RecoSys For HeartDisease

Group 9

2018126131 Huh Jaehyuk
2018126132 Heo Jinnyeong
2018125078 Kim Dongchan
2018125079 Kim Chanju
2017125081 Yoo Seungkyung

: Contents

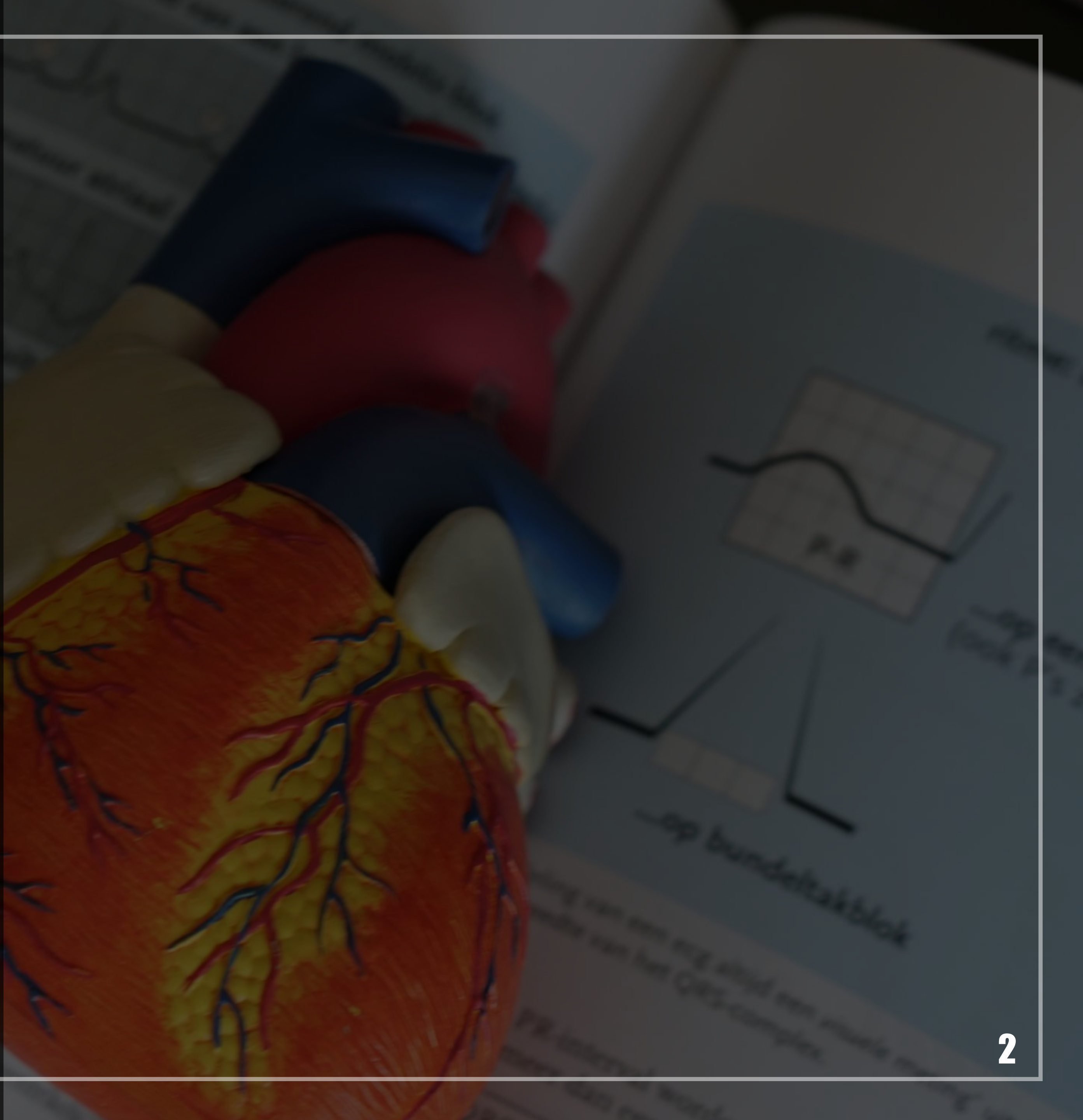
| 1st Dataset

| 2nd Models

| 3rd Selected Model

| 4th Application for RecoSys

| 5th Conclusion & Limitations



01

Dataset

Total 319.7k

17 features

3 features

Data source
Center for Disease Control and Prevention
2020 survey data

Features
BMI, Smoking, AlcoholDrinking, Stroke,
Physical Health, Mental Health, DiffWalking
etc.

Key Features
Stroke, DiffWalking, AgeCategory

01

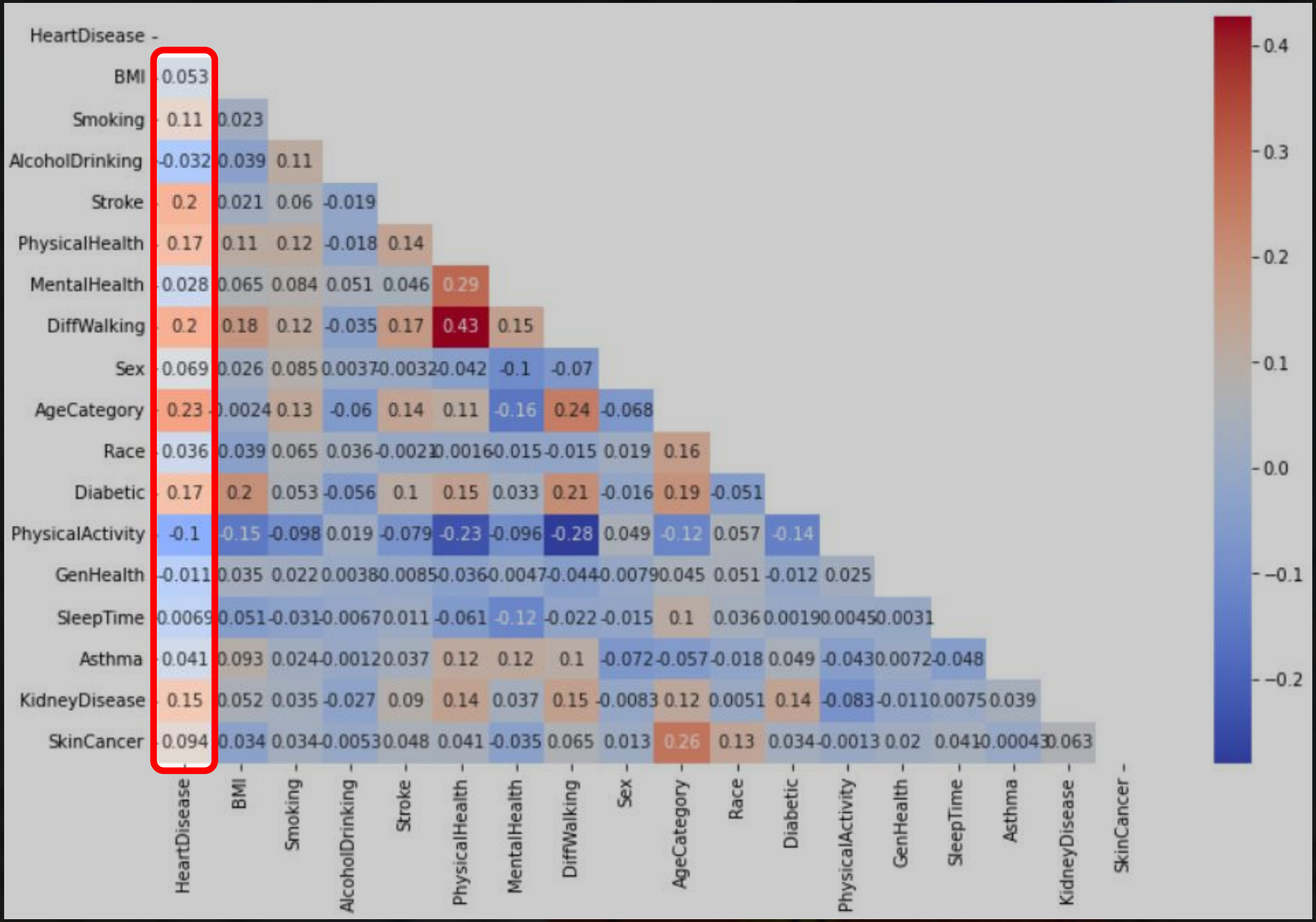
Dataset

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory
0	No	16.60	Yes	No	No	3.0	30.0	No	Female	55-59
1	No	20.34	No	No	Yes	0.0	0.0	No	Female	80 or older
2	No	26.58	Yes	No	No	20.0	30.0	No	Male	65-69
3	No	24.21	No	No	No	0.0	0.0	No	Female	75-79
4	No	23.71	No	No	No	28.0	0.0	Yes	Female	40-44

	Race	Diabetic	PhysicalActivity	GenHealth	SleepTime	Asthma	KidneyDisease	SkinCancer
0	White	Yes	Yes	Very good	5.0	Yes	No	Yes
1	White	No	Yes	Very good	7.0	No	No	No
2	White	Yes	Yes	Fair	8.0	Yes	No	No
3	White	No	No	Good	6.0	No	No	Yes
4	White	No	Yes	Very good	8.0	No	No	No

01
Dataset

Correlation Matrix



01

Dataset

Task

Key Features

By Machin Learning Models →

Binary Classification

Identify which feature is key feature
Show the rate of contraction
Recommend How to reduce incidence

: Contents

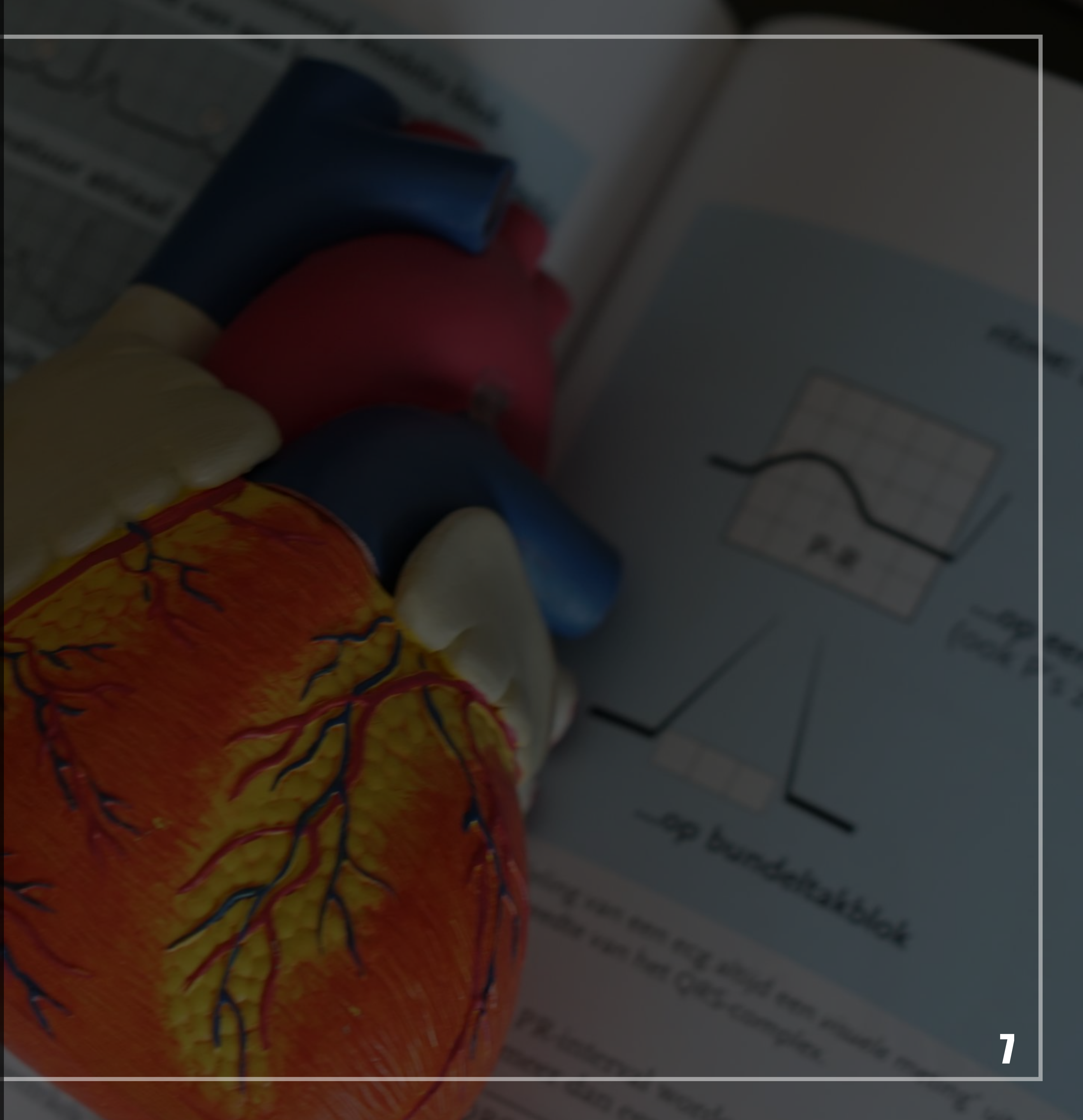
| 1st Dataset

| 2nd Models

| 3rd Selected Model

| 4th Application for RecoSys

| 5th Conclusion & Limitations



02

Models

Logistic Regression

SVM
(Support Vector Machines)

TSNE
(T-Distributed Stochastic Neighbor Embedding)

KNN
(k-Nearest Neighbor)

ANN
(Artificial Neural Network)

02
Models

Data Preprocessing

```
label_encoder = preprocessing.LabelEncoder()

df['HeartDisease']= label_encoder.fit_transform(df['HeartDisease'])
df['Smoking']= label_encoder.fit_transform(df['Smoking'])
df['AlcoholDrinking']= label_encoder.fit_transform(df['AlcoholDrinking'])
df['Stroke']= label_encoder.fit_transform(df['Stroke'])
df['DiffWalking']= label_encoder.fit_transform(df['DiffWalking'])
df['Sex']= label_encoder.fit_transform(df['Sex'])
df['AgeCategory']= label_encoder.fit_transform(df['AgeCategory'])
df['Race']= label_encoder.fit_transform(df['Race'])
df['Diabetic']= label_encoder.fit_transform(df['Diabetic'])
df['PhysicalActivity']= label_encoder.fit_transform(df['PhysicalActivity'])
df['GenHealth']= label_encoder.fit_transform(df['GenHealth'])
df['Asthma']= label_encoder.fit_transform(df['Asthma'])
df['KidneyDisease']= label_encoder.fit_transform(df['KidneyDisease'])
df['SkinCancer']= label_encoder.fit_transform(df['SkinCancer'])
```

02

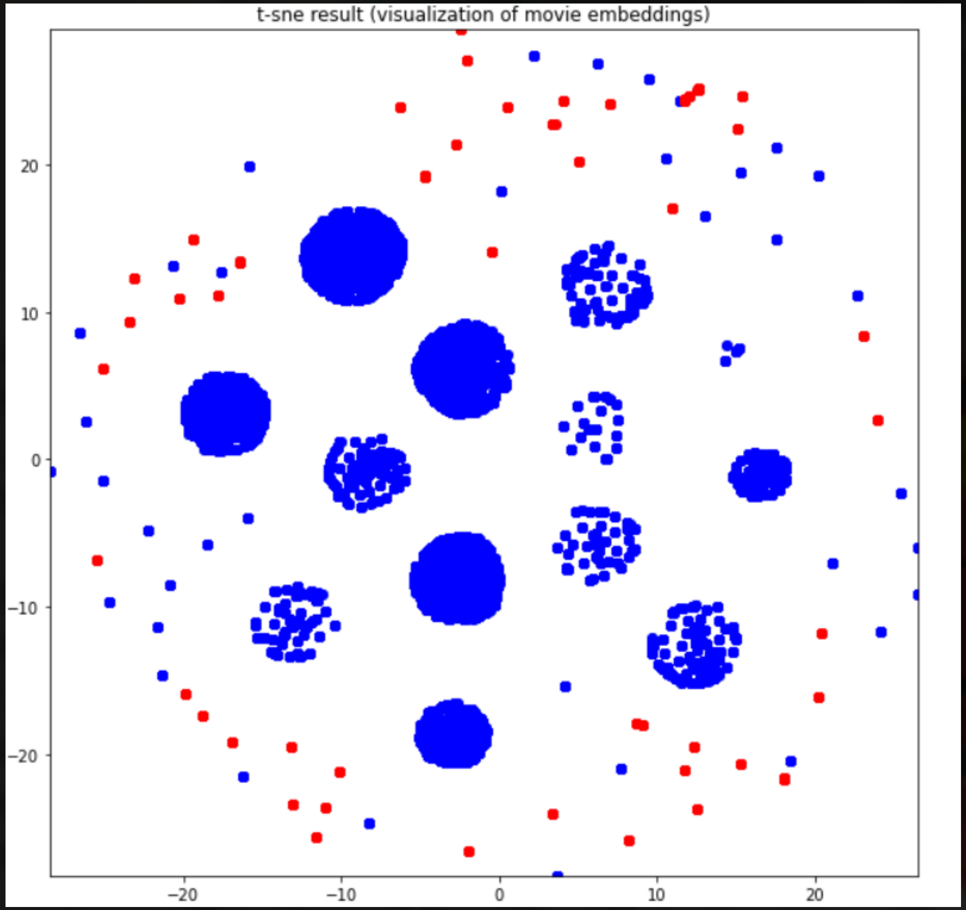
Models

TSNE

(T-Distributed Stochastic Neighbor Embedding)

3 features

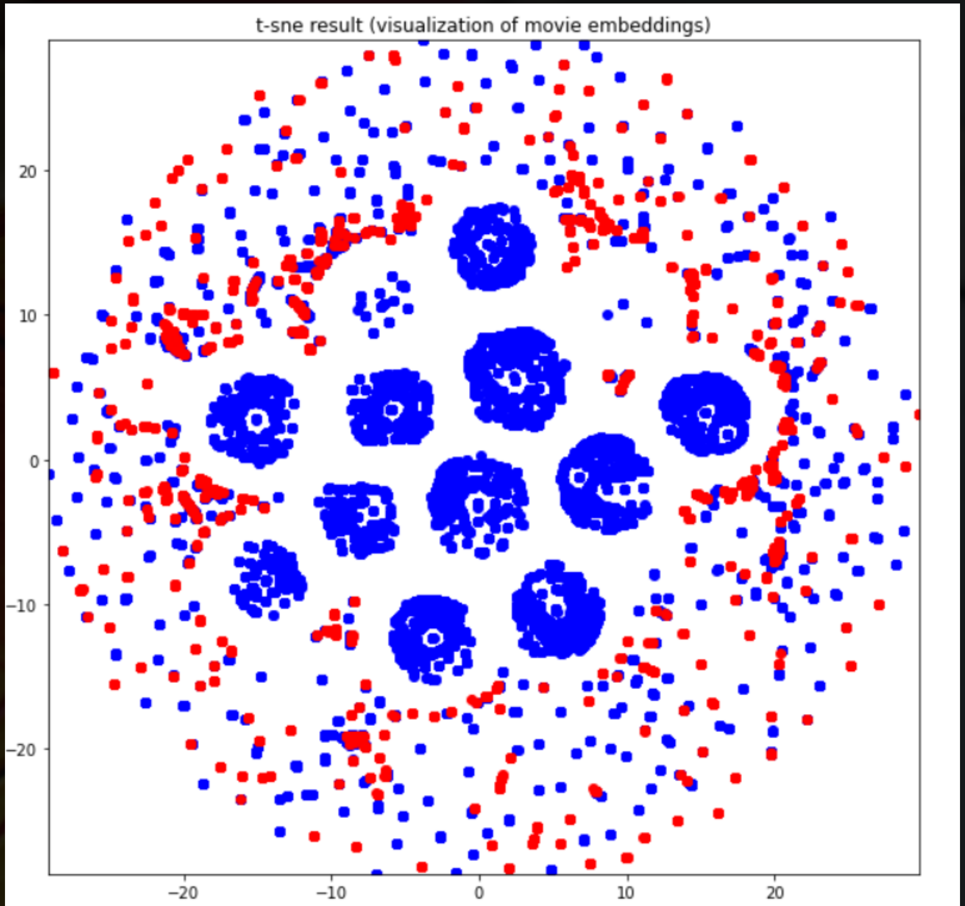
Stroke, DiffWalking, AgeCategory



Label = (1,0)
Label = (0,1)

4 features

Stroke, DiffWalking, AgeCategory, PhysicalHealth



02

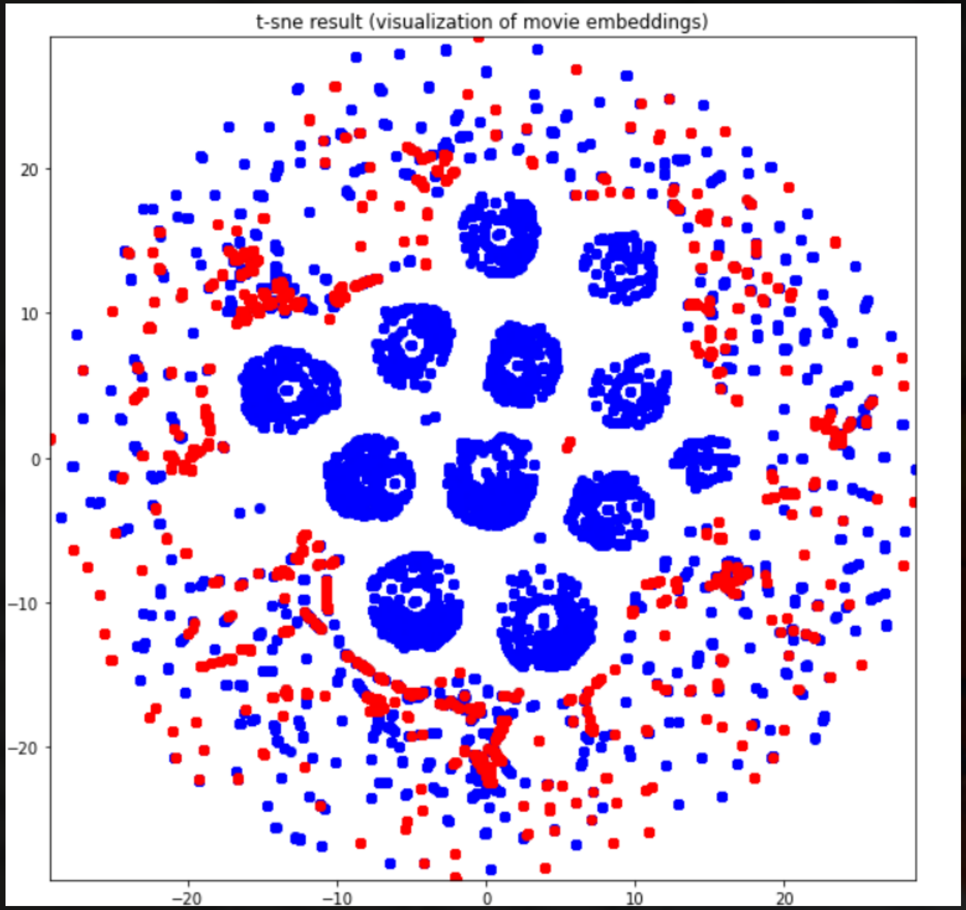
Models

TSNE

(T-Distributed Stochastic Neighbor Embedding)

5 features

Stroke, DiffWalking, AgeCategory, PhysicalHealth, KidneyDisease

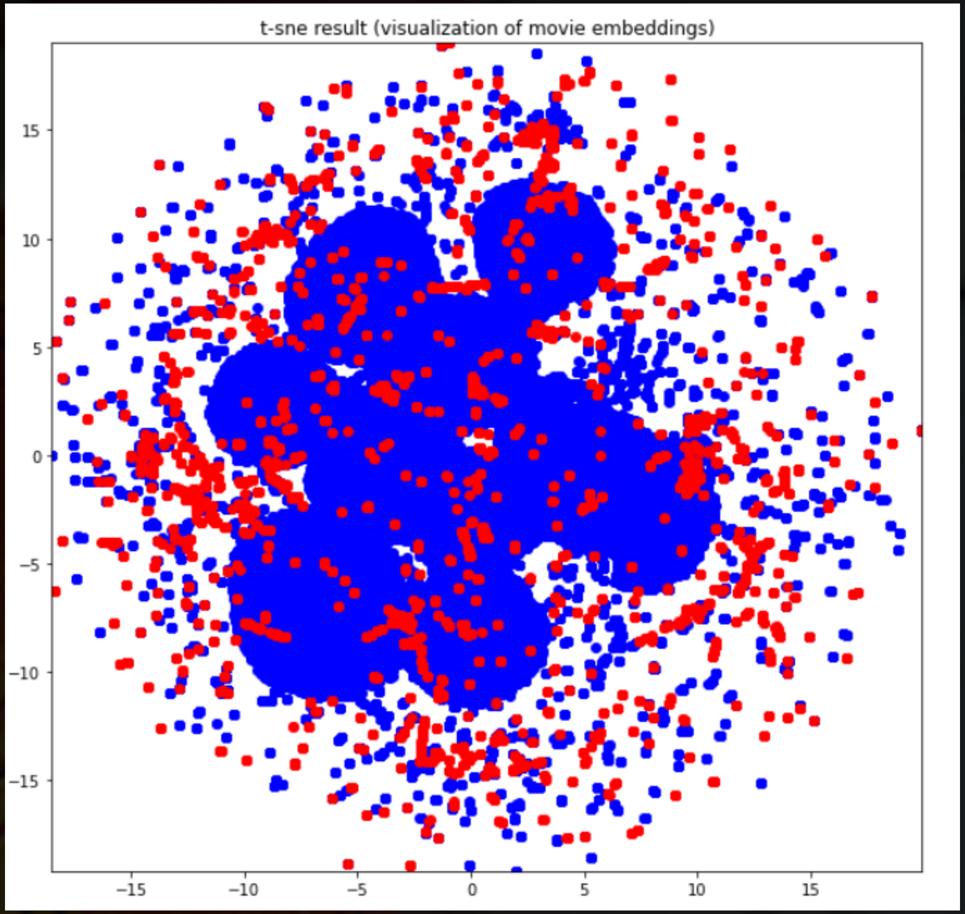


Label = (1,0)

Label = (0,1)

6 features

Stroke, DiffWalking, AgeCategory, PhysicalHealth, KidneyDisease Smoking



02 Models

Logistic Regression

Model & Hyperparameters

```
class BinaryClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(17, 1) #self.linear = {w,b}
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        return self.sigmoid(self.linear(x))

model = BinaryClassifier()

optimizer = optim.SGD(model.parameters(), lr=0.01)

nb_epochs = 100
```

Train Accuracy & Test Accuracy

```
Train Epoch    0/100 Train Cost: 0.408149 Train Accuracy 91.43%
Train Epoch   20/100 Train Cost: 0.355904 Train Accuracy 90.99%
Train Epoch   40/100 Train Cost: 0.329248 Train Accuracy 90.94%
Train Epoch   60/100 Train Cost: 0.312790 Train Accuracy 91.18%
Train Epoch   80/100 Train Cost: 0.302912 Train Accuracy 91.33%
Train Epoch  100/100 Train Cost: 0.296622 Train Accuracy 91.38%
```

```
Test Epoch    0/100 Test Cost: 0.437619 Test Accuracy 89.41%
Test Epoch   20/100 Test Cost: 0.324188 Test Accuracy 91.03%
Test Epoch   40/100 Test Cost: 0.310905 Test Accuracy 91.32%
Test Epoch   60/100 Test Cost: 0.303641 Test Accuracy 91.33%
Test Epoch   80/100 Test Cost: 0.298252 Test Accuracy 91.34%
Test Epoch  100/100 Test Cost: 0.294003 Test Accuracy 91.33%
```

02
Models

Logistic Regression

Weight & bias

```
for name, param in model.named_parameters():
    parm = {}
    parm[name]=param.detach().numpy()
    print(parm)
```

```
{'linear.weight': array([[-0.05709493, -0.1483185 , -0.20287965,  0.0196627 ,  0.04520231,
        -0.00625067,  0.20764819,  0.18666841,  0.15783934, -0.15874131,
        -0.04945571,  0.12815215, -0.07809287, -0.16721553, -0.02663539,
        -0.18812351,  0.15431833]], dtype=float32)}
{'linear.bias': array([-0.15068565], dtype=float32)}
```

```
def practice_exercise(inputs):
    x = 0
    for i in range(len(trained_w)):
        x += trained_w[i]*inputs[i]
    x += trained_b
    return x
```

Test Dataset

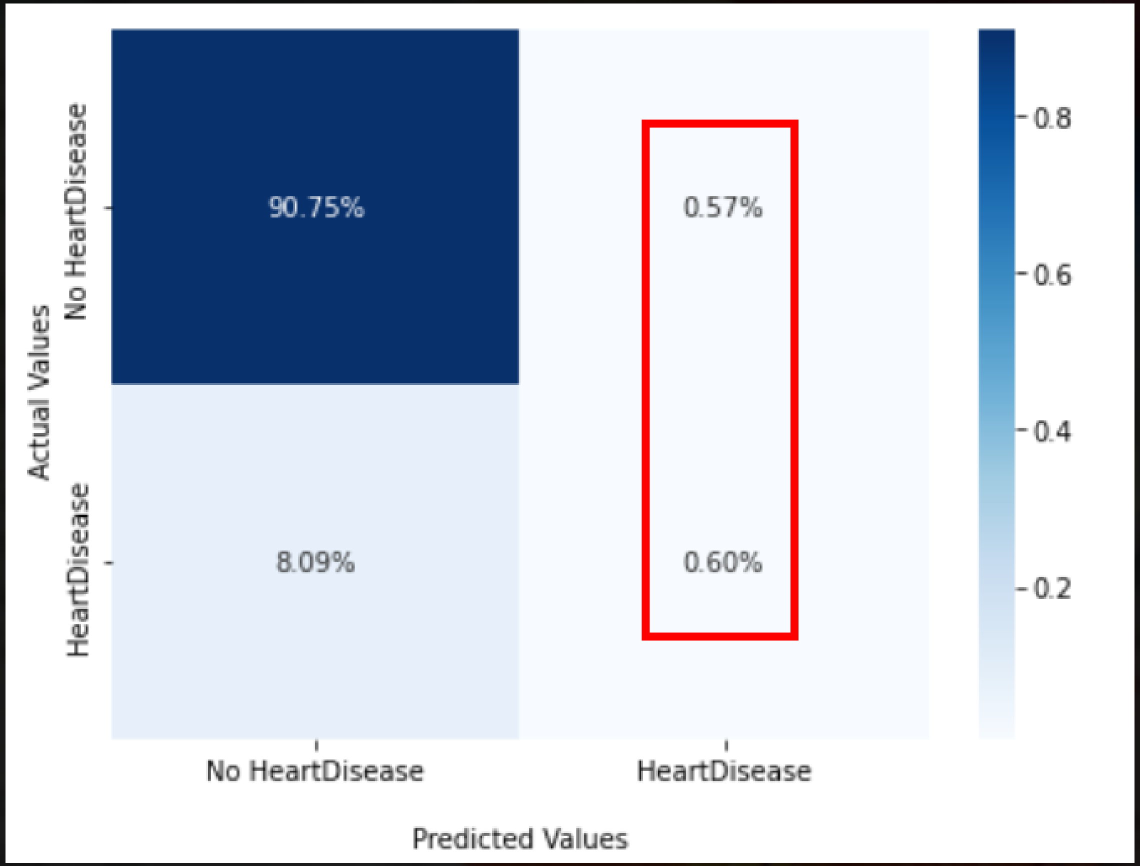
```
outputs          label

[-1.83500796] [0.]
[-3.80138024] [0.]
[-2.44095826] [0.]
[-3.11195034] [0.]
[-1.6935326] [0.]
[-3.16170667] [0.]
[-1.11333294] [0.]
[-2.86128795] [0.]
[-3.32781749] [0.]
[-3.56311156] [0.]
[-2.2868853] [0.]
[-0.81954123] [0.]
[-2.19062338] [1.]
[-2.39412434] [0.]
[-1.44953226] [0.]
[0.05649307] [1.]
[-2.18116314] [1.]
[-2.80931583] [1.]
[-2.61233098] [0.]
[-2.69429381] [0.]
```

02

Models

Logistic Regression

Confusion Matrix



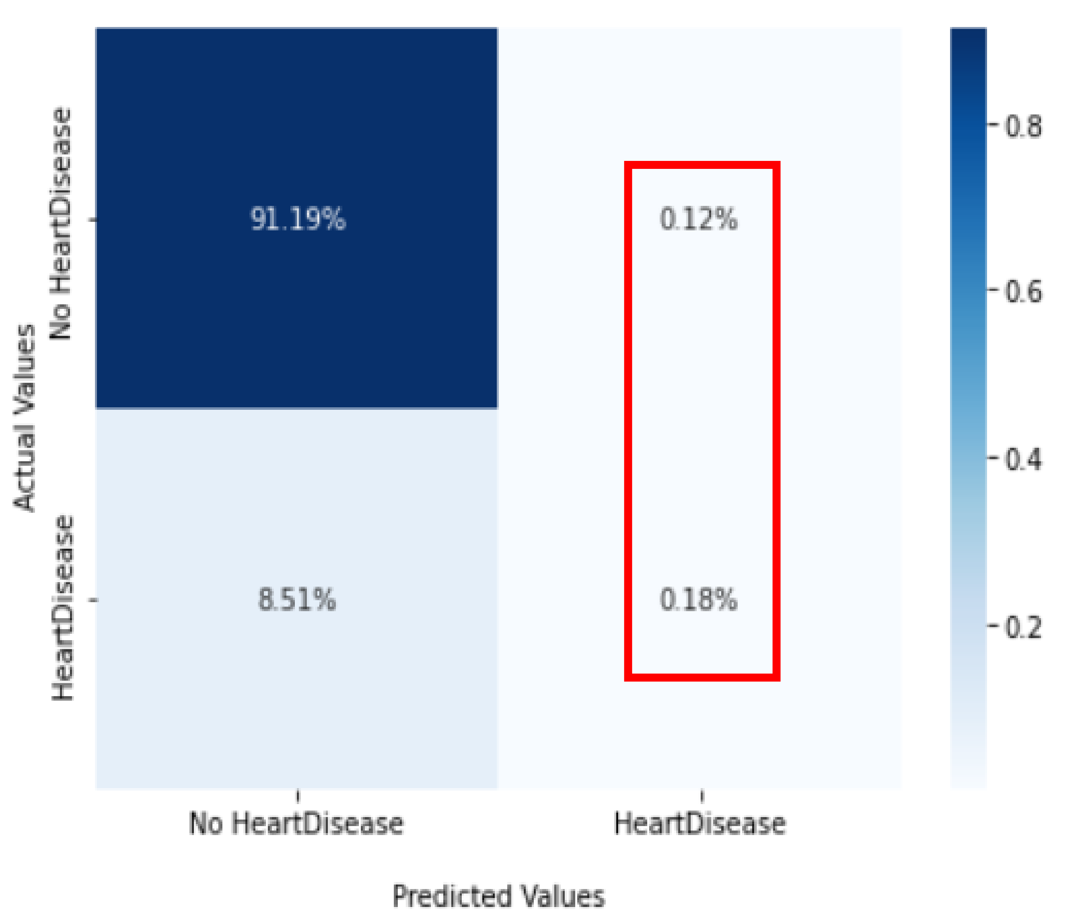
02 Models

SVM
(Support Vector Machines)

Model

```
from sklearn.svm import SVC
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

svmpoly_clf = SVC(kernel="poly", gamma='auto') # Play with degree and C
svmpoly_clf.fit(X_train, y_train)

y_pred_svm=svmpoly_clf.predict(X_test)
cm5=confusion_matrix(y_test,y_pred_svm)

plt.figure(figsize=(7,5))

ax = sns.heatmap(cm5/np.sum(cm5),fmt='.2%', annot=True, cmap='Blues')

ax.set_xlabel('₩nPredicted Values')
ax.set_ylabel('Actual Values ');

ax.xaxis.set_ticklabels(['No HeartDisease','HeartDisease'])
ax.yaxis.set_ticklabels(['No HeartDisease','HeartDisease'])

plt.show()
```

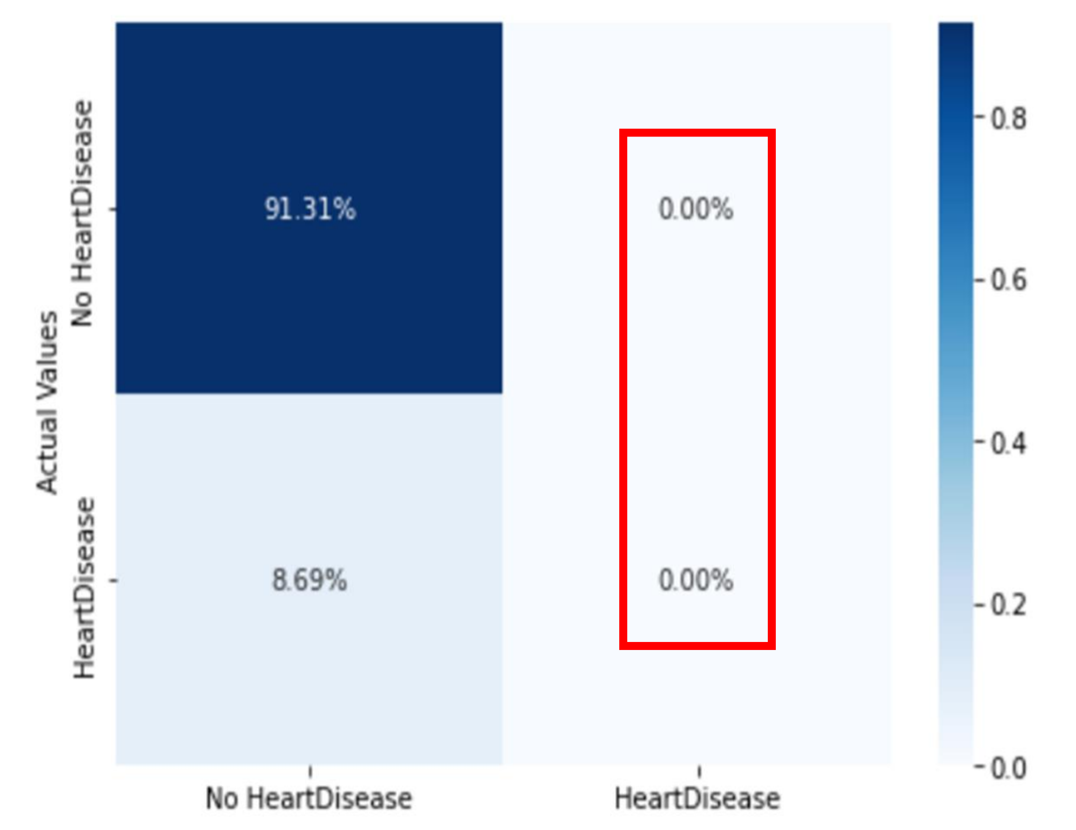
Confusion Matrix



02
Models

SVM
(Support Vector Machines)

Confusion Matrix
(2 & 3 & 4 & 5 features)



02 Models

KNN
(K-Nearest Neighbor)

Model

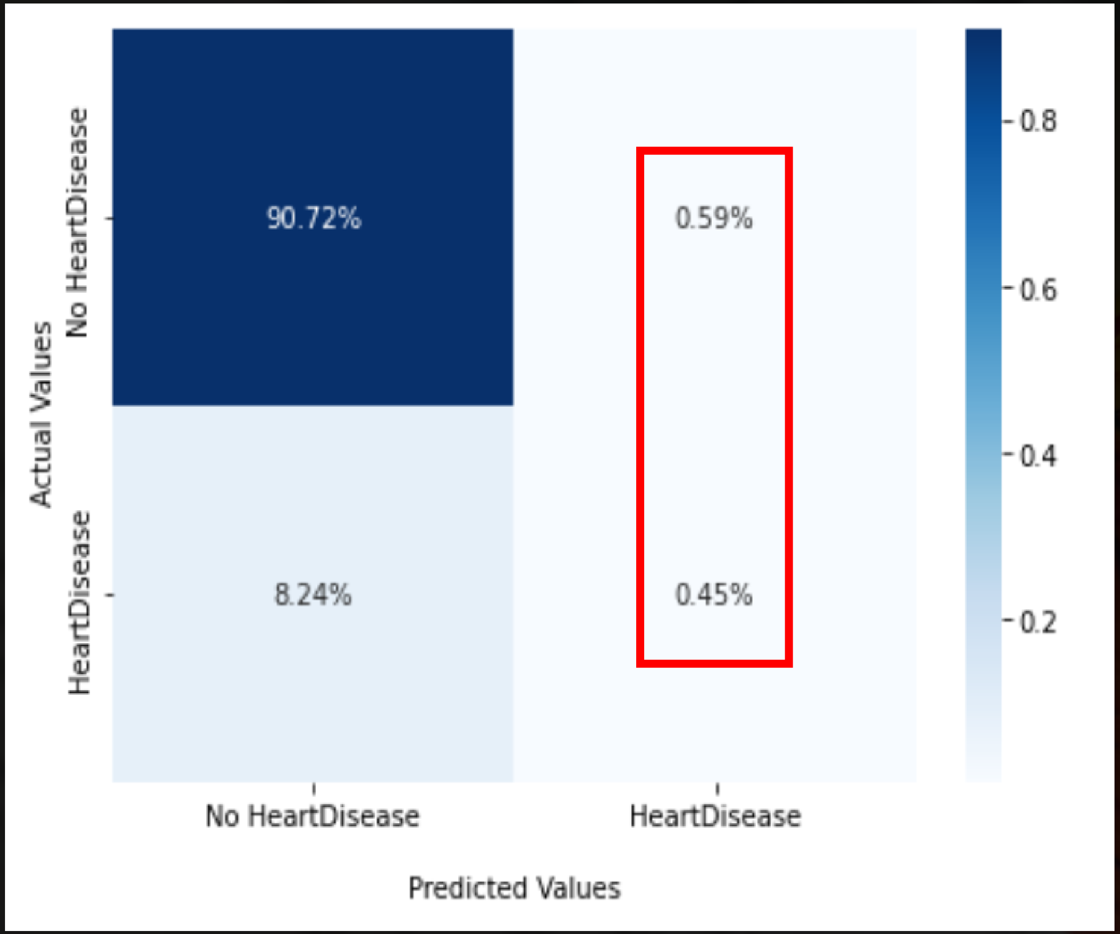
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

def evaluate_model(model, x_test, y_test):
    from sklearn import metrics

    # Predict Test Data
    y_pred = model.predict(x_test)

    # Calculate accuracy, precision, recall, f1-score, and kappa score
    acc = metrics.accuracy_score(y_test, y_pred)
    prec = metrics.precision_score(y_test, y_pred)
    rec = metrics.recall_score(y_test, y_pred)
    f1 = metrics.f1_score(y_test, y_pred)
```

```python
    f1 = metrics.f1_score(y_test, y_pred)
    kappa = metrics.cohen_kappa_score(y_test, y_pred)

    # Calculate area under curve (AUC)
    y_pred_proba = model.predict_proba(x_test)[::,1]
    fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)
    auc = metrics.roc_auc_score(y_test, y_pred_proba)

    # Display confussion matrix
    cm = metrics.confusion_matrix(y_test, y_pred)

    return {'acc': acc, 'prec': prec, 'rec': rec, 'f1': f1, 'kappa': ka
            'fpr': fpr, 'tpr': tpr, 'auc': auc, 'cm': cm}
knn_eval = evaluate_model(knn, X_test, y_test)
```

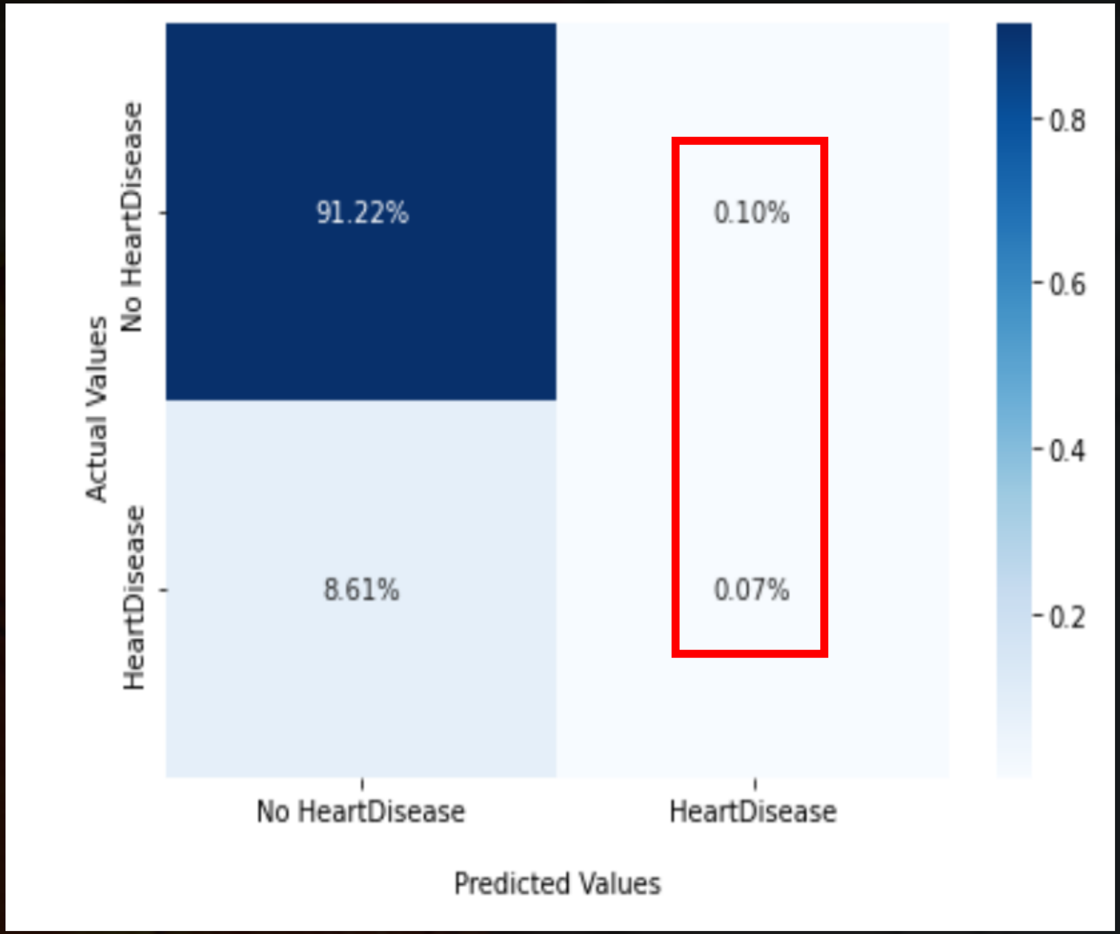
02
Models

KNN
(K-Nearest Neighbor)

Confusion Matrix
(17 features)



Confusion Matrix
(3 features)



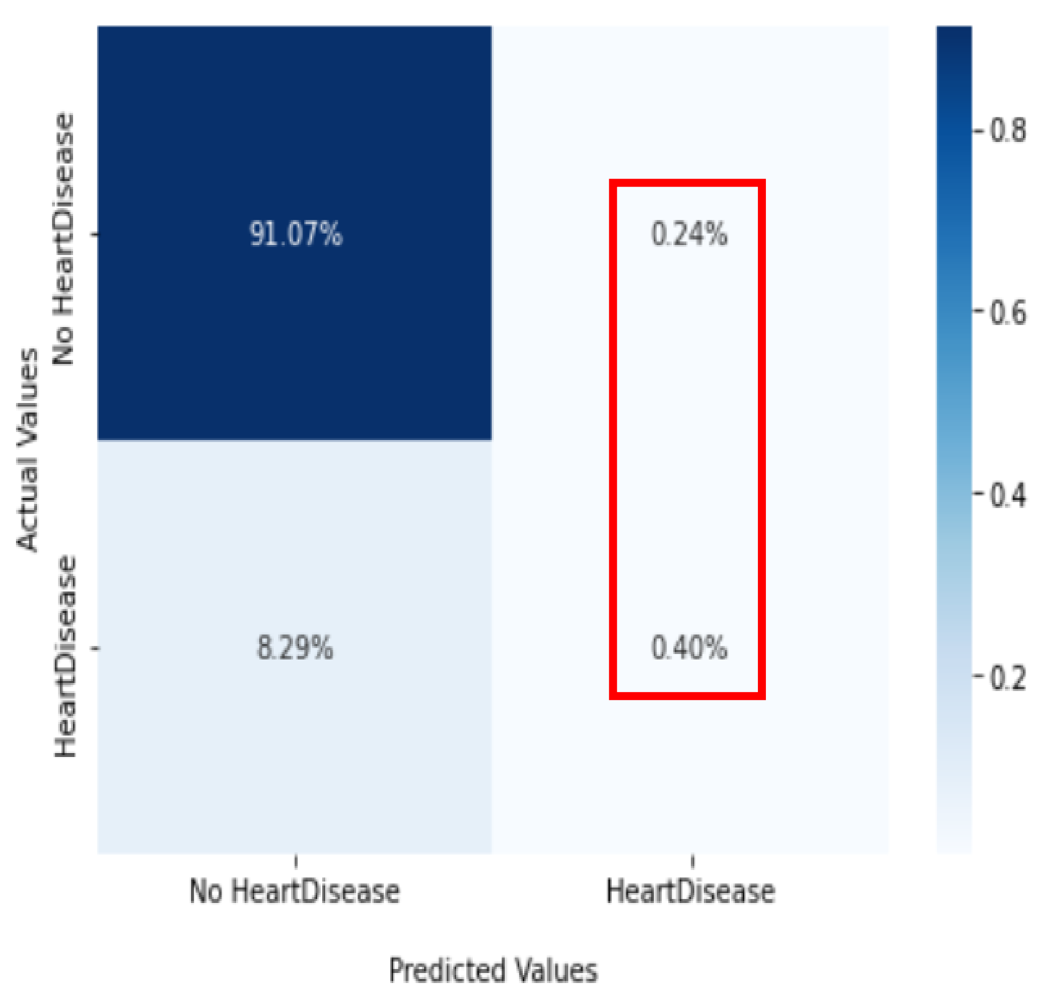
02 Models

ANN
(Artificial Neural Network)

Model

```
model = keras.Sequential([
    # 1
    layers.Dense(17, activation='relu'),
    # 2
    layers.Dense(17, activation='relu'),

    ...

    # 12
    layers.Dense(2, activation='relu')])

model.compile(
    optimizer='SGD',
    loss='binary_crossentropy',
    metrics=['binary_accuracy'])

model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    batch_size=64,
    epochs=10)
```

Confusion Matrix



: Contents

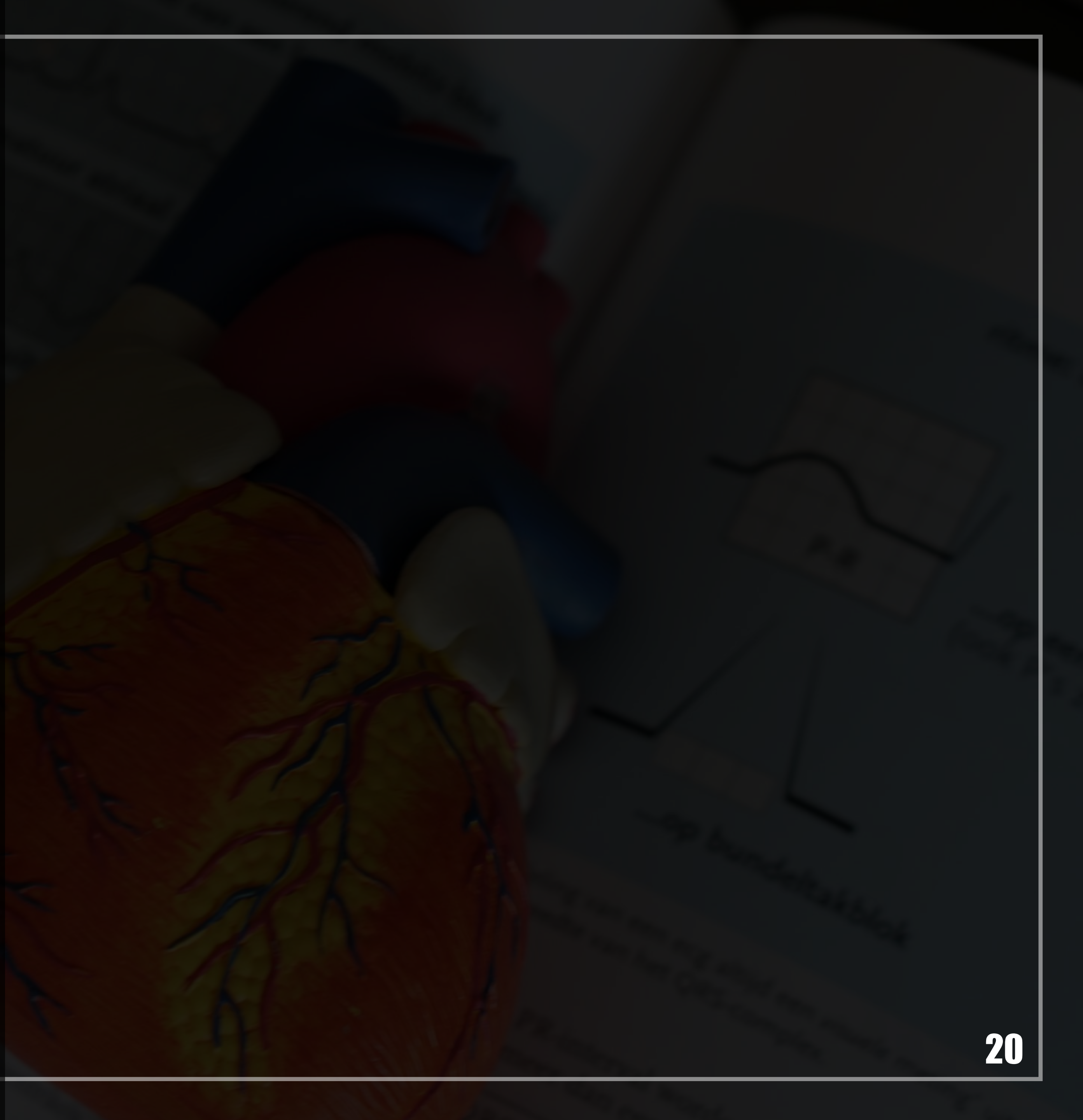
| 1st Dataset

| 2nd Models

| **3rd Selected Model**

| 4th Application for RecoSys

| 5th Conclusion & Limitations



03

Selected Model

ANN
(Artificial Neural Network)

Prepare Data

```
from torch.utils.data import TensorDataset, DataLoader, random_split

input_tensor = torch.from_numpy(input_np.astype(np.float32))
label_tensor = torch.from_numpy(label_np.astype(np.int64))

dataset = TensorDataset(input_tensor, label_tensor)

train_len = int(len(dataset) * 0.7)
test_len = int(len(dataset) - train_len)


(train_len, test_len)
```

```
(223856, 95939)
```

```
train_dataset, test_dataset = random_split(dataset, [train_len, test_len])


train_loader = DataLoader(train_dataset, batch_size=64, num_workers=2, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, num_workers=2, shuffle=True)
```

03

Selected Model

ANN

(Artificial Neural Network)

Selected Model

```python
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.fc1 = nn.Linear(17,17)
        self.fc2 = nn.Linear(17,2)
        self.relu = nn.ReLU()

    def forward(self, x):
        #1
        x = self.fc1(x)
        x = self.relu(x)


        ...


        #12
        x = self.fc2(x)
        x = self.relu(x)
```

12 Hidden Layer
Fully Connected
ReLU Activation Function

Model with B.N. & Dropout

```python
class Model_bn_do(nn.Module):
    def __init__(self):
        super(Model_bn_do, self).__init__()
        self.fc1 = nn.Linear(17,17)
        self.fc2 = nn.Linear(17,2)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.8)
        self.bn1 = nn.BatchNorm1d(17)

    def forward(self, x):
        #1
        x = self.dropout(x)
        x = self.fc1(x)
        x = self.bn1(x)
        x = self.relu(x)

        ...
```

12 Hidden Layer
Dropout 20%
Batch Normalization

03

Selected Model

ANN
(Artificial Neural Network)
Function for Training

```
def training_epoch(train_loader, network, loss_func, optimizer, epoch):
    train_losses = []
    train_correct = 0
    log_interval = 500

    for batch_idx, (inputs, label) in enumerate(train_loader):
        optimizer.zero_grad()

        batch_size = inputs.size()[0]
        inputs = inputs.view(-1, 17)

        outputs = network(inputs)

        loss = loss_func(outputs, label)
        train_losses.append(loss.item())

        pred = torch.max(outputs,1)[1]
        train_correct += pred.eq(label).sum()

        loss.backward()

        optimizer.step()

        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]₩tLoss: {:.6f}'
                  .format(epoch, batch_idx * len(label), len(train_loader.dataset),100. * batch_idx / len(train_loader),
                          loss.item()))

    return train_losses, train_correct
```

03

Selected Model

ANN

(Artificial Neural Network)

Function for Test

```
def test_epoch(test_loader, network, loss_func):
    correct = 0

    test_losses = []

    with torch.no_grad():
        for batch_idx, (inputs, label) in enumerate(test_loader):
            batch_size = inputs.size()[0]
            inputs = inputs.view(-1, 17)

            outputs = network(inputs)

            loss = loss_func(outputs, label)
            test_losses.append(loss.item())

            pred = torch.max(outputs,1)[1]
            correct += pred.eq(label).sum()

        test_accuracy = 100. * correct / len(test_loader.dataset)

        print('Test set: Accuracy: {}/{} ({:.0f}%)₩n'
              .format(correct, len(test_loader.dataset),100. * correct / len(test_loader.dataset)))
    return test_losses, test_accuracy
```

03

Selected Model

ANN
(Artificial Neural Network)

Function for Save Model

```
def saveModel():
    path = "C:/Users/JinnyeongHeo/Desktop/2022-1/추천시스템/HeartDisease_model/NetModel.pth"
    torch.save(network.state_dict(), path)
```

03
Selected Model

ANN
(Artificial Neural Network)
Training

```python
def training(network, learning_rate = 0.0519):

    best_accuracy = 0.0

    epoches = 10

    cls_loss = nn.CrossEntropyLoss()
    optimizer = optim.SGD(network.parameters(), lr = learning_rate)

    train_losses_per_epoch = []
    test_losses_per_epoch = []

    train_accuracies = []
    test_accuracies = []


    for epoch in range(epoches):

        network.train()

        train_losses, train_correct = training_epoch(train_loader,network,cls_loss,optimizer, epoch)

        average_loss = np.mean(train_losses)
        train_losses_per_epoch.append(average_loss)

        train_accuracy = train_correct / len(train_loader.dataset) * 100
        train_accuracies.append(train_accuracy)

        print('₩nTraining set: Accuracy: {}/{} ({:.0f}%)'
              .format(train_correct, len(train_loader.dataset),100. * train_correct / len(train_loader.dataset)))


        network.eval()
```

03

Selected Model

ANN
(Artificial Neural Network)

Test & Save

```
    correct = 0
    with torch.no_grad():
        test_losses, test_accuracy = test_epoch(test_loader, network, cls_loss)

    test_losses_per_epoch.append(np.mean(test_losses))
    test_accuracies.append(test_accuracy)

    if test_accuracy > best_accuracy:
        saveModel()
        best_accuracy = test_accuracy

return train_losses_per_epoch, test_losses_per_epoch, train_accuracies, test_accuracies
```

03
Selected Model

ANN
(Artificial Neural Network)

Accuracy of Selected Model

```
Train Epoch: 0 [0/223856 (0%)]  Loss: 0.662906
Train Epoch: 0 [32000/223856 (14%)]     Loss: 0.200548
Train Epoch: 0 [64000/223856 (29%)]     Loss: 0.387938
Train Epoch: 0 [96000/223856 (43%)]     Loss: 0.311434
Train Epoch: 0 [128000/223856 (57%)]    Loss: 0.274645
Train Epoch: 0 [160000/223856 (71%)]    Loss: 0.236688
Train Epoch: 0 [192000/223856 (86%)]    Loss: 0.199064

Training set: Accuracy: 204669/223856 (91%)
Test set: Accuracy: 87753/95939 (91%)

...

Train Epoch: 9 [0/223856 (0%)]  Loss: 0.159966
Train Epoch: 9 [32000/223856 (14%)]     Loss: 0.281964
Train Epoch: 9 [64000/223856 (29%)]     Loss: 0.242223
Train Epoch: 9 [96000/223856 (43%)]     Loss: 0.214790
Train Epoch: 9 [128000/223856 (57%)]    Loss: 0.179357
Train Epoch: 9 [160000/223856 (71%)]    Loss: 0.197916
Train Epoch: 9 [192000/223856 (86%)]    Loss: 0.182939

Training set: Accuracy: 204731/223856 (91%)
Test set: Accuracy: 87789/95939 (92%)
```

Accuracy of Model with B.N. & Dropout

```
Train Epoch: 0 [0/223856 (0%)]  Loss: 0.685913
Train Epoch: 0 [32000/223856 (14%)]     Loss: 0.271475
Train Epoch: 0 [64000/223856 (29%)]     Loss: 0.317150
Train Epoch: 0 [96000/223856 (43%)]     Loss: 0.357402
Train Epoch: 0 [128000/223856 (57%)]    Loss: 0.235426
Train Epoch: 0 [160000/223856 (71%)]    Loss: 0.350099
Train Epoch: 0 [192000/223856 (86%)]    Loss: 0.381706

Training set: Accuracy: 204503/223856 (91%)
Test set: Accuracy: 87753/95939 (91%)

...

Train Epoch: 9 [0/223856 (0%)]  Loss: 0.273268
Train Epoch: 9 [32000/223856 (14%)]     Loss: 0.202772
Train Epoch: 9 [64000/223856 (29%)]     Loss: 0.239401
Train Epoch: 9 [96000/223856 (43%)]     Loss: 0.310069
Train Epoch: 9 [128000/223856 (57%)]    Loss: 0.344180
Train Epoch: 9 [160000/223856 (71%)]    Loss: 0.204027
Train Epoch: 9 [192000/223856 (86%)]    Loss: 0.340818

Training set: Accuracy: 204669/223856 (91%)
Test set: Accuracy: 87753/95939 (91%)
```

03

Selected Model

ANN

(Artificial Neural Network)

Initialization

```
def init_xavier(m):
    if isinstance(m, nn.Linear):
        nn.init.xavier_uniform_(m.weight)

def init_kaiming(m):
    if isinstance(m, nn.Linear):
        nn.init.kaiming_uniform_(m.weight)
network_xavier = Model()
network_xavier.apply(init_xavier)

network_kaiming = Model()
network_kaiming.apply(init_kaiming)
```

Accuracy of Xavier & Kaiming

```
Train Epoch: 9 [0/223856 (0%)]  Loss: 0.085851
Train Epoch: 9 [32000/223856 (14%)]    Loss: 0.363344
Train Epoch: 9 [64000/223856 (29%)]    Loss: 0.260326
Train Epoch: 9 [96000/223856 (43%)]    Loss: 0.237395
Train Epoch: 9 [128000/223856 (57%)]   Loss: 0.232701
Train Epoch: 9 [160000/223856 (71%)]   Loss: 0.314334
Train Epoch: 9 [192000/223856 (86%)]   Loss: 0.219238

Training set: Accuracy: 204667/223856 (91%)
Test set: Accuracy: 87753/95939 (91%)
```

```
Train Epoch: 9 [0/223856 (0%)]  Loss: 0.273268
Train Epoch: 9 [32000/223856 (14%)]    Loss: 0.202772
Train Epoch: 9 [64000/223856 (29%)]    Loss: 0.239401
Train Epoch: 9 [96000/223856 (43%)]    Loss: 0.310069
Train Epoch: 9 [128000/223856 (57%)]   Loss: 0.344180
Train Epoch: 9 [160000/223856 (71%)]   Loss: 0.204027
Train Epoch: 9 [192000/223856 (86%)]   Loss: 0.340818

Training set: Accuracy: 204669/223856 (91%)
Test set: Accuracy: 87753/95939 (91%)
```

03

Selected Model

ANN

(Artificial Neural Network)

Selected Model

```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.fc1 = nn.Linear(17,17)
        self.fc2 = nn.Linear(17,2)
        self.relu = nn.ReLU()

    def forward(self, x):
        #1
        x = self.fc1(x)
        x = self.relu(x)


        ...


        #12
        x = self.fc2(x)
        x = self.relu(x)
```

12 Hidden Layer
Fully Connected
ReLU Activation Function

: Contents

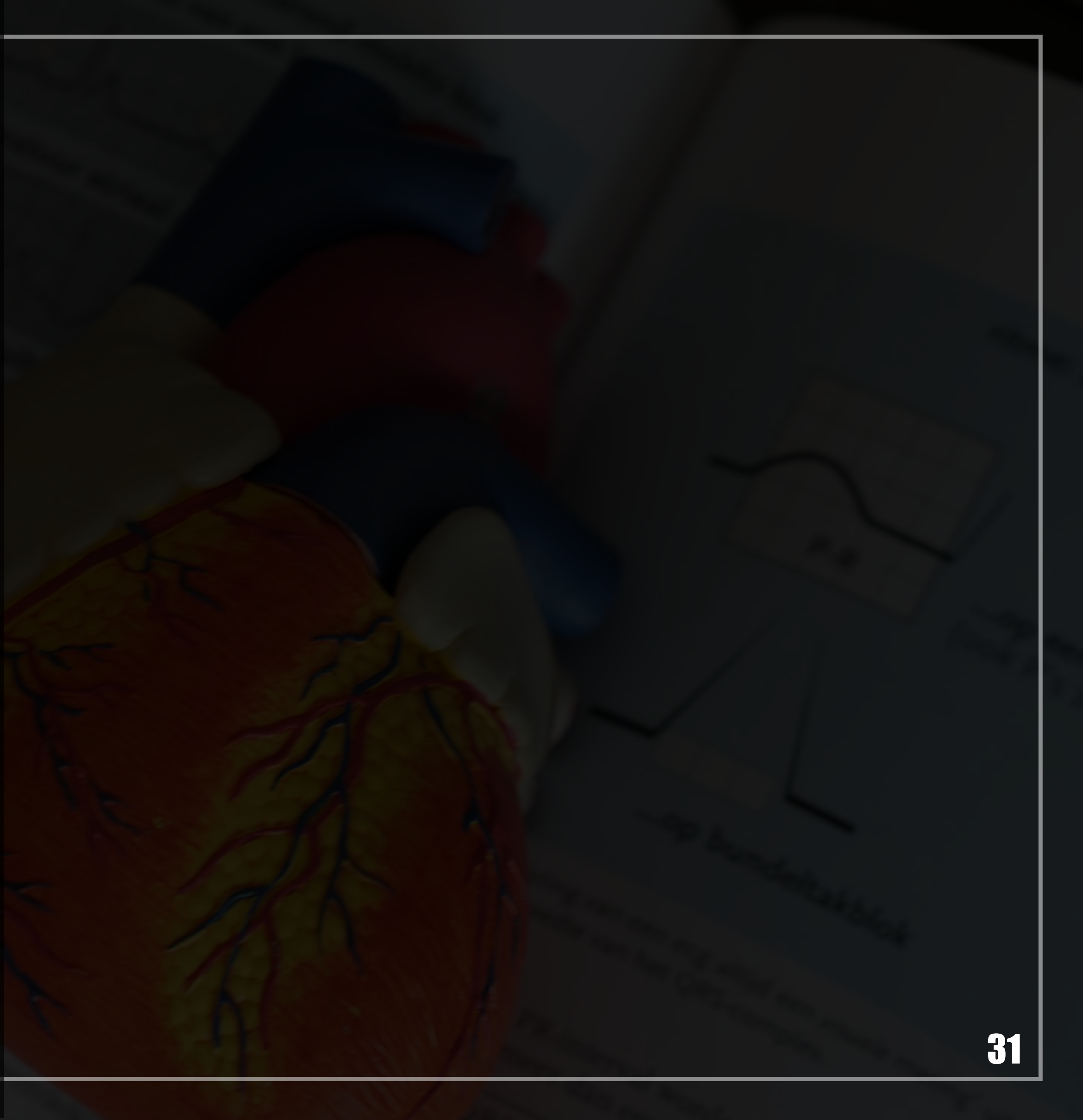
| 1st Dataset

| 2nd Models

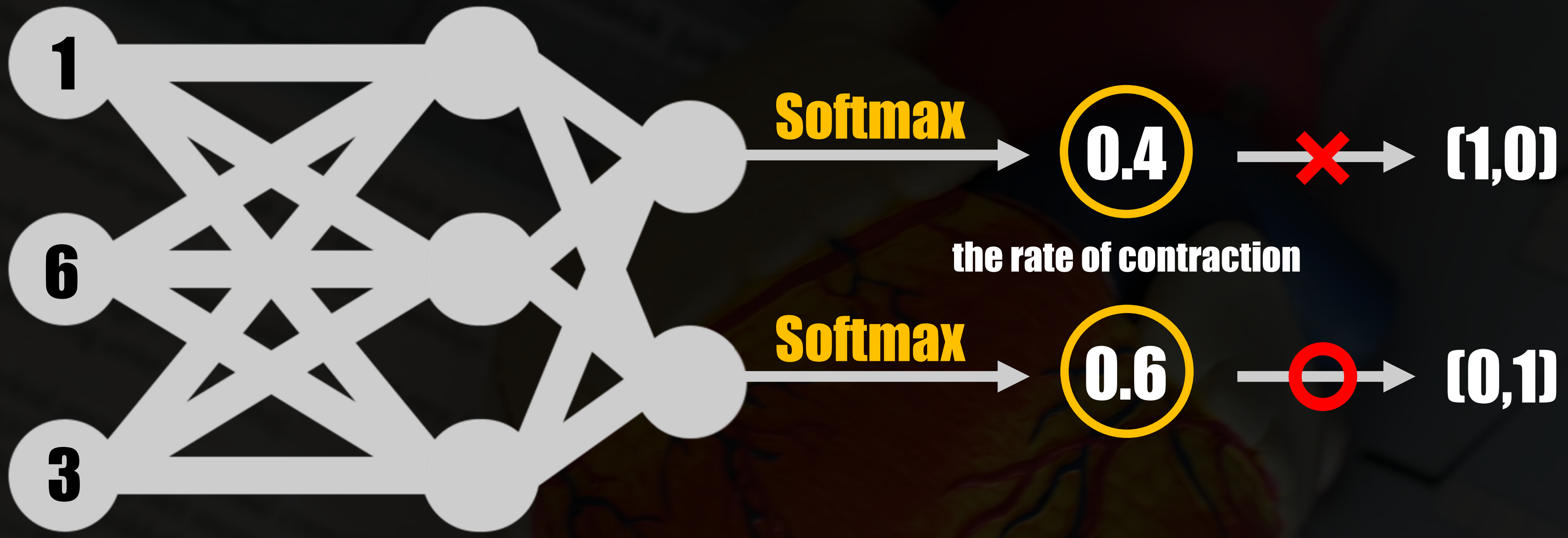
| 3rd Selected Model

| 4th Application for RecoSys

| 5th Conclusion & Limitations

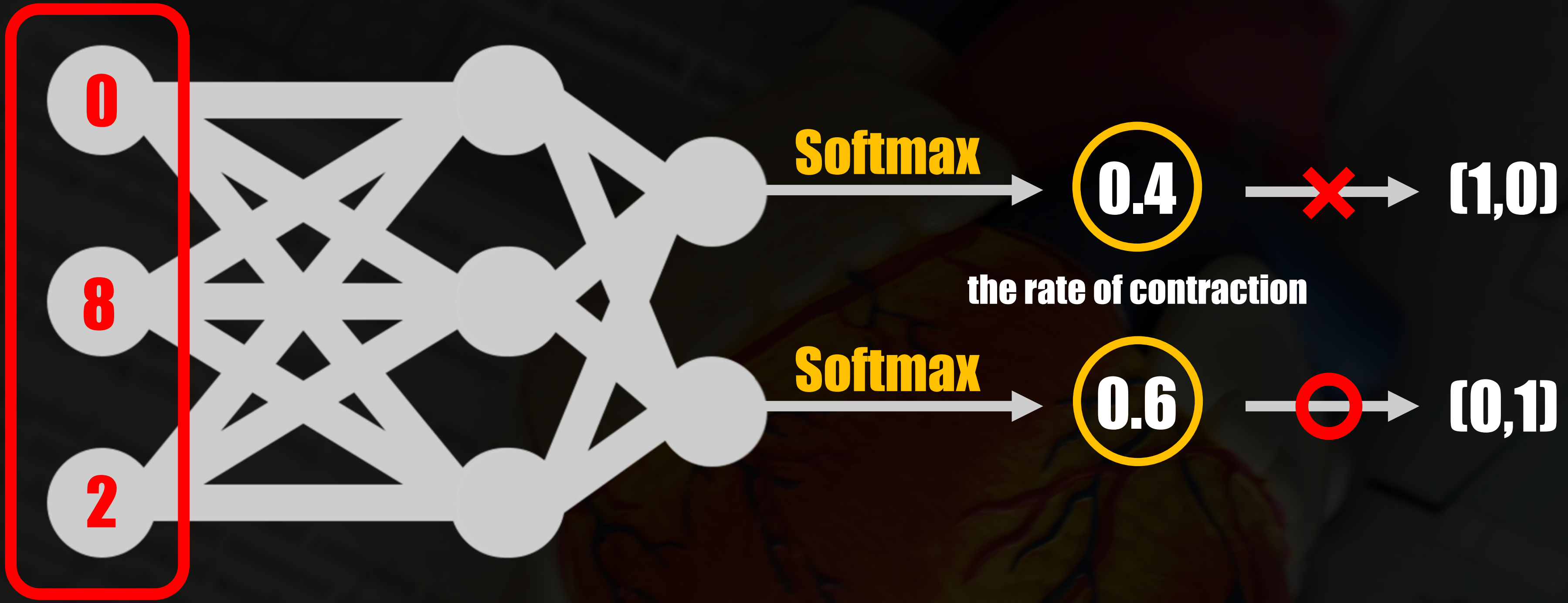


Softmax



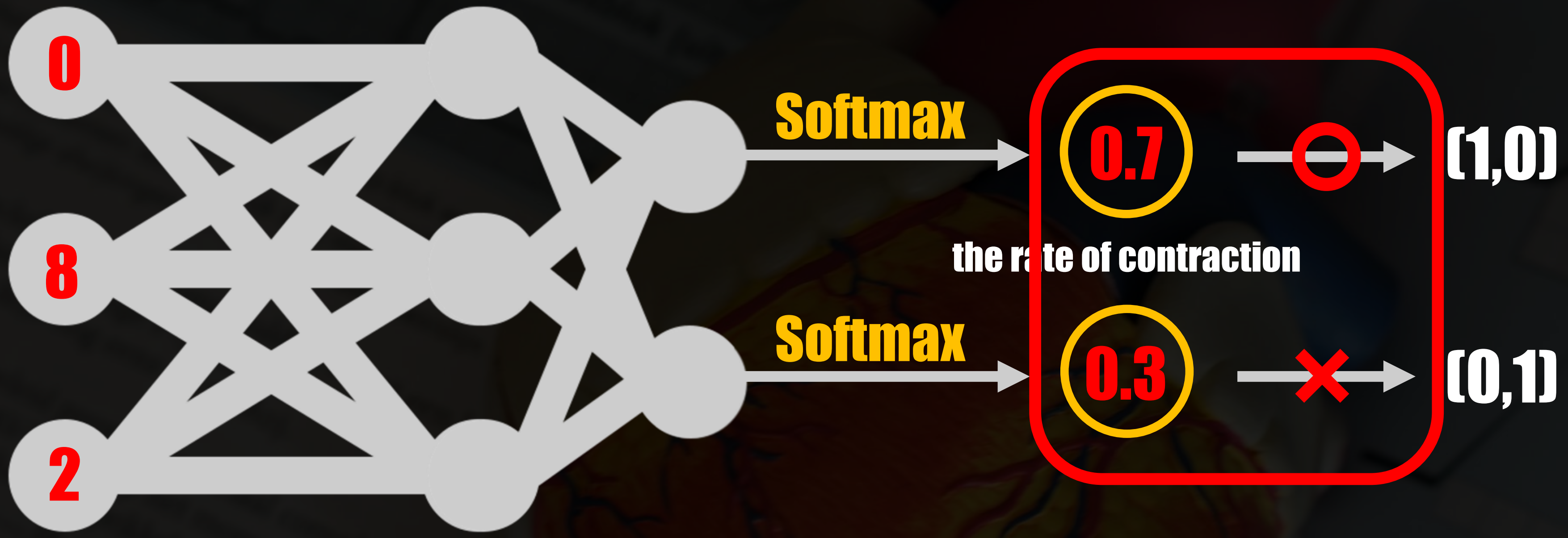
The result of softmax is **the rate of contraction**
Apply this to RecSys

Softmax



The result of softmax is the rate of contraction
Apply this to RecSys

Softmax



The rate of contraction can be reduced by Manipulating feature values (input)

Softmax

Function for Softmax

```
def practice_exercise(inputs, network):
    with torch.no_grad():
        outputs = network(inputs)
        softmax_f = nn.Softmax(dim=1)
        pred_hd = softmax_f(outputs)
        print(pred_hd)
    return pred_hd
```

04
Application for RecoSys

Softmax

Load the best parameters

```
network1 = Model2()
PATH = "C:/Users/JinnyeongHeo/Desktop/2022-1/추천시스템/HeartDisease_model/NetModel.pth"
network1.load_state_dict(torch.load(PATH))
```

Data about people who have heart Disease

```
c = torch.Tensor(input_data_hdy[10000:10020].values)
practice_exercise(c,network1)
```

```
tensor([[26.5800,  1.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  1.0000,
         11.0000,  5.0000,  2.0000,  1.0000,  4.0000,  8.0000,  0.0000,  0.0000,
          0.0000],
        [26.5200,  1.0000,  0.0000,  0.0000,  6.0000,  5.0000,  0.0000,  0.0000,
         10.0000,  5.0000,  0.0000,  1.0000,  1.0000,  6.0000,  1.0000,  0.0000,
          0.0000],
        [32.9300,  0.0000,  0.0000,  0.0000,  1.0000,  1.0000,  0.0000,  1.0000,
         10.0000,  5.0000,  1.0000,  1.0000,  2.0000,  7.0000,  0.0000,  0.0000,
          1.0000],
        [32.5500,  1.0000,  0.0000,  0.0000, 15.0000, 10.0000,  0.0000,  1.0000,
         11.0000,  5.0000,  0.0000,  1.0000,  3.0000,  8.0000,  0.0000,  0.0000,
          1.0000],
        [23.6500,  1.0000,  0.0000,  0.0000, 30.0000,  5.0000,  1.0000,  1.0000,
         10.0000,  5.0000,  2.0000,  0.0000,  3.0000,  8.0000,  0.0000,  0.0000,
          1.0000],
        [40.6900,  0.0000,  0.0000,  0.0000, 30.0000, 30.0000,  1.0000,  1.0000,
          7.0000,  5.0000,  2.0000,  1.0000,  3.0000,  6.0000,  0.0000,  0.0000,
          0.0000],
```

Data about people who don't have heart Disease

```
d = torch.Tensor(input_data_hdn[10000:10020].values)
practice_exercise(d,network1)
```

```
tensor([[23.3000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
          9.0000,  5.0000,  0.0000,  1.0000,  0.0000,  8.0000,  0.0000,  0.0000,
          0.0000],
        [19.4000,  1.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
          9.0000,  5.0000,  0.0000,  1.0000,  4.0000,  8.0000,  0.0000,  0.0000,
          1.0000],
        [36.0500,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,
          4.0000,  0.0000,  0.0000,  1.0000,  0.0000,  8.0000,  0.0000,  0.0000,
          0.0000],
        [21.7000,  0.0000,  0.0000,  0.0000,  0.0000, 30.0000,  0.0000,  1.0000,
          1.0000,  5.0000,  0.0000,  1.0000,  4.0000,  8.0000,  0.0000,  0.0000,
          0.0000],
        [35.4400,  0.0000,  0.0000,  0.0000,  0.0000,  0.0000,  1.0000,  0.0000,
          8.0000,  3.0000,  1.0000,  0.0000,  1.0000,  5.0000,  1.0000,  0.0000,
          0.0000],
        [28.2500,  0.0000,  0.0000,  0.0000,  0.0000,  5.0000,  0.0000,  1.0000,
          8.0000,  5.0000,  0.0000,  1.0000,  4.0000,  7.0000,  0.0000,  0.0000,
          0.0000],
```

Softmax

Softmax about people who have heart Disease

```
tensor([[0.7487, 0.2513],
        [0.7856, 0.2144],
        [0.7888, 0.2112],
        [0.7420, 0.2580],
        [0.5819, 0.4181],
        [0.8313, 0.1687],
        [0.6979, 0.3021],
        [0.7180, 0.2820],
        [0.9040, 0.0960],
        [0.5568, 0.4432],
        [0.9545, 0.0455],
        [0.8314, 0.1686],
        [0.5858, 0.4142],
        [0.9565, 0.0435],
        [0.7709, 0.2291],
        [0.9540, 0.0460],
        [0.8081, 0.1919],
        [0.9665, 0.0335],
        [0.7864, 0.2136],
        [0.8232, 0.1768]])
```

tmp1

Mean = 0.2104

Softmax about people who don't have heart Disease

```
tensor([[0.9744, 0.0257]
        [0.9574, 0.0426]
        [0.9739, 0.0261]
        [0.9965, 0.0035]
        [0.8071, 0.1929]
        [0.9595, 0.0405]
        [0.9618, 0.0382]
        [0.9882, 0.0118]
        [0.8761, 0.1239]
        [0.9933, 0.0067]
        [0.9759, 0.0241]
        [0.9945, 0.0055]
        [0.9079, 0.0921]
        [0.9759, 0.0241]
        [0.9750, 0.0250]
        [0.9668, 0.0332]
        [0.9568, 0.0432]
        [0.9623, 0.0377]
        [0.9919, 0.0081]
        [0.9269, 0.0731] )
```

Mean = 0.0439

04

Application for RecoSys

Softmax

Data of tmp1

```
tensor([[23.6500,  1.0000,  0.0000,  0.0000, 30.0000,  5.0000,  1.0000,  1.0000,
         10.0000,  5.0000,  2.0000,  0.0000,  3.0000,  8.0000,  0.0000,  0.0000,
          1.0000]])
tensor([[0.5819, 0.4181]])
```

'BMI', 'Smoking', 'AlcoholDrinking', 'Stroke', 'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'Sex', 'AgeCategory', 'Race', 'Diabetic', 'PhysicalActivity', 'GenHealth', 'SleepTime', 'Asthma', 'KidneyDisease', 'SkinCancer'

Manipulate Data of tmp1

```
tensor([[23.6500,  1.0000,  0.0000,  0.0000, 30.0000,  5.0000,  0.0000,  1.0000,
         10.0000,  5.0000,  2.0000,  0.0000,  3.0000,  8.0000,  0.0000,  0.0000,
          1.0000]])
tensor([[0.6616, 0.3384]])
```

Softmax

Data of tmp1

```
tensor([[23.6500,  1.0000,  0.0000,  0.0000, 30.0000,  5.0000,  1.0000,  1.0000,
         10.0000,  5.0000,  2.0000,  0.0000,  3.0000,  8.0000,  0.0000,  0.0000,
          1.0000]])
tensor([[0.5819, 0.4181]])
```

'BMI', 'Smoking', 'AlcoholDrinking', 'Stroke', 'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'Sex', 'AgeCategory', 'Race', 'Diabetic', 'PhysicalActivity', 'GenHealth', 'SleepTime', 'Asthma', 'KidneyDisease', 'SkinCancer'

Manipulate Data of tmp1

```
tensor([[23.6500,  1.0000,  0.0000,  0.0000, 30.0000,  5.0000,  0.0000,  1.0000,
         10.0000,  5.0000,  2.0000,  0.0000,  3.0000,  8.0000,  0.0000,  0.0000,
          1.0000]])
tensor([[0.6616, 0.3384]])
```

Incidence reduced by 0.0797

Softmax

Data of tmp1

```
tensor([[23.6500,  1.0000,  0.0000,  0.0000, 30.0000,  5.0000,  0.0000,  1.0000,
         10.0000,  5.0000,  2.0000,  0.0000,  3.0000,  8.0000,  0.0000,  0.0000,
          1.0000]])
tensor([[0.6616, 0.3384]])
```

'BMI', 'Smoking', 'AlcoholDrinking', 'Stroke', 'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'Sex', 'AgeCategory', 'Race', 'Diabetic', 'PhysicalActivity', 'GenHealth', 'SleepTime', 'Asthma', 'KidneyDisease', 'SkinCancer'

Manipulate Data of tmp1

```
tensor([[23.6500,  1.0000,  0.0000,  0.0000, 30.0000,  5.0000,  0.0000,  1.0000,
         10.0000,  5.0000,  2.0000,  3.0000,  3.0000,  8.0000,  0.0000,  0.0000,
          1.0000]])
tensor([[0.7170, 0.2830]])
```

Softmax

Data of tmp1

```
tensor([[23.6500,  1.0000,  0.0000,  0.0000, 30.0000,  5.0000,  0.0000,  1.0000,
         10.0000,  5.0000,  2.0000,  0.0000,  3.0000,  8.0000,  0.0000,  0.0000,
          1.0000]])
tensor([[0.6616, 0.3384]])
```

'BMI', 'Smoking', 'AlcoholDrinking', 'Stroke', 'PhysicalHealth', 'MentalHealth', '**DiffWalking**', 'Sex', 'AgeCategory', 'Race', 'Diabetic', '**PhysicalActivity**', 'GenHealth', 'SleepTime', 'Asthma', 'KidneyDisease', 'SkinCancer'

Manipulate Data of tmp1

```
tensor([[23.6500,  1.0000,  0.0000,  0.0000, 30.0000,  5.0000,  0.0000,  1.0000,
         10.0000,  5.0000,  2.0000,  3.0000,  3.0000,  8.0000,  0.0000,  0.0000,
          1.0000]])
tensor([[0.7170, 0.2830]])
```

Incidence reduced by 0.0554

Softmax

Data of Team Member

```
Jinnyeong = torch.Tensor([[22.0000,  0.0000,  0.0000,  0.0000, 0.0000,  5.0000,  0.0000,  0.0000,
         1.0000,  3.0000,  0.0000,  10.0000,  2.0000,  7.0000,  0.0000,  0.0000,
          0.0000]])
```

```
tensor([[0.9960, 0.0040]])
```

```
Chanju = torch.Tensor([[20.4400,  0.0000,  1.0000,  0.0000,  0.0000,  3.0000,  0.0000,  1.0000,
                        1.0000,  3.0000,  0.0000,  1.0000,  4.0000,  8.0000,  0.0000,  0.0000,
                        0.0000]])
```

```
tensor([[0.9953, 0.0047]])
```

```
Jaehyuk = torch.Tensor([[29.4000,  0.0000,  0.0000,  0.0000,  4.0000, 4.0000,  0.0000,  0.0000,
                        1.0000,  5.00000,  0.0000,  1.0000,  4.0000,  8.0000,  1.0000,   0.0000,
                        0.0000]])
```

```
tensor([[0.9953, 0.0047]])
```

```
Dongchan = torch.Tensor([[30.0600, 1.0000,  1.0000,  0.0000,  20.0000, 30.0000,  0.0000, 0.0000,
                         1.0000,  0.0000,  0.0000,  1.0000,  2.0000,  8.0000,  0.0000,  0.0000,
                         0.0000]])
```

```
tensor([[0.9779, 0.0221]])
```

04

Application
for RecoSys

Softmax

Data of Team Member

```
Dongchan = torch.Tensor([[30.0600, 1.0000, 1.0000, 0.0000, 20.0000, 30.0000, 0.0000, 0.0000,
                          1.0000, 0.0000, 0.0000, 1.0000, 2.0000, 8.0000, 0.0000, 0.0000,
                          0.0000]])
tensor([[0.9779, 0.0221]])
```

AgeCategory

```
Dongchan = torch.Tensor([[30.0600, 1.0000, 1.0000, 0.0000, 20.0000, 30.0000, 0.0000, 0.0000,
                          12.0000, 0.0000, 0.0000, 1.0000, 2.0000, 8.0000, 0.0000, 0.0000,
                          0.0000]])
tensor([[0.8550, 0.1450]])
```

12% increase

1	2	3	4	5	6	7	8	9	10	11	12
2.21	2.64	3.22	4.03	4.9	5.8	6.7	7.24	8.98	10.75	12.48	14.5

0.43	0.58	0.81	0.87	0.9	0.9	0.54	1.74	1.77	1.73	2.02

04

Application for RecoSys

Softmax

Data of Team Member

```
Dongchan = torch.Tensor([[30.0600, 1.0000, 1.0000, 0.0000, 20.0000, 30.0000, 0.0000, 0.0000,
                          1.0000, 0.0000, 0.0000, 1.0000, 2.0000, 8.0000, 0.0000, 0.0000,
                          0.0000]])
tensor([[0.9779, 0.0221]])
```

Smoking

```
Dongchan = torch.Tensor([[30.0600, 0.0000, 1.0000, 0.0000, 20.0000, 30.0000, 0.0000, 0.0000,
                          1.0000, 0.0000, 0.0000, 1.0000, 2.0000, 8.0000, 0.0000, 0.0000,
                          0.0000]])
tensor([[0.9805, 0.0195]])
```

0.3% decrease

```
Dongchan = torch.Tensor([[30.0600, 1.0000, 1.0000, 0.0000, 20.0000, 30.0000, 0.0000, 0.0000,
                          12.0000, 0.0000, 0.0000, 1.0000, 2.0000, 8.0000, 0.0000, 0.0000,
                          0.0000]])
tensor([[0.8550, 0.1450]])
```

```
Dongchan = torch.Tensor([[30.0600, 0.0000, 1.0000, 0.0000, 20.0000, 30.0000, 0.0000, 0.0000,
                          12.0000, 0.0000, 0.0000, 1.0000, 2.0000, 8.0000, 0.0000, 0.0000,
                          0.0000]])
tensor([[0.8749, 0.1251]])
```

2% decrease

04

Application for RecoSys

Softmax

Data of Team Member

```
Dongchan = torch.Tensor([[30.0600, 0.0000,  1.0000,  0.0000,  20.0000, 30.0000,  0.0000, 0.0000,
                          12.0000,  0.0000,  0.0000,  1.0000,  [illegible]  [illegible]  [illegible].0000,  0.0000,
                          0.0000]])
tensor([[0.8749, 0.1251]])
```

PhysicalActivity

```
Dongchan = torch.Tensor([[30.0600, 0.0000,  1.0000,  0.0000,  20.0000, 30.0000,  0.0000, 0.0000,
                          12.0000,  0.0000,  0.0000,  5.0000,  2.0000,  8.0000,  0.0000,  0.0000,
                          0.0000]])
tensor([[0.9062, 0.0938]])
```

3% decrease

Stop Smoking & Physical Activity

14.5% → 9.3%

: Contents

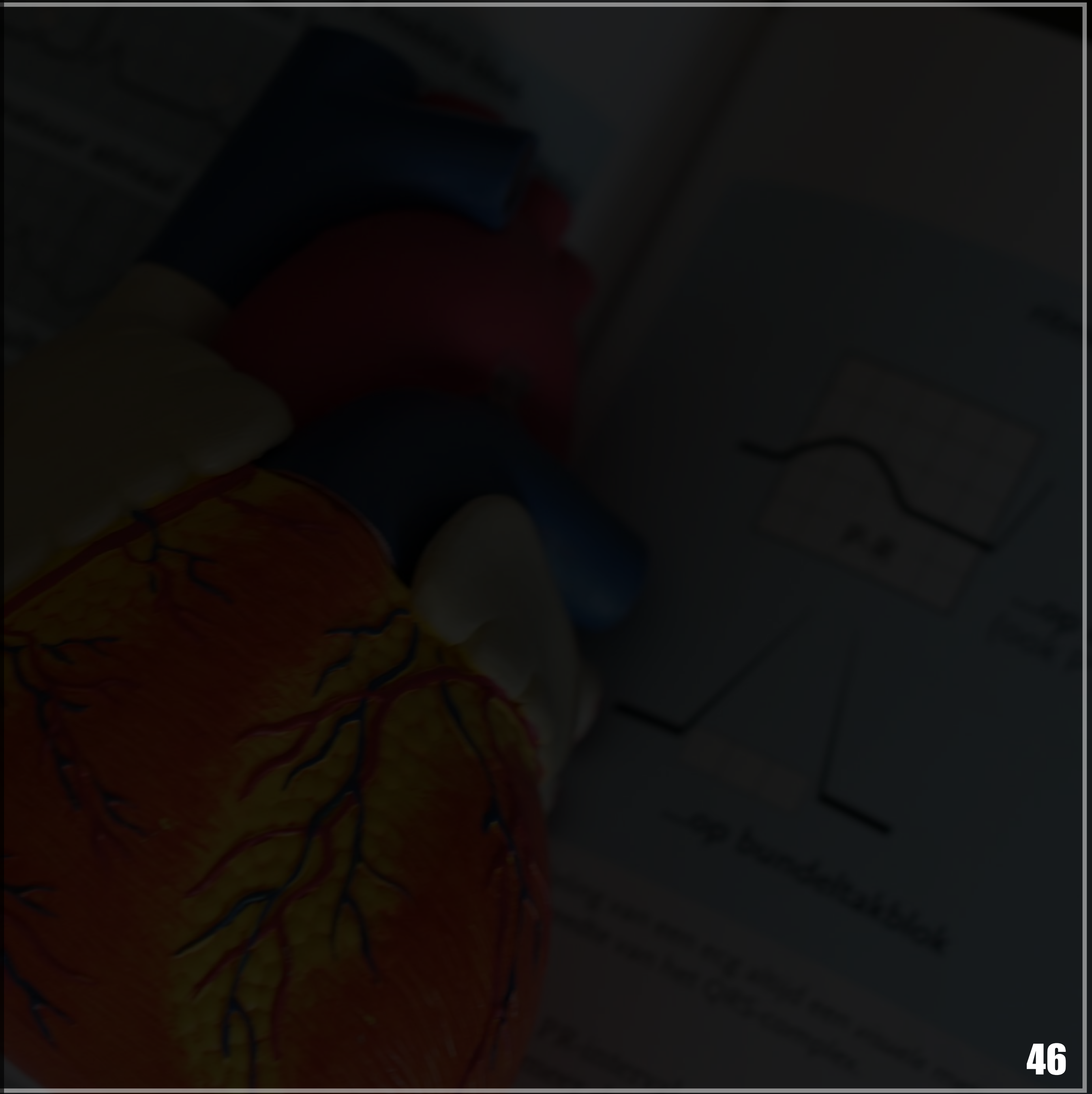
| 1st Dataset

| 2nd Models

| 3rd Selected Model

| 4th Application for RecoSys

| 5th Conclusion & Limitations



05

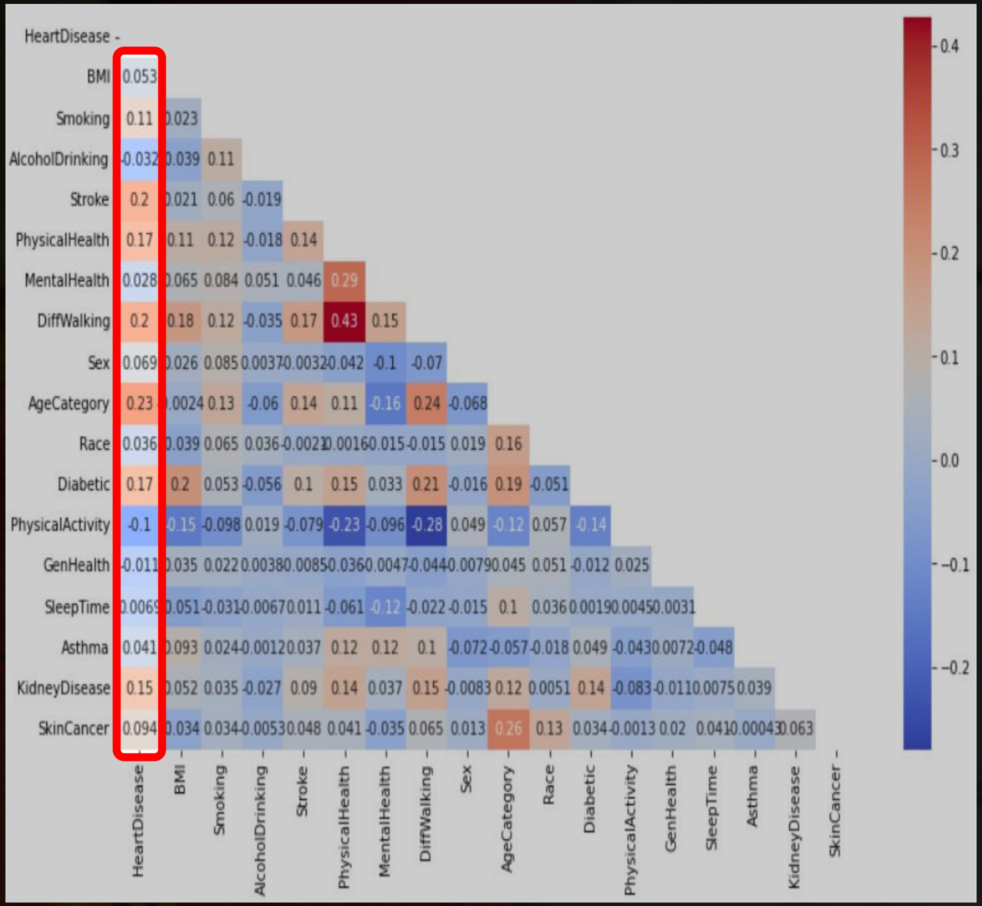
Conclusion & Limitations

Conclusion

Accuracy of ANN

```
Train Epoch: 0 [0/223856 (0%)]  Loss: 0.662906
Train Epoch: 0 [32000/223856 (14%)]     Loss: 0.200548
Train Epoch: 0 [64000/223856 (29%)]     Loss: 0.387938
Train Epoch: 0 [96000/223856 (43%)]     Loss: 0.311434
Train Epoch: 0 [128000/223856 (57%)]    Loss: 0.274645
Train Epoch: 0 [160000/223856 (71%)]    Loss: 0.236688
Train Epoch: 0 [192000/223856 (86%)]    Loss: 0.199064

Training set: Accuracy: 204669/223856 (91%)
Test set: Accuracy: 87753/95939 (91%)

...

Train Epoch: 9 [0/223856 (0%)]  Loss: 0.159966
Train Epoch: 9 [32000/223856 (14%)]     Loss: 0.281964
Train Epoch: 9 [64000/223856 (29%)]     Loss: 0.242223
Train Epoch: 9 [96000/223856 (43%)]     Loss: 0.214790
Train Epoch: 9 [128000/223856 (57%)]    Loss: 0.179357
Train Epoch: 9 [160000/223856 (71%)]    Loss: 0.197916
Train Epoch: 9 [192000/223856 (86%)]    Loss: 0.182939

Training set: Accuracy: 204731/223856 (91%)
Test set: Accuracy: 87789/95939 (92%)
```

Correlation Matrix



05

Conclusion & Limitations

Conclusion

Incidence Reduced

```
tensor([[23.6500,  1.0000,  0.0000,  0.0000, 30.0000,  5.0000,  1.0000,  1.0000,
         10.0000,  5.0000,  2.0000,  0.0000,  3.0000,  8.0000,  0.0000,  0.0000,
          1.0000]])
tensor([[0.5819, 0.4181]])
```

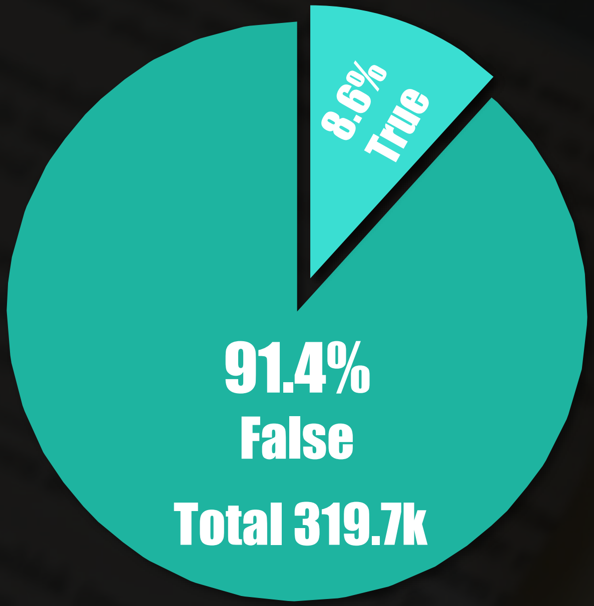
'Stroke', 'DiffWalking′ ,
'AgeCategory'PhysicalActivity

```
tensor([[23.6500,  1.0000,  0.0000,  0.0000, 30.0000,  5.0000,  0.0000,  1.0000,
         10.0000,  5.0000,  2.0000,  3.0000,  3.0000,  8.0000,  0.0000,  0.0000,
          1.0000]])
tensor([[0.7170, 0.2830]])
```

Using this model,
We can recommend what they should do

Limitations

Data Bias



Softmax about people who have heart Disease

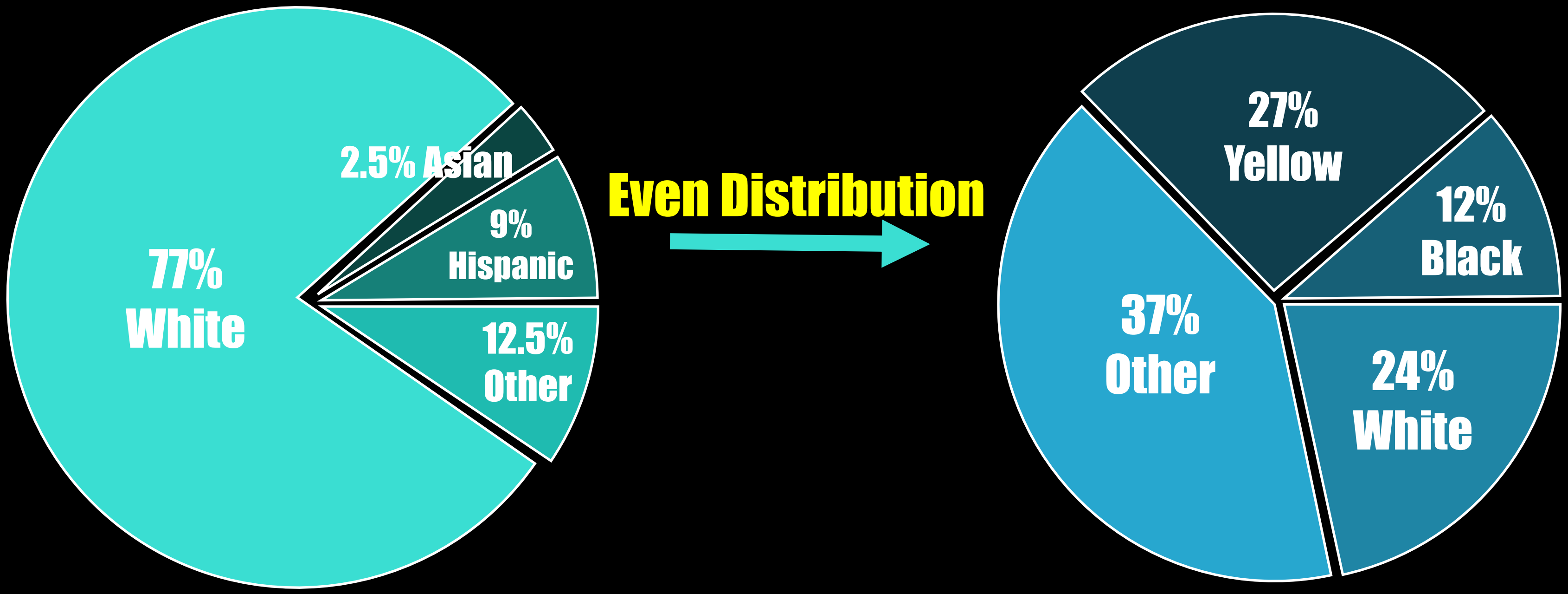
```
tensor([[0.7487, 0.2513],
        [0.7856, 0.2144],
        [0.7888, 0.2112],
        [0.7420, 0.2580],
        [0.5819, 0.4181],
        [0.8313, 0.1687],
        [0.6979, 0.3021],
        [0.7180, 0.2820],
        [0.9040, 0.0960],
        [0.5568, 0.4432],
        [0.9545, 0.0455],
        [0.8314, 0.1686],
        [0.5858, 0.4142],
        [0.9565, 0.0435],
```

Softmax about people who don't have heart Disease

```
tensor([[0.9744, 0.0257],
        [0.9574, 0.0426],
        [0.9739, 0.0261],
        [0.9965, 0.0035],
        [0.8071, 0.1929],
        [0.9595, 0.0405],
        [0.9618, 0.0382],
        [0.9882, 0.0118],
        [0.8761, 0.1239],
        [0.9933, 0.0067],
        [0.9759, 0.0241],
        [0.9945, 0.0055],
        [0.9079, 0.0921],
        [0.9759, 0.0241],
```

05

Conclusion & Limitations



05

Conclusion & Limitations

Powerful GPU

Q & A