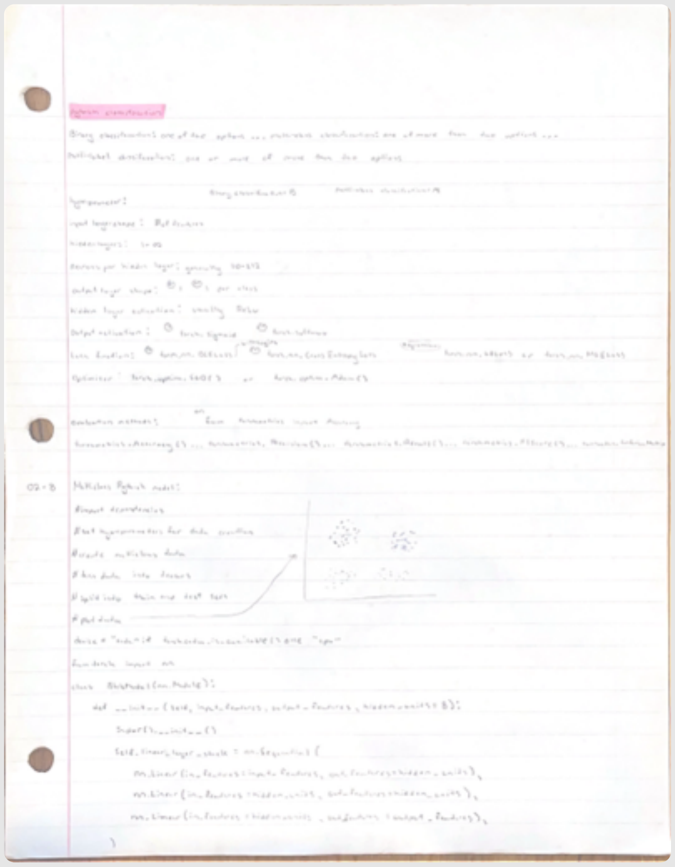
# 02Classification

## command



[illegible] classification

Binary classification: one of two options ... multiclass classification: one of more than two options ...

Multilabel classification: one or more of more than two options

hyperparameter: Binary classification [illegible] Multiclass classification [illegible]

input layer shape: [illegible]

hidden layers: [illegible]

neurons per hidden layer: generally 10-512

output layer shape: [illegible] per class

hidden layer activation: usually ReLU

output activation: torch.sigmoid torch.softmax

loss function: torch.nn.BCELoss [illegible] torch.nn.CrossEntropyLoss [illegible] torch.nn.[illegible] or torch.nn.[illegible]

optimizer: torch.optim.SGD() or torch.optim.Adam()

evaluation methods: from torchmetrics import Accuracy

torchmetrics.Accuracy() ... torchmetrics.Precision() ... torchmetrics.Recall() ... torchmetrics.F1Score() ... [illegible]

02-B Multiclass PyTorch model:

#import dependencies

#set hyperparameters for data creation

#create multiclass data

#turn data into tensors

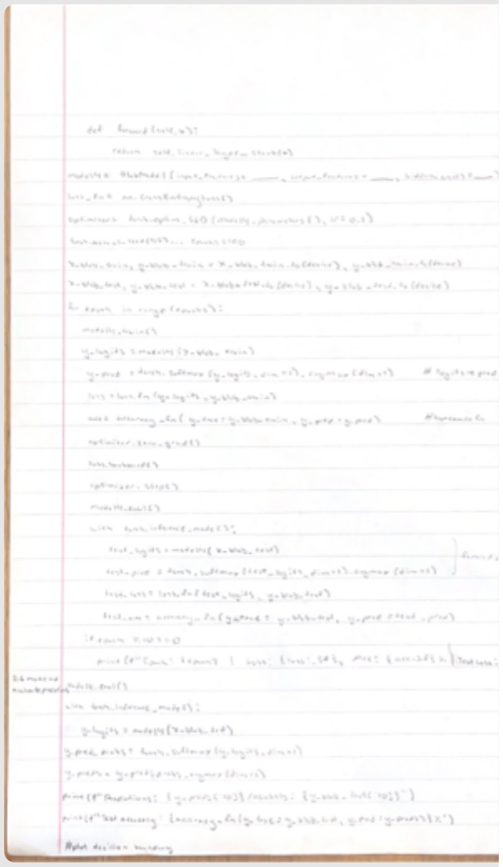
#split into train and test sets

#plot data

device = "cuda" if torch.cuda.is_available() else "cpu"

from torch import nn

```
class BlobModel(nn.Module):
    def __init__(self, input_features, output_features, hidden_units=8):
        super().__init__()
        self.linear_layer_stack = nn.Sequential(
            nn.Linear(in_features=input_features, out_features=hidden_units),
            nn.Linear(in_features=hidden_units, out_features=hidden_units),
            nn.Linear(in_features=hidden_units, out_features=output_features),
        )
```



```
    def forward(self, x):
        return self.linear_layer_stack(x)
model_4 = BlobModel(input_features=[illegible], output_features=[illegible], hidden_units=[illegible]
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model_4.parameters(), lr=0.1)
torch.manual_seed(42) ... epochs = 100
X_blob_train, y_blob_train = X_blob_train.to(device), y_blob_train.to(device)
X_blob_test, y_blob_test = X_blob_test.to(device), y_blob_test.to(device)
for epoch in range(epochs):
    model_4.train()
    y_logits = model_4(X_blob_train)
    y_pred = torch.softmax(y_logits, dim=1).argmax(dim=1)          # [illegible]
    loss = loss_fn(y_logits, y_blob_train)
    acc = accuracy_fn(y_true=y_blob_train, y_pred=y_pred)          # [illegible]
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    model_4.eval()
    with torch.inference_mode():
        test_logits = model_4(X_blob_test)
        test_pred = torch.softmax(test_logits, dim=1).argmax(dim=1)
        test_loss = loss_fn(test_logits, y_blob_test)
        test_acc = accuracy_fn(y_true=y_blob_test, y_pred=test_pred)
    if epoch % 10 == 0:
        print(f"Epoch: {epoch} | Loss: {loss:.5f}, Acc: {acc:.2f}% | Test loss: [illegible]
model_4.eval()
with torch.inference_mode():
    y_logits = model_4(X_blob_test)
y_pred_probs = torch.softmax(y_logits, dim=1)
y_preds = y_pred_probs.argmax(dim=1)
print(f"Predictions: {y_preds[:10]}\nLabels: {y_blob_test[:10]}")
print(f"Test accuracy: {accuracy_fn(y_true=y_blob_test, y_pred=y_preds)}%")
```

Plot decision boundary