CSM308 Машин Сургалт хичээлийн 2023-2024 явцын шалгалт 2

**1-р хэсэг 2 оноо**

1. **Санамсаргүй сургалтын 2 ангилалт сан үүсгэж 80,20 харьцаагаар хуваан дүрсэлж харуул.**

**import numpy as np**

**import pandas as pd**

**from sklearn.model\_selection import train\_test\_split**

**import matplotlib.pyplot as plt**

**data\_size = 100**

**data = np.vstack((np.random.normal(0, 1, (data\_size, 2)), np.random.normal(5, 1, (data\_size, 2))))**

**labels = [0] \* data\_size + [1] \* data\_size**

**df = pd.DataFrame(data, columns=['Feature\_1', 'Feature\_2'])**

**df['Label'] = labels**

**train\_set, test\_set = train\_test\_split(df, test\_size=0.2, random\_state=42, stratify=df['Label'])**

**fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))**

**ax1.scatter(train\_set['Feature\_1'], train\_set['Feature\_2'], c=train\_set['Label'], alpha=0.6)**

**ax1.set\_title('Train Set (80%)')**

**ax2.scatter(test\_set['Feature\_1'], test\_set['Feature\_2'], c=test\_set['Label'], alpha=0.6)**

**ax2.set\_title('Test Set (20%)')**

**plt.show()**

1. **Титаник сургалтын санг цэвэрлэж сургалтанд бэлтгэн графикаар дүрсэлж харуул**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import StandardScaler, LabelEncoder**

**df = pd.read\_csv('train.csv')**

**df['Age'].fillna(df['Age'].median(), inplace=True)**

**df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)**

**df.drop(columns=['Cabin', 'Ticket', 'Name'], inplace=True)**

**label\_encoder = LabelEncoder()**

**df['Sex'] = label\_encoder.fit\_transform(df['Sex'])**

**df['Embarked'] = label\_encoder.fit\_transform(df['Embarked'])**

**#target**

**X = df.drop(columns=['Survived'])**

**y = df['Survived']**

**scaler = StandardScaler()**

**X\_scaled = scaler.fit\_transform(X)**

**X\_scaled\_df = pd.DataFrame(X\_scaled, columns=X.columns)**

**fig, axes = plt.subplots(1, 2, figsize=(14, 6))**

**sns.histplot(data=df, x='Age', hue='Survived', multiple='stack', ax=axes[0])**

**axes[0].set\_title('Age Distribution by Survival')**

**sns.histplot(data=df, x='Fare', hue='Survived', multiple='stack', ax=axes[1])**

**axes[1].set\_title('Fare Distribution by Survival')**

**plt.tight\_layout()**

**plt.show()**

1. **MNIST сургалтын санг уншиж авах Dataset болон DataLoader ашиглан 1 багцыг зургаар дүрсэлж харуул**

**import torch**

**from torchvision import datasets, transforms**

**from torch.utils.data import DataLoader**

**import matplotlib.pyplot as plt**

**transform = transforms.Compose([transforms.ToTensor()])**

**mnist\_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)**

**mnist\_loader = DataLoader(mnist\_dataset, batch\_size=1, shuffle=True)**

**data\_iter = iter(mnist\_loader)**

**images, labels = next(data\_iter)**

**plt.imshow(images[0].squeeze(), cmap='gray')**

**plt.title(f"Label: {labels[0].item()}")**

**plt.axis('off')**

**plt.show()**

1. **MSE алдааны функцын шууд болон гэдрэг тархалтыг боловсруулах функцыг бич**

**import numpy as np**

**def mse\_direct\_and\_exponential(y\_true, y\_pred, alpha=1.0):**

**mse\_direct = np.mean((y\_true - y\_pred) \*\* 2)**

**mse\_exponential = np.exp(alpha \* mse\_direct)**

**return mse\_direct, mse\_exponential**

**y\_true = np.array([1.5, 2.0, 3.5])**

**y\_pred = np.array([1.4, 2.1, 3.2])**

**mse\_direct, mse\_exponential = mse\_direct\_and\_exponential(y\_true, y\_pred)**

**print("Шууд MSE:", mse\_direct)**

**print("Гэдрэг MSE:", mse\_exponential)**

1. **Шугаман давхрагын шууд болон гэдрэг тархалыг боловсруулах функцыг бичнэ үү**

**import numpy as np**

**class LinearLayer:**

**def \_\_init\_\_(self, input\_dim, output\_dim, learning\_rate=0.01):**

**self.weights = np.random.randn(input\_dim, output\_dim) \* 0.01**

**self.bias = np.zeros((1, output\_dim))**

**self.learning\_rate = learning\_rate**

**def forward(self, X):**

**self.input = X**

**self.output = np.dot(X, self.weights) + self.bias**

**return self.output**

**def backward(self, d\_out):**

**self.d\_weights = np.dot(self.input.T, d\_out)**

**self.d\_bias = np.sum(d\_out, axis=0, keepdims=True)**

**d\_input = np.dot(d\_out, self.weights.T)**

**self.weights -= self.learning\_rate \* self.d\_weights**

**self.bias -= self.learning\_rate \* self.d\_bias**

**return d\_input**

1. **Шугаман регрессын шууд болон гэдрэг тархалтыг боловсруулах функцыг бичнэ үү**

**import numpy as np**

**class LinearRegression:**

**def \_\_init\_\_(self, input\_dim, learning\_rate=0.01):**

**# Жингүүдийг санамсаргүй утгаар эхлүүлэх**

**self.weights = np.random.randn(input\_dim, 1) \* 0.01**

**self.bias = 0**

**self.learning\_rate = learning\_rate**

**def forward(self, X):**

**"""**

**Шугаман регрессын шууд тархалт.**

**X: Оролтын өгөгдөл (num\_samples, input\_dim)**

**Буцаах утга: Таамагласан утга (num\_samples, 1)**

**"""**

**self.input = X # Оролтын өгөгдлийг хадгалах (гэдрэг тархалтад ашиглах)**

**self.output = np.dot(X, self.weights) + self.bias**

**return self.output**

**def backward(self, y\_true):**

**"""**

**Гэдрэг тархалт буюу градиент тооцоолол.**

**y\_true: Бодит утга (num\_samples, 1)**

**"""**

**# Алдааны функцын градиент тооцоолол**

**num\_samples = self.input.shape[0]**

**error = self.output - y\_true**

**# Жин ба бясааны градиент**

**d\_weights = (1 / num\_samples) \* np.dot(self.input.T, error)**

**d\_bias = (1 / num\_samples) \* np.sum(error)**

**# Жинг шинэчлэх**

**self.weights -= self.learning\_rate \* d\_weights**

**self.bias -= self.learning\_rate \* d\_bias**

**# Жишээ ашиглалт**

**# Жишээ өгөгдөл**

**X = np.array([[1], [2], [3], [4]]) # Бие даасан хувьсагч**

**y = np.array([[2], [4], [6], [8]]) # Зорилтот хувьсагч**

**# Загварыг үүсгэх ба сургалт**

**model = LinearRegression(input\_dim=1, learning\_rate=0.01)**

**# Сургалт явуулах**

**epochs = 1000**

**for epoch in range(epochs):**

**predictions = model.forward(X)**

**model.backward(y)**

**# Алдаа тооцож хэвлэх**

**if epoch % 100 == 0:**

**loss = np.mean((predictions - y) \*\* 2)**

**print(f"Epoch {epoch}, Loss: {loss}")**

1. **Дурын сургалтын санг k-fold аргаар хуваа**

**import torch**

**from torchvision import datasets, transforms**

**from torch.utils.data import DataLoader, Subset**

**from sklearn.model\_selection import KFold**

**transform = transforms.Compose([transforms.ToTensor()])**

**mnist\_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)**

**k = 5 # 5-fold хуваалт**

**kf = KFold(n\_splits=k, shuffle=True, random\_state=42)**

**fold = 1**

**for train\_index, test\_index in kf.split(mnist\_dataset):**

**train\_subset = Subset(mnist\_dataset, train\_index)**

**test\_subset = Subset(mnist\_dataset, test\_index)**

**train\_loader = DataLoader(train\_subset, batch\_size=64, shuffle=True)**

**test\_loader = DataLoader(test\_subset, batch\_size=64, shuffle=False)**

**print(f"Fold {fold}")**

**print(f"Training samples: {len(train\_subset)}, Validation samples: {len(test\_subset)}")**

**fold += 1**

1. **Зураган өгөгдлийг 0-1 хооронд scale хий**

**from torchvision import datasets, transforms**

**from torch.utils.data import DataLoader**

**import matplotlib.pyplot as plt**

**# Зурагт өгөгдлийг 0-1 хооронд scale хийх ToTensor transform**

**transform = transforms.Compose([transforms.ToTensor()])**

**# MNIST өгөгдлийг ачаалах**

**mnist\_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)**

**data\_loader = DataLoader(mnist\_dataset, batch\_size=1, shuffle=True)**

**# Жишээ зураг харах**

**data\_iter = iter(data\_loader)**

**images, labels = next(data\_iter)**

**# Зургийг дүрслэх**

**plt.imshow(images[0].squeeze(), cmap='gray')**

**plt.title(f"Label: {labels[0].item()}")**

**plt.axis('off')**

**plt.show()**

1. **Текст өгөгдлийг токеназор ашиглан үр дүнг хэвлэн харуул**

**from transformers import AutoTokenizer**

**def tokenize\_text(text, model\_name="bert-base-uncased"):**

**# Модельд тохирсон токенизаторыг ачаалах**

**tokenizer = AutoTokenizer.from\_pretrained(model\_name)**

**# Текстийг токенжуулах**

**encoding = tokenizer(text, add\_special\_tokens=True)**

**# Токен болон ID-г авах**

**tokens = tokenizer.convert\_ids\_to\_tokens(encoding["input\_ids"])**

**token\_ids = encoding["input\_ids"]**

**return tokens, token\_ids**

**# Жишээ ашиглалт**

**text\_data = "Machine learning enables computers to learn from data."**

**tokens, token\_ids = tokenize\_text(text\_data)**

**# Үр дүнг хэвлэх**

**print("Токенууд:", tokens)**

**print("Токен ID-ууд:", token\_ids)**

1. **Дурын 2 өгөгдлийн санг нэгтгэн шинэ өгөгдлийн сан үүсгэж Dataset болон DataLoader функцуудыг хэрэгжүүл**

**import torch**

**from torch.utils.data import Dataset, DataLoader, ConcatDataset**

**class CustomDataset(Dataset):**

**def \_\_init\_\_(self, data, targets):**

**self.data = data**

**self.targets = targets**

**def \_\_len\_\_(self):**

**return len(self.data)**

**def \_\_getitem\_\_(self, idx):**

**return self.data[idx], self.targets[idx]**

**class MergedDataset(Dataset):**

**def \_\_init\_\_(self, dataset1, dataset2):**

**# Concatenate datasets to form a new merged dataset**

**self.dataset = ConcatDataset([dataset1, dataset2])**

**def \_\_len\_\_(self):**

**return len(self.dataset)**

**def \_\_getitem\_\_(self, idx):**

**return self.dataset[idx]**

**data1 = torch.randn(100, 3)**

**targets1 = torch.randint(0, 2, (100,))**

**data2 = torch.randn(150, 3)**

**targets2 = torch.randint(0, 2, (150,))**

**dataset1 = CustomDataset(data1, targets1)**

**dataset2 = CustomDataset(data2, targets2)**

**merged\_dataset = MergedDataset(dataset1, dataset2)**

**merged\_dataloader = DataLoader(merged\_dataset, batch\_size=32, shuffle=True)**

**for batch\_data, batch\_targets in merged\_dataloader:**

**print(batch\_data.shape, batch\_targets.shape)**

**2-р хэсэг 3 оноо**

1. **3 давхрагат нерон ангилагч хэрэгжүүлэх объект функц бичнэ үү**

**import numpy as np**

**class ThreeLayerNeuralClassifier:**

**def \_\_init\_\_(self, n\_input, n\_hidden, n\_output, learning\_rate=0.01):**

**# Параметрүүдийг оруулах**

**self.n\_input = n\_input**

**self.n\_hidden = n\_hidden**

**self.n\_output = n\_output**

**self.learning\_rate = learning\_rate**

**# Жин, хазайлт утгуудыг эхлүүлэх**

**self.W1 = np.random.randn(n\_input, n\_hidden) \* 0.01**

**self.b1 = np.zeros((1, n\_hidden))**

**self.W2 = np.random.randn(n\_hidden, n\_output) \* 0.01**

**self.b2 = np.zeros((1, n\_output))**

**def sigmoid(self, z):**

**# Sigmoid идэвхжүүлэгч функц**

**return 1 / (1 + np.exp(-z))**

**def sigmoid\_derivative(self, z):**

**# Sigmoid функцын тууштай хувьсагч**

**return self.sigmoid(z) \* (1 - self.sigmoid(z))**

**def forward\_propagation(self, X):**

**self.Z1 = np.dot(X, self.W1) + self.b1 # Z1 = X \* W1 + b1**

**self.A1 = self.sigmoid(self.Z1) # А1 = Sigmoid(Z1)**

**self.Z2 = np.dot(self.A1, self.W2) + self.b2 # Z2 = A1 \* W2 + b2**

**self.A2 = self.sigmoid(self.Z2) # А2 = Sigmoid(Z2)**

**return self.A2**

**def compute\_loss(self, Y, Y\_hat):**

**# Алдааг тооцоолох: бинар лог-алдаа**

**m = Y.shape[0] # Дата багцын хэмжээ**

**loss = -np.sum(Y \* np.log(Y\_hat) + (1 - Y) \* np.log(1 - Y\_hat)) / m # Логистик алдаа**

**return loss**

**def backward\_propagation(self, X, Y):**

**m = X.shape[0] # Дата багцын хэмжээ**

**# Гаралт давхаргын хураангуй**

**dZ2 = self.A2 - Y**

**dW2 = np.dot(self.A1.T, dZ2) / m**

**db2 = np.sum(dZ2, axis=0, keepdims=True) / m**

**# Нууц давхаргын хураангуй**

**dZ1 = np.dot(dZ2, self.W2.T) \* self.sigmoid\_derivative(self.Z1)**

**dW1 = np.dot(X.T, dZ1) / m**

**db1 = np.sum(dZ1, axis=0, keepdims=True) / m**

**# Жин, хазайлт шинэчлэх**

**self.W1 -= self.learning\_rate \* dW1 # W1-г шинэчлэх**

**self.b1 -= self.learning\_rate \* db1 # b1-г шинэчлэх**

**self.W2 -= self.learning\_rate \* dW2 # W2-г шинэчлэх**

**self.b2 -= self.learning\_rate \* db2 # b2-г шинэчлэх**

**def train(self, X, Y, epochs=1000):**

**# Сургалт**

**for epoch in range(epochs):**

**# (forward propagation)**

**Y\_hat = self.forward\_propagation(X)**

**# Алдаа тооцоолох**

**loss = self.compute\_loss(Y, Y\_hat)**

**# (backward propagation)**

**self.backward\_propagation(X, Y)**

**if epoch % 100 == 0:**

**print(f"Epoch {epoch}: Loss = {loss}")**

**def predict(self, X):**

**# Шинэ өгөгдөл дээр таамаглал хийх**

**Y\_hat = self.forward\_propagation(X)**

**return (Y\_hat > 0.5).astype(int) # 0.5-ээс их бол 1, бага бол 0 гэж тооцоолно**

**# Жишээ хэрэглээ**

**n\_input = 2**

**n\_hidden = 4**

**n\_output = 1**

**learning\_rate = 0.01**

**X = np.random.randn(100, n\_input)**

**Y = (np.sum(X, axis=1) > 0).reshape(-1, 1).astype(int)**

**# Нейрон сүлжээ үүсгэх, сургалт хийх**

**nn = ThreeLayerNeuralClassifier(n\_input, n\_hidden, n\_output, learning\_rate)**

**nn.train(X, Y, epochs=1000)**

**# Шинэ өгөгдөл дээр таамаглал хийх**

**predictions = nn.predict(X)**

**print("Predictions:", predictions) # Таамаглал**

1. **Шугаман ангилагч сургалтыг дурын сургалтын санд сурган үр дүнг хэвлэ**
2. **MNIST санг PCA ашиглан 3 хэмжээст графикаар дүрсэлж харуул. 200 жишээ ашиглана.**
3. **Логистик регресс housing price dataset дээр сурган 3 таамаглалыг хэвлэн харуул**
4. **3 давхрагат ANN сүлжээг CUPY ашиглан дурын өгөгдөлд сурга**
5. **4 давхрагат ANN сүлжээг numpy ашиглан дурын өгөгдлийн санд сурга**
6. **Дурын сургасан загвар болон бусад сургалтанд ашиглагдах хэрэглэгдэхүүнүүдийн хамт хадгал Pytorch**
7. **Дурын хадгалсан загварыг уншиж аван үр дүнг тестийн санд хэвлэж харуул**
8. **Resnet18 өмнө сургасан загварыг MNIST өгөгдлийн санд finetune хийж тестийн үр дүнг хэвлэж харуул. Нэмэлт сургалт хийх шаардлагагүй**
9. **EfficientNet-B0 өмнө сургасан загварыг MNIST өгөгдлийн санд finetune хийж тестийн үр дүнг хэвлэж харуул. Нэмэлт сургалт хийх шаардлагагүй**

**3-р хэсэг 5 оноо**

1. **Өөрийн plagiarism шалгагчыг текст өгөгдөлд сургана уу**

import numpy as np

class NeuralNetwork:

def \_\_init\_\_(self, input\_size, hidden\_size, output\_size, learning\_rate=0.1):

# Давхаргуудын жинг эхлүүлэх

self.learning\_rate = learning\_rate

self.weights\_input\_hidden = np.random.randn(input\_size, hidden\_size)

self.weights\_hidden\_output = np.random.randn(hidden\_size, output\_size)

self.bias\_hidden = np.zeros((1, hidden\_size))

self.bias\_output = np.zeros((1, output\_size))

def sigmoid(self, x):

return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(self, x):

return x \* (1 - x)

def forward(self, X):

# Оролтоос далд давхарга руу

self.hidden\_input = np.dot(X, self.weights\_input\_hidden) + self.bias\_hidden

self.hidden\_output = self.sigmoid(self.hidden\_input)

# Далд давхаргаас гаралтын давхарга руу

self.output\_input = np.dot(self.hidden\_output, self.weights\_hidden\_output) + self.bias\_output

self.output = self.sigmoid(self.output\_input)

return self.output

def backward(self, X, y, output):

# Алдаа тооцоолох

output\_error = y - output

output\_delta = output\_error \* self.sigmoid\_derivative(output)

# Далд давхаргын алдаа ба градиент

hidden\_error = np.dot(output\_delta, self.weights\_hidden\_output.T)

hidden\_delta = hidden\_error \* self.sigmoid\_derivative(self.hidden\_output)

# Жин болон бясаа шинэчлэх

self.weights\_hidden\_output += self.learning\_rate \* np.dot(self.hidden\_output.T, output\_delta)

self.bias\_output += self.learning\_rate \* np.sum(output\_delta, axis=0, keepdims=True)

self.weights\_input\_hidden += self.learning\_rate \* np.dot(X.T, hidden\_delta)

self.bias\_hidden += self.learning\_rate \* np.sum(hidden\_delta, axis=0, keepdims=True)

def train(self, X, y, epochs=10000):

for epoch in range(epochs):

output = self.forward(X)

self.backward(X, y, output)

if epoch % 1000 == 0:

loss = np.mean(np.square(y - output))

print(f"Epoch {epoch}, Loss: {loss}")

def predict(self, X):

return self.forward(X)

# Жишээ ашиглалт

# Оролт 2, далд 4, гаралт 1 нейронтой, сургалтын хурд 0.1

nn = NeuralNetwork(input\_size=2, hidden\_size=4, output\_size=1, learning\_rate=0.1)

# Оролт болон гаралтын өгөгдлүүд

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

y = np.array([[0], [1], [1], [0]]) # XOR функц

# Сургалт

nn.train(X, y, epochs=10000)

# Таамаглал

print("Predictions:")

print(nn.predict(X))

1. **Covid19 өгөгдөлд GRU сурган тестийн өгөгдөлд үр дүнг графикаар дүрсэлж харуул**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import GRU, Dense

from tensorflow.keras.optimizers import Adam

# 1. COVID-19 Өгөгдлийг интернетээс шууд ачаалах

data\_url = 'https://covid.ourworldindata.org/data/owid-covid-data.csv'

data = pd.read\_csv(data\_url)

# Тохиолдлын мэдээллийг ачаалах

data = data[['date', 'new\_cases']].dropna() # Өдөр бүрийн тохиолдлын тоог ашиглах

data['date'] = pd.to\_datetime(data['date'])

data.set\_index('date', inplace=True)

# 2. Өгөгдлийг 0-1 хооронд нормчлох

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(data[['new\_cases']])

# 3. Сургалтын болон тестийн өгөгдөл үүсгэх

train\_size = int(len(scaled\_data) \* 0.8)

train\_data = scaled\_data[:train\_size]

test\_data = scaled\_data[train\_size:]

# 4. Сургалтын өгөгдлийг цонхолж үүсгэх

def create\_dataset(data, time\_step=60):

X, Y = [], []

for i in range(len(data) - time\_step):

X.append(data[i:i + time\_step, 0])

Y.append(data[i + time\_step, 0])

return np.array(X), np.array(Y)

time\_step = 60 # 60 хоногийн өмнөх мэдээлэл дээр үндэслэнэ

X\_train, y\_train = create\_dataset(train\_data, time\_step)

X\_test, y\_test = create\_dataset(test\_data, time\_step)

# 5. Хэлбэрийг GRU-д тохируулах

X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)

X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)

# 6. GRU загвар үүсгэх

model = Sequential([

GRU(50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)),

GRU(50, return\_sequences=False),

Dense(25),

Dense(1)

])

# Загварыг компиляцлах

model.compile(optimizer=Adam(learning\_rate=0.001), loss='mean\_squared\_error')

# 7. Загварыг сургалтанд оруулах

model.fit(X\_train, y\_train, epochs=20, batch\_size=32, validation\_data=(X\_test, y\_test))

# 8. Загварын таамаглалыг гаргах ба скейлийг буцааж өөрчлөх

train\_predict = model.predict(X\_train)

test\_predict = model.predict(X\_test)

train\_predict = scaler.inverse\_transform(train\_predict)

y\_train = scaler.inverse\_transform([y\_train])

test\_predict = scaler.inverse\_transform(test\_predict)

y\_test = scaler.inverse\_transform([y\_test])

# 9. Таамаглалыг графикаар дүрслэх

# Сургалтын өгөгдлийн таамаглал

train\_plot = np.empty\_like(scaled\_data)

train\_plot[:, :] = np.nan

train\_plot[time\_step:len(train\_predict) + time\_step, :] = train\_predict

# Тест өгөгдлийн таамаглал

test\_plot = np.empty\_like(scaled\_data)

test\_plot[:, :] = np.nan

test\_plot[len(train\_predict) + (time\_step \* 2):len(scaled\_data), :] = test\_predict

# Бодит өгөгдөл ба таамаглалыг дүрслэх

plt.figure(figsize=(12, 6))

plt.plot(scaler.inverse\_transform(scaled\_data), label='Бодит өгөгдөл')

plt.plot(train\_plot, label='Сургалтын өгөгдөлд таамагласан')

plt.plot(test\_plot, label='Тест өгөгдөлд таамагласан')

plt.xlabel('Өдөр')

plt.ylabel('Тохиолдлын тоо')

plt.legend()

plt.show()

1. **SVM сургалтыг титаник санд сурган тестийн өгөгдөлд гарсан үр дүнг харуул**
2. **Дурын сургалтын бүх үнэлгээг дүрсэлж харуул f1, recall, precision, confusion matrix**
3. **AdaBoost ашиглан титаник сургалтын сан дээрх сургалтыг base learner шугаман ангилагчтай сурган үр дүнг хэвлэ**
4. **ANN загварын сургалтыг hyperparametertune хийж хамгийн өндөр гүйцэтгэлийг хамгийн хурданаар гаргах параметрүүдийг хэвлэ**
5. **Дурын 2 сургасан загварыг ensemble хийн үр дүнг хэвлэ**
6. **2 загварыг ensemble сургалтын арга ашиглан үр дүнг хэвлэ**
7. **GMM ашиглан iris өгөгдлийн санг хяналтгүй сургалт хийн үр дүнг графикаар дүрсэлж харуул**
8. **CIFAR100 өгөгдлийн санд ANN сурган үр дүнг f1, recall, precision, confusion matrix тус тус харуул**