model

- **Highly correlated features**

```
x_train_cat = x_train[['Gender','Married','Dependents','Education','Self_Employed','Property_Area']]
x_train_num = x_train[['ApplicantIncome',	'CoapplicantIncome',	'LoanAmount',	'Loan_Amount_Term',	'Credit_History']]

x_train_dummies = pd.get_dummies(x_train_cat)

x_train_dummies = x_train_dummies.drop(['Gender_Male', 'Married_No', 'Education_Not Graduate', 'Self_Employed_No'], axis = 1)

scaler = StandardScaler()
scaler.fit(x_train_num)
x_train_scaled = scaler.transform(x_train_num)
```

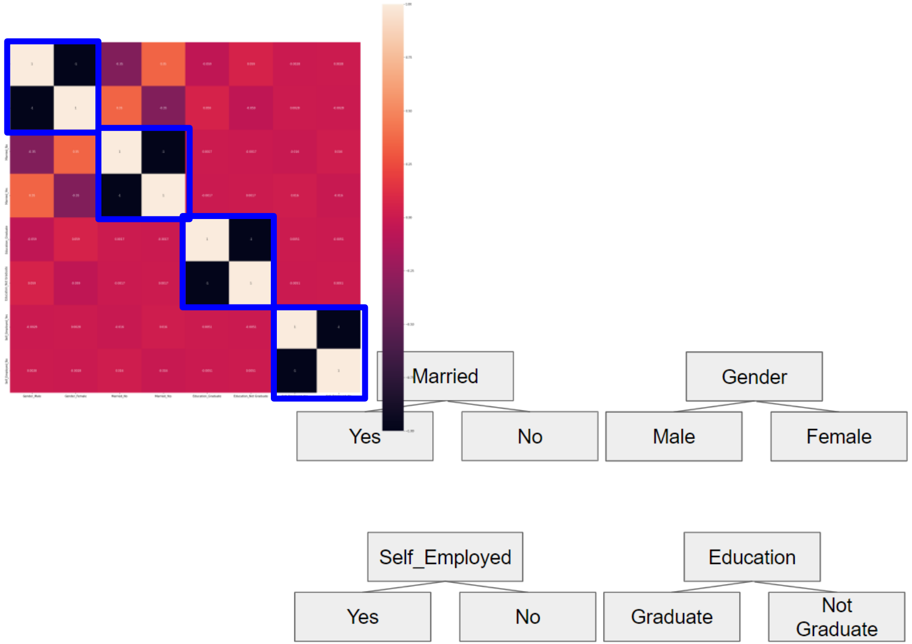
Trying to enhance the accuracy of model

- **Highly correlated features**



Trying to enhance the accuracy of model

- **Highly correlated features**



- 4 pairs of features that are highly correlated
- Remove one feature from each pair
- Removed features

  Married_No

  Gender_Male

  Self_Employed_No

  Education_Not Graduate

Trying to enhance the accuracy of model

- **Highly correlated features**

```
x_train_cat = x_train[['Gender','Married','Dependents','Education','Self_Employed','Property_Area']]
x_train_num = x_train[['ApplicantIncome',	'CoapplicantIncome',	'LoanAmount',	'Loan_Amount_Term',	'Credit_History']]

x_train_dummies = pd.get_dummies(x_train_cat)

x_train_dummies = x_train_dummies.drop(['Gender_Male', 'Married_No', 'Education_Not Graduate', 'Self_Employed_No'], axis = 1)

scaler = StandardScaler()
scaler.fit(x_train_num)
x_train_scaled = scaler.transform(x_train_num)
```

Trying to enhance the accuracy of model

- **Developing the model with selected features**

```
X_test[0] = list(x_test_trans['Gender_Female'].values)
X_test[1] = list(x_test_trans['Married_Yes'].values)
X_test[2] = list(x_test_trans['Dependents_0'].values)
X_test[3] = list(x_test_trans['Dependents_1'].values)
X_test[4] = list(x_test_trans['Dependents_2'].values)
X_test[5] = list(x_test_trans['Dependents_3+'].values)
X_test[6] = list(x_test_trans['Education_Graduate'].values)
X_test[7] = list(x_test_trans['Self_Employed_Yes'].values)
X_test[8] = list(x_test_trans['Property_Area_Rural'].values)
X_test[9] = list(x_test_trans['Property_Area_Semiurban'].values)
X_test[10] = list(x_test_trans['Property_Area_Urban'].values)
X_test[11] = list(x_test_trans['ApplicantIncome'].values)
X_test[12] = list(x_test_trans['CoapplicantIncome'].values)
X_test[13] = list(x_test_trans['LoanAmount'].values)
X_test[14] = list(x_test_trans['Loan_Amount_Term'].values)
X_test[15] = list(x_test_trans['Credit_History'].values)
```

```
accuracy : 0.78125
```

- performance was improved with feature selection

Compare the accuracy

- **Developing the model without piheaan library**

  : performance evaluation is done with same settings
  ex) learning rate = 0.01, iteration = 100, same feature sets(after feature selection)

```
self.weights = 2 * np.random.rand(16) - 1
self.bias = 0

# 경사 하강법을 사용한 가중치와 편향 업데이트
for i in range(self.num_steps):
    train_inputs = np.dot(x_train_trans, self.weights) + self.bias
    predictions = self.sigmoid(train_inputs)

    # 가중치와 편향의 그래디언트 계산
    dw = (1 / train_n) * np.dot(x_train_trans.T, (predictions - y_train))
    db = (1 / train_n) * np.sum(predictions - y_train)

    # 가중치와 편향 업데이트
    self.weights -= self.learning_rate * dw
    self.bias -= self.learning_rate * db

test_inputs = np.dot(x_test_trans, self.weights) + self.bias
predictions = self.sigmoid(test_inputs)
predicted_classes = [1 if prediction.real >= 0.5 else 0 for prediction in predictions]
cnt = 0
y_test = y_test.reset_index(drop=True)
for i in range(test_n):
    if predicted_classes[i] == y_test[i]:
        cnt+=1
```

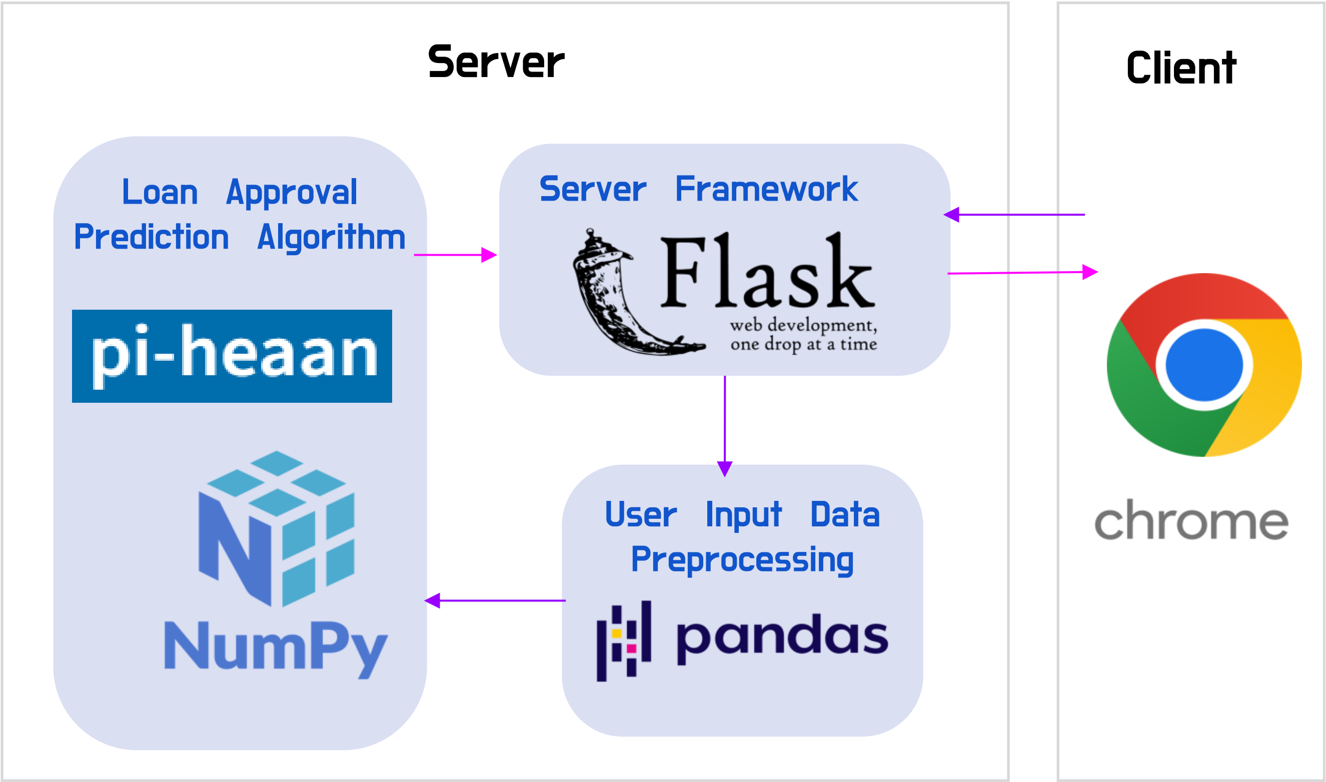
```
accuracy : 0.7916666666666666
```

Accuracy of homomorphic model

→ `accuracy : 0.78125`

: We can say that **Homomorphic encryption is not so harmful for accuracy**

Design Overview



Model Code

- **Methods**

```python
def enc_setting(self):
    ## set parameter -> 동형 암호를 위한 파라미터와 context를 아래와 같이 셋업한다
    params = heaan.ParameterPreset.FGb
    self.context = heaan.make_context(params) # context has paramter information
    heaan.make_bootstrappable(self.context) # make parameter bootstrapable
    # Bootstrapping:일반 동형암호 알고리즘이 완전 동형암호로 사용되기 위해 곱셈에서 증가하는 노이즈를 줄이는 과정

    ## create and save keys
    key_file_path = "./keys"
    self.sk = heaan.SecretKey(self.context) # create secret key
    os.makedirs(key_file_path, mode=0o775, exist_ok=True)
    # os.makedirs: py에서 폴더를 생성, exist_ok:폴더가 존재하지 않으면 생성, 존재하면 그냥 있음
    # mode(권한모드): A Integer value representing mode of the newly created directory. Default value 0o777 is used.
    self.sk.save(key_file_path+"/secretkey.bin") # save secret key

    self.key_generator = heaan.KeyGenerator(self.context, self.sk) # create public key
    self.key_generator.gen_common_keys()
    self.key_generator.save(key_file_path+"/") # save public key

    key_file_path = "./keys"

    self.sk = heaan.SecretKey(self.context,key_file_path+"/secretkey.bin") # load secret key
    self.pk = heaan.KeyPack(self.context, key_file_path+"/") # load public key
    self.pk.load_enc_key()
    self.pk.load_mult_key()

    self.eval = heaan.HomEvaluator(self.context,self.pk) # to load piheaan basic function
    self.dec = heaan.Decryptor(self.context) # for decrypt
    self.enc = heaan.Encryptor(self.context)# for encrypt

    log_slots = 15
    num_slots = 2**log_slots
```

- enc_setting()
  : setting several parameter which are needed for homomorphic encryption

Model Code

- **Methods**

```python
def training(self):
    accuracy=[]
    data= pd.read_csv('./Dataset.csv')
    data.dropna(inplace=True) # 결측치 제거

    target = data['Loan_Status'].apply(self.trans_target_type)
    data.drop('Loan_ID', axis=1, inplace=True) #인덱스 열 제거

    best_accuracy = 0
    start = time.time()
    for ittr_num in range(50):
        print("-----------------" + str(ittr_num)+"th test" + "-----------------")
        x = data[['Gender','Married','Dependents','Education','Self_Employed','ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_
        y = target

        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,stratify=y, random_state=0)

        x_train_cat = x_train[['Gender','Married','Dependents','Education','Self_Employed','Property_Area']]
        x_train_num = x_train[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_History']]

        x_train_dummies = pd.get_dummies(x_train_cat)

        x_train_dummies = x_train_dummies.drop(['Gender_Male', 'Married_No', 'Education_Not Graduate', 'Self_Employed_No'], axis = 1)

        scaler = StandardScaler()
        scaler.fit(x_train_num)
        x_train_scaled = scaler.transform(x_train_num)

        x_train_scaled_df = pd.DataFrame(x_train_scaled

        x_train_scaled_df.index = x_train_num.index
        x_train_scaled_df.columns = x_train_num.columns

        x_train_trans = pd.concat([x_train_dummies, x_t

        x_test_cat = x_test[['Gender','Married','Depend
        x_test_num = x_test[['ApplicantIncome', 'Coappl

        x_test_dummies = pd.get_dummies(x_test_cat)

        x_test_dummies = x_test_dummies.drop(['Gender_M
```

```python
scaler = StandardScaler()
scaler.fit(x_test_num)
x_test_scaled = scaler.transform(x_test_num)

x_test_scaled_df = pd.DataFrame(x_test_scaled)

x_test_scaled_df.index = x_test_num.index
x_test_scaled_df.columns = x_test_num.columns

x_test_trans = pd.concat([x_test_dummies, x_test_scaled_df], axis=1)

train_n = x_train_trans.shape[0]
```

- training()
  : load data and preprocess the data

- With data, train the logistic regression model(optimize beta with self.step())

Model Code

- **Methods**

```python
def step(self, learning_rate, ctxt_X, ctxt_Y, ctxt_beta, n, log_slots, context, eval):
    '''
    ctxt_X, ctxt_Y : data for training
    ctxt_beta : initial value beta
    n : the number of row in train_data 데이터 개수수
    '''
    ctxt_rot = heaan.Ciphertext(context)
    ctxt_tmp = heaan.Ciphertext(context)

    ## step1(가중치 갱신)
    # beta0
    ctxt_beta0 = heaan.Ciphertext(context)
    eval.left_rotate(ctxt_beta, 16*n, ctxt_beta0) #20:beta의 개수

    # compute  ctxt_tmp = beta1*x1 + beta2*x2 + ... + beta20*x20 +
    eval.mult(ctxt_beta, ctxt_X, ctxt_tmp)

    for i in range(4):
        eval.left_rotate(ctxt_tmp, n*2**(3-i), ctxt_rot)
        eval.add(ctxt_tmp, ctxt_rot, ctxt_tmp)
    eval.add(ctxt_tmp, ctxt_beta0, ctxt_tmp)

    msg_mask = heaan.Message(log_slots)
    for i in range(n):
        msg_mask[i] = 1
    eval.mult(ctxt_tmp, msg_mask, ctxt_tmp)

    ## step2
    # compute sigmoid
    approx.sigmoid(eval, ctxt_tmp, ctxt_tmp, 8.0)
    eval.bootstrap(ctxt_tmp, ctxt_tmp)
    msg_mask = heaan.Message(log_slots)
    # if sigmoid(0) -> return 0.5
    for i in range(n, self.num_slots):
        msg_mask[i] = 0.5
    eval.sub(ctxt_tmp, msg_mask, ctxt_tmp)

    ## step3
    # compute  (learning_rate/n) * (y_(j) - p_(j))
    ctxt_d = heaan.Ciphertext(context)
    eval.sub(ctxt_Y, ctxt_tmp, ctxt_d)
    eval.mult(ctxt_d, learning_rate / n, ctxt_d)

    eval.right_rotate(ctxt_d, 16*n, ctxt_tmp) # for beta0
    for i in range(4):
        eval.right_rotate(ctxt_d, n * 2**i, ctxt_rot)
        eval.add(ctxt_d, ctxt_rot, ctxt_d)
    eval.add(ctxt_d, ctxt_tmp, ctxt_d)
```

```python
## step4
# compute  (learning_rate/n) * (y_(j) - p_(j)) * x_(j)
ctxt_X_j = heaan.Ciphertext(context)
msg_X0 = heaan.Message(log_slots)
for i in range(16*n, 17*n):
    msg_X0[i] = 1
eval.add(ctxt_X, msg_X0, ctxt_X_j)
eval.mult(ctxt_X_j, ctxt_d, ctxt_d)

## step5
# compute  Sum_(all j) (learning_rate/n) * (y_(j) - p_(j)) * x_(j)
for i in range(5):
    eval.left_rotate(ctxt_d, 2**(16-i), ctxt_rot)
    eval.add(ctxt_d, ctxt_rot, ctxt_d)
msg_mask = heaan.Message(log_slots)

for i in range(20):
    msg_mask[i * n] = 1
eval.mult(ctxt_d, msg_mask, ctxt_d)

for i in range(10):
    eval.right_rotate(ctxt_d, 2**i, ctxt_rot)
    eval.add(ctxt_d, ctxt_rot, ctxt_d)

## step6
# update beta
eval.add(ctxt_beta, ctxt_d, ctxt_d)
return ctxt_d
```

- step method
  : optimizing beta

- almost same with example code
  customized with regard to our dataset

Model Code

- **Methods**

```
def compute_sigmoid(self, ctxt_X, ctxt_beta, n, log_slots, eval, context, num_slots):
    '''
    ctxt_X : data for evaluation
    ctxt_beta : estimated beta from function 'step'
    n : the number of row in test_data
    '''
    ctxt_rot = heaan.Ciphertext(context)
    ctxt_tmp = heaan.Ciphertext(context)

    # beta0
    ctxt_beta0 = heaan.Ciphertext(context)
    eval.left_rotate(ctxt_beta, 16*n, ctxt_beta0)

    # compute x * beta + beta0
    ctxt_tmp = heaan.Ciphertext(context)
    eval.mult(ctxt_beta, ctxt_X, ctxt_tmp)

    for i in range(4):
        eval.left_rotate(ctxt_tmp, n*2**(3-i), ctxt_rot)
        eval.add(ctxt_tmp, ctxt_rot, ctxt_tmp)
    eval.add(ctxt_tmp, ctxt_beta0, ctxt_tmp)

    msg_mask = heaan.Message(log_slots)
    for i in range(n):
        msg_mask[i] = 1
    eval.mult(ctxt_tmp, msg_mask, ctxt_tmp)

    # compute sigmoid
    approx.sigmoid(eval, ctxt_tmp, ctxt_tmp, 8.0)
    eval.bootstrap(ctxt_tmp, ctxt_tmp)
    msg_mask = heaan.Message(log_slots)
    for i in range(n, num_slots):
        msg_mask[i] = 0.5
    eval.sub(ctxt_tmp, msg_mask, ctxt_tmp)

    return ctxt_tmp
```

- compute_sigmoid method

  : compute sigmoid value with optimized beta

- almost same with example code

  customized with regard to our dataset

Model Code

- **Methods**

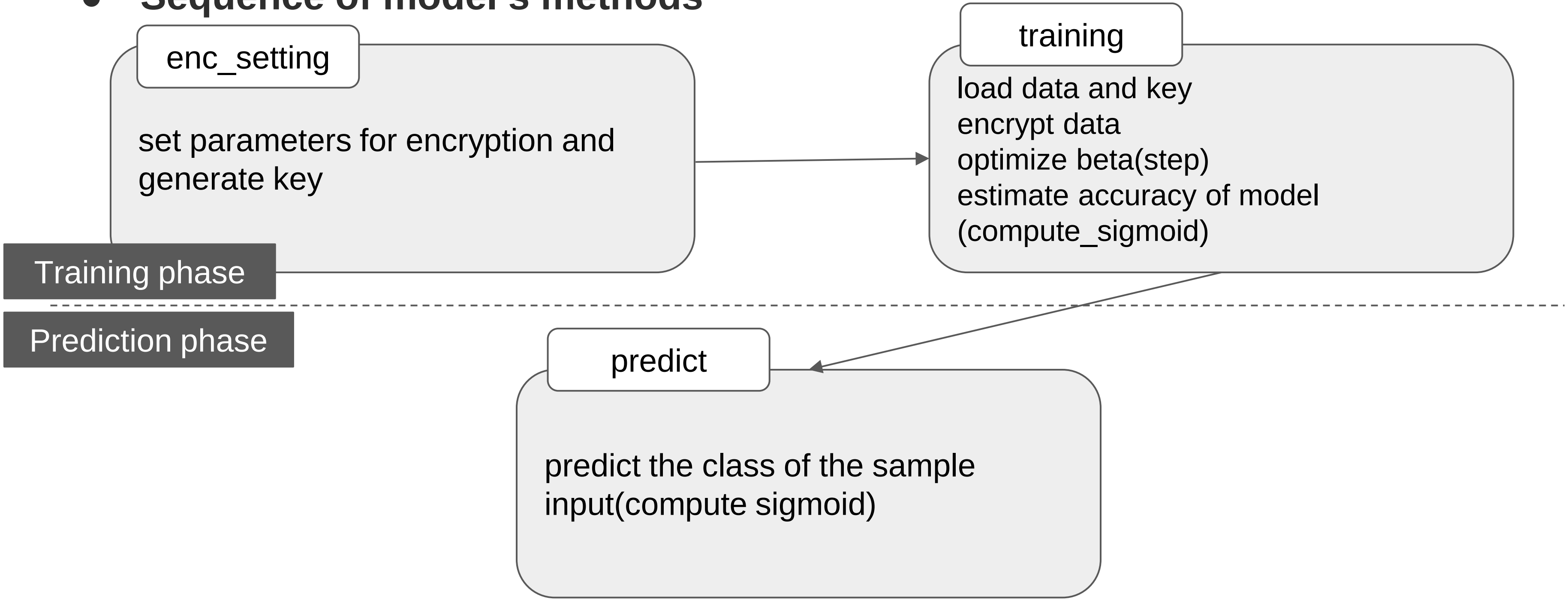
```
def predict(self, X):
    print("current beta : ", self.betas)
    msg_X_test = heaan.Message(self.log_slots)
    ctxt_X_test = heaan.Ciphertext(self.context)

    for i in range(16):
        msg_X_test[i] = X[i]

    self.enc.encrypt(msg_X_test, self.pk, ctxt_X_test)

    sigmoid = self.compute_sigmoid(ctxt_X_test, self.betas, 1, 15, self.eval,self.context, 2**15)
    print("sigmoid value : ", sigmoid)
    res = heaan.Message(self.log_slots)
    self.dec.decrypt(sigmoid, self.sk, res)
    if res[0].real >= 0.50:
        return 1
    else :
        return 0
```

- predict method
  : with given X(sample), make prediction

- threshold : 0.5

Model Code

- **Sequence of model's methods**



Application code

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <center>
{#    # The screen where users enter their personal information to see their loan approval prediction results#}
    <form action="{{url_for('result3')}}" method="post">

      <h1>LOAN APPROVAL</h1>
      <h4>To see if you're eligible for a loan, please fill out the information below<br></h4>
      Name: <input type="text" name="uname" /> <br><br>

      Gender:
      <input type="radio" name="gender" value=0 />Male
      <input type="radio" name="gender" value=1 />Female<br><br>

      Are you Married?
      <input type="radio" name="mrg" value=1 />Yes
      <input type="radio" name="mrg" value=0 />No<br><br>

      Are you a graduate?
      <input type="radio" name="edu" value=1 />Yes
      <input type="radio" name="edu" value=0 />No<br><br>

      Are you self-employed?
      <input type="radio" name="self" value=1 />Yes
      <input type="radio" name="self" value=0 />No<br><br>
```

main.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
      <center>
{#        This screen shows the prediction results based on the personal information entered by the user.#}
        <br><br><br>
        <h1 style="...">  PREDICTION  </h1> <br><br><br><br><br>

    {%if data == 0%}
    <style>
        body {
            background-color: lightcoral;
        }
    </style>
    <h1 style="..."> {{user}}, You can't get a loan :(</h1>

    {%else%}
    <style>
        body {
            background-color: yellowgreen;
        }
    </style>
    <h1 style="..."> {{user}}, You are eligible for a loan from the bank! :)</h1>

    {%endif%}
        <br>
```

result.html

Application code

```python
# import the model and train o
model = LR_with_enc.LR_enc()
model.enc_setting()
model.training()
```

```python
if __name__ == '__main__':
    app.run(debug=True)
```

```python
@app.route('/')
def main():
    return render_template('main.html')

# Executed when a client sends a POST method requ
@app.route('/result3', methods=['POST'])
def result3():
    # Save the personal information entered by th
    name = request.form['uname']
    gender = int(request.form['gender'])
    mrg = int(request.form['mrg'])
    edu = int(request.form['edu'])
    self = int(request.form['self'])
    dep = request.form['dep']
    income = int(request.form['income'])
    co_income = int(request.form['co_income'])
    credit = int(request.form['credit'])
    prop = request.form['prop']
    amount = int(request.form['amount'])
    term = int(request.form['term'])
```

- Import the model and train once before running the service

- Show the screen of a form where users can enter personal information to predict whether they will be approved for a loan

- If client sends a POST method request to the '/result3' path, Save the personal information entered by the user in the appropriate data type

Application code

```python
# Make a list of the stored values. In this case, all values selected by the user other than t
list = [gender, mrg, 0, 0, 0, 0, edu, self, income, co_income, amount, term, credit, 0, 0, 0]
```

```python
# Based on the value selec
if dep == "0":
    list[2] = 1
elif dep == "1":
    list[3] = 1
elif dep == "2":
    list[4] = 1
elif dep == "3+":
    list[5] = 1
```

```python
if prop == "Urban":
    list[13] = 1
elif prop == "Semiurban":
    list[14] = 1
elif prop == "Rural":
    list[15] = 1
```

```python
df = pd.DataFrame(data=[list], columns=['Gender_Female', 'Married_Yes', 'Dependents_0', 'D
df = preprocess(df)

arr = df.values.tolist() # Convert the preprocessed data back to an array format for the m
pred = model.predict(arr[0]) # Loan approval prediction results
return render_template('result.html', data=pred, user=name) # pass in the prediction and t
```

- Make a list of the stored values. In this case, all values selected by the user other than those entered directly are zeroed.
- Based on the value selected by the user, change only that value from 0 to 1 among several values
- To preprocess the user input data, convert the data into dataframe type
- After preprocessing, convert the preprocessed data back to an array format for the model to predict whether the loan will be approved
- Pass in the prediction and the user's name, and render the result page

final - Google Drive | 발표 개략 - Google Docs | IS_Term_final - Google Slides | Document

127.0.0.1:5000

LOAN APPROVAL

To see if you're eligible for a loan, please fill out the information below

Name:

Gender: ○ Male ○ Female

Are you Married? ○ Yes ○ No

Are you a graduate? ○ Yes ○ No

Are you self-employed? ○ Yes ○ No

How many dependents do you have? ○ 0 ○ 1 ○ 2 ○ 3+

Income($):

Coapplicant income($):

Do you have a credit history that meets our guidelines? ○ Yes ○ No

Where is the property area? ○ Urban ○ Semiurban ○ Rural

Desired loan amount (1000$):

Desired loan term (months):

Submit

Conclusions & Limitations

- There are no significant accuracy difference between normal model and model with homomorphic encryption
  : homomorphic encryption can serve almost same service with much higher security

- **Not all the settings are same between tests**
  : Gradient descent function may not exactly be same between homomorphic model and normal model(LR model without piheaan library)
  → accuracy comparison may be not accurate

- **Time consumed for training is far later than normal model (time unit : sec)**

```
normal model training time :  0.1967738
homomorphic enc model training time :  7.1871798
```