1. Packages

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.model_selection import StratifiedKFold
import sys
from sklearn.linear_model import LogisticRegression
from scipy.sparse import csr_matrix
from sklearn.model_selection import train_test_split
import tqdm.notebook as tqdm
```

2. Load processed data

```
if 'google.colab' in sys.modules:
    from google.colab import drive
    drive.mount('/content/drive')
    DATA_PATH = '/content/drive/MyDrive/DL_Project/data/'
else:
    DATA_PATH = './Documents/Classes/AML/proj/archive/'
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
data = pd.read_pickle(DATA_PATH + 'train.pkl')
data.head()
```

	customer_ID	S_2	target	P_2	D_39	B_1	B_2	R_1	S_3	D_41	...	D_136
5	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-08-04	0.0	1.167299	-0.753275	-0.596688	1.050376	-0.200328	-0.899846	0.779986	...	-1.110223e-16
6	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-09-18	0.0	1.125353	-0.742243	-0.382683	1.070227	-0.103956	-1.012425	0.203476	...	-1.110223e-16
7	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-10-08	0.0	0.905170	-0.720954	-0.468895	1.017525	-1.711672	-0.941778	0.726907	...	-1.110223e-16
8	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-11-20	0.0	1.209616	-0.555256	-0.419792	1.043782	-1.082661	-1.020662	-0.702712	...	-1.110223e-16
9	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-12-04	0.0	0.546506	-0.770175	-0.719339	1.070227	-1.089397	-1.055643	-0.206806	...	-1.110223e-16

5 rows × 191 columns

```
data_X, data_Y = data.drop(['customer_ID', 'S_2', 'target'], axis=1), np.ravel(data['target'])
```

```
del data
```

```
from sklearn.model_selection import train_test_split
train_X, test_X, train_Y, test_Y = train_test_split(data_X, data_Y, test_size=0.4, shuffle=False)
```

```
train_X.head()
```

	P_2	D_39	B_1	B_2	R_1	S_3	D_41	B_3	D_42	D_43	...	D_136	D_137	D_138	D_139
5	1.167299	-0.753275	-0.596688	1.050376	-0.200328	-0.899846	0.779986	-0.418402	0.144807	0.594656	...	-1.110223e-16	8.673617e-18	-4.547474e-13	-0.482683
												-1.110223e-	8.673617e-	-4.547474e-	

3. Build model

3.1 Baseline: Logistic Regression

```
clf = LogisticRegression(random_state=0, max_iter=500).fit(train_X, train_Y)
```

5 rows × 188 columns

3.1.2 Evaluation for logistic regression model

```
from sklearn.metrics import classification_report
```

```
print('Accuracy:', 100* clf.score(train_X, train_Y), '%')
train_predictions = clf.predict(train_X)
train_report = classification_report(train_Y, train_predictions)
print(train_report)
```

```
Accuracy: 92.29614876023989 %
              precision    recall  f1-score   support

         0.0       0.94      0.97      0.96   1304809
         1.0       0.68      0.51      0.58    153215

    accuracy                           0.92   1458024
   macro avg       0.81      0.74      0.77   1458024
weighted avg       0.92      0.92      0.92   1458024
```

```
# testing
print('Accuracy:', 100* clf.score(test_X, test_Y), '%')
test_predictions = clf.predict(test_X)
test_report = classification_report(test_Y, test_predictions)
print(test_report)
```

```
Accuracy: 92.36154548896315 %
              precision    recall  f1-score   support

         0.0       0.94      0.97      0.96    871134
         1.0       0.68      0.51      0.58    100882

    accuracy                           0.92    972016
   macro avg       0.81      0.74      0.77    972016
weighted avg       0.92      0.92      0.92    972016
```

```
del train_predictions, test_predictions, train_report, test_report
```

Double-click (or enter) to edit

3.2 TCN Model

https://towardsdatascience.com/temporal-coils-intro-to-temporal-convolutional-networks-for-time-series-forecasting-in-python-5907c04febc6

https://github.com/jakeret/tcn/blob/master/tcn.py

3.2.1 Building TCN model

```
import torch
import torch.nn as nn
from torch.nn.utils import weight_norm
```

```
class ResidualBlock(nn.Module):
    def __init__(self,
                 inputs,
```

```
                 outputs,
                 kernel_size: int,
                 dilation: int,
                 dropout_rate: float,
                 ):
        super(ResidualBlock, self).__init__()

        # First convolutional layer
        self.Conv1 =  weight_norm(nn.Conv1d(inputs, outputs, kernel_size=kernel_size, dilation=dilation))
        self.Relu1 = nn.ReLU()
        self.Dropout1 = nn.Dropout(dropout_rate)

        # Second convolutional layer
        self.Conv2 =  weight_norm(nn.Conv1d(outputs, outputs, kernel_size=kernel_size, dilation=dilation))
        self.Relu2 = nn.ReLU()
        self.Dropout2 = nn.Dropout(dropout_rate)

        self.net = nn.Sequential(self.Conv1, self.Relu1, self.Dropout1,
                                 self.Conv2, self.Relu2, self.Dropout2)

    def forward(self, inputs):
        out = self.net(inputs)
        return out
```

```
class TemporalConvNet(nn.Module):
    def __init__(self, num_inputs, num_channels, kernel_size, dropout):
        super(TemporalConvNet, self).__init__()

        self.blocks = nn.ModuleList()
        self.depth = len(num_channels)

        for i in range(self.depth):
            dilation_size = 2 ** i
            in_channels = num_inputs if i == 0 else num_channels[i-1]
            out_channels = num_channels[i]
            self.blocks.append(ResidualBlock(in_channels, out_channels, kernel_size, dilation_size, dropout))

        self.fc = nn.Linear(num_channels[-1], 1)

    def forward(self, x):
        for block in self.blocks:
            x = block(x)
        x = x.mean(dim=2)
        x = self.fc(x)
        return x.squeeze()
```

3.2.2 Training

Update: Create time window first, then split train & test (with shuffle)

```
data_X.values.shape[0]
```

```
2430040
```

```
# create dataloader
from torch.utils.data import TensorDataset, DataLoader

window_size = 8

X_values = torch.Tensor(data_X.values[:100000])
Y_values = torch.Tensor(data_Y[:100000])

# Create windows of 5 months in the input data
inputs = [X_values[i:i+window_size].transpose(0, 1) for i in range(len(X_values) - window_size + 1)]
labels = Y_values[window_size-1:]

inputs = torch.stack(inputs)  # Convert list of tensors to a single tensor

# Create your dataloader
dataset = TensorDataset(inputs, labels)

train_data, test_data = torch.utils.data.random_split(dataset, [0.7, 0.3])

del dataset

```

```
batch_size = 64
train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=True)
```

```
feature_num = data_X.values.shape[1]
```

```
del data_X, data_Y
```

```
def get_test_loss(model, test_loader, loss_function):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.eval()
    test_loss = 0
    num_test_predictions_correct = 0
    num_test_predictions_total = 0

    with torch.no_grad():
        for data in tqdm.tqdm(test_loader, colour='green', desc='test', leave=False):
            inputs, labels = data
            inputs = inputs.to(device)
            labels = labels.to(device)

            labels = labels.unsqueeze(1)  # Add an extra dimension to the target tensor

            outputs = model(inputs)
            labels = labels.squeeze(1)
            loss = loss_function(outputs, labels)
            test_loss += loss.item()

            pred_labels = (torch.sigmoid(outputs) > 0.5).float()
            num_test_predictions_correct += (pred_labels == labels).sum().item()
            num_test_predictions_total += len(pred_labels)

    test_loss /= len(test_loader)
    test_accuracy = num_test_predictions_correct / num_test_predictions_total * 100

    return test_loss, test_accuracy
```

```
import tqdm.notebook as tqdm

def train(model, train_loader, loss_function, optimizer, epochs, test_loader=None):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)
    train_record = {'loss': [], 'acc': []}
    test_record = {'loss': [], 'acc': []}

    for e in tqdm.trange(epochs, desc='Epoch', colour='pink'):
        model.train()
        epoch_loss = 0.0
        num_train_predictions_correct = 0
        num_train_predictions_total = 0

        for i, data in enumerate(tqdm.tqdm(train_loader, desc='Batch', colour='blue', leave=False), 0):
            inputs, labels = data
            inputs = inputs.to(device)
            labels = labels.to(device)

            labels = labels.unsqueeze(1)  # Add an extra dimension to the target tensor

            optimizer.zero_grad()

            # Forward, backward, optimize
            outputs = model(inputs)  # no need to transpose here
            labels = labels.squeeze(1)
            loss = loss_function(outputs, labels)
            loss.backward()
            optimizer.step()

            # Compute train accuracy
            pred_labels = (torch.sigmoid(outputs) > 0.5).float()

            num_train_predictions_correct += (pred_labels == labels).sum().item()
            num_train_predictions_total += len(pred_labels)

            # Loss
            epoch_loss += loss.item()

        epoch_loss /= len(train_loader)
        epoch_accuracy = num_train_predictions_correct / num_train_predictions_total*100
```

```
        print(f"Epoch [{e+1}/{epochs}]\tTrain_Loss: {epoch_loss:.4f}\tTrain_Accuracy: {epoch_accuracy:.2f}%")
        train_record['loss'].append(epoch_loss)
        train_record['acc'].append(epoch_accuracy)

        if test_loader:
          test_loss, test_acc = get_test_loss(model, test_loader, loss_function)
          print(f"Epoch [{e+1}/{epochs}]\tTest_Loss: {test_loss:.4f}\tTest_Accuracy: {test_acc:.2f}%")
          test_record['loss'].append(test_loss)
          test_record['acc'].append(test_acc)

    print("Training complete!")
    return train_record, test_record
```

3.2.3 Tune hyper-parameters

```
def optimize(epochs, num_channels, kernel_size, dropout, learning_rate):

    # Define the loss function
    loss_function = nn.BCEWithLogitsLoss()

    # num_inputs represents the number of features we have
    num_inputs = feature_num # Update the number of input channels to match the number of features

    # Create the TemporalConvNet instance
    tcn_model = TemporalConvNet(num_inputs, num_channels, kernel_size, dropout)
    optimizer = torch.optim.Adam(tcn_model.parameters(), lr=learning_rate)

    # Train the model
    # epochs = epochs
    train_record, _ = train(tcn_model, train_loader, loss_function, optimizer, epochs)

    return tcn_model, train_record['acc']
```

default

```
## initialize model with default values

# num_channels refers to the number of output channels in each layer of the TCN
# it determines the number of layers in TCN
epochs = 10
# Number of output channels in each layer of the TCN
default_num_channels = [32, 64]
# Kernel size for the convolutional layers
default_kernel_size = 2
# Dropout rate
default_dropout = 0.2
# Learning rate
default_learning_rate = 0.01

TCN_trained_model, acc = optimize(epochs, default_num_channels, default_kernel_size, default_dropout, default_learning_rate)
```

Epoch: 100% 10/10 [00:43<00:00, 4.15s/it]

```
Epoch [1/10]    Train_Loss: 0.2163      Train_Accuracy: 90.78%
Epoch [2/10]    Train_Loss: 0.1935      Train_Accuracy: 91.60%
Epoch [3/10]    Train_Loss: 0.1816      Train_Accuracy: 92.13%
Epoch [4/10]    Train_Loss: 0.1691      Train_Accuracy: 92.76%
Epoch [5/10]    Train_Loss: 0.1615      Train_Accuracy: 93.20%
Epoch [6/10]    Train_Loss: 0.1506      Train_Accuracy: 93.74%
Epoch [7/10]    Train_Loss: 0.1430      Train_Accuracy: 94.17%
Epoch [8/10]    Train_Loss: 0.1343      Train_Accuracy: 94.69%
Epoch [9/10]    Train_Loss: 0.1291      Train_Accuracy: 94.92%
Epoch [10/10]   Train_Loss: 0.1234      Train_Accuracy: 95.25%
Training complete!
```

tuning parameters

```
## change learning rate
lr_list = [round(value * 0.001 + 0.001, 3) for value in range(0, 10)]
result = []
for learning_rate in lr_list:
  print(f"learning rate = {learning_rate}")
```

```
model, acc = optimize(epochs, default_num_channels, default_kernel_size, default_dropout, learning_rate)
```

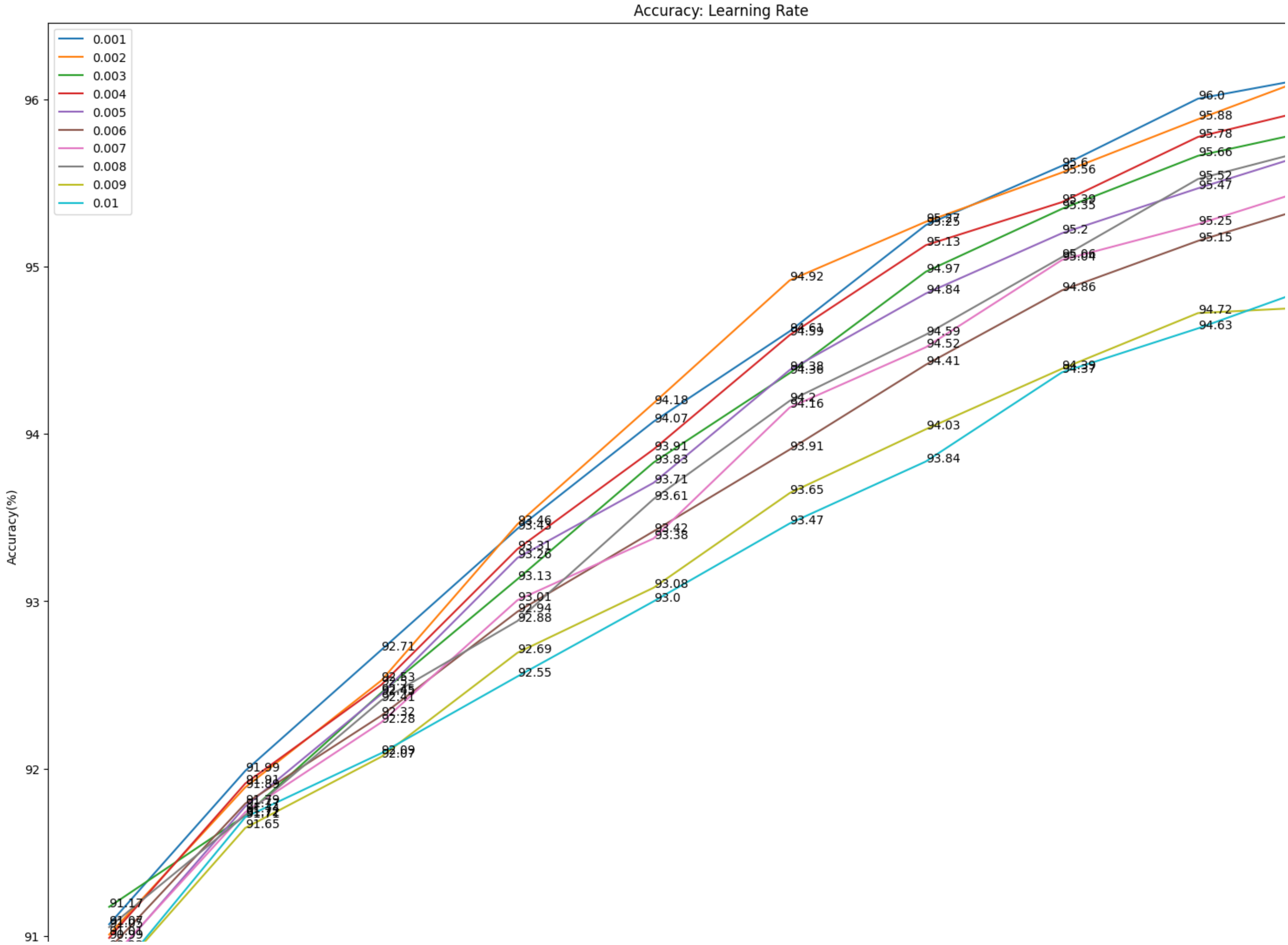
```
Epoch [8/10]    Train_Loss: 0.1119      Train_Accuracy: 95.56%
Epoch [9/10]    Train_Loss: 0.1069      Train_Accuracy: 95.88%
Epoch [10/10]   Train_Loss: 0.1003      Train_Accuracy: 96.19%
Training complete!
learning rate = 0.003
Epoch: 100%                                         10/10 [00:42<00:00, 4.25s/it]
Epoch [1/10]    Train_Loss: 0.2100      Train_Accuracy: 91.17%
Epoch [2/10]    Train_Loss: 0.1881      Train_Accuracy: 91.72%
Epoch [3/10]    Train_Loss: 0.1735      Train_Accuracy: 92.45%
Epoch [4/10]    Train_Loss: 0.1605      Train_Accuracy: 93.13%
Epoch [5/10]    Train_Loss: 0.1466      Train_Accuracy: 93.83%
Epoch [6/10]    Train_Loss: 0.1355      Train_Accuracy: 94.36%
Epoch [7/10]    Train_Loss: 0.1245      Train_Accuracy: 94.97%
Epoch [8/10]    Train_Loss: 0.1167      Train_Accuracy: 95.35%
Epoch [9/10]    Train_Loss: 0.1106      Train_Accuracy: 95.66%
Epoch [10/10]   Train_Loss: 0.1062      Train_Accuracy: 95.84%
Training complete!
learning rate = 0.004
Epoch: 100%                                         10/10 [00:42<00:00, 4.20s/it]
Epoch [1/10]    Train_Loss: 0.2088      Train_Accuracy: 90.99%
Epoch [2/10]    Train_Loss: 0.1873      Train_Accuracy: 91.91%
Epoch [3/10]    Train_Loss: 0.1735      Train_Accuracy: 92.50%
Epoch [4/10]    Train_Loss: 0.1591      Train_Accuracy: 93.31%
Epoch [5/10]    Train_Loss: 0.1460      Train_Accuracy: 93.91%
Epoch [6/10]    Train_Loss: 0.1334      Train_Accuracy: 94.59%
Epoch [7/10]    Train_Loss: 0.1230      Train_Accuracy: 95.13%
Epoch [8/10]    Train_Loss: 0.1168      Train_Accuracy: 95.39%
Epoch [9/10]    Train_Loss: 0.1087      Train_Accuracy: 95.78%
Epoch [10/10]   Train_Loss: 0.1055      Train_Accuracy: 95.97%
Training complete!
learning rate = 0.005
Epoch: 100%                                         10/10 [00:42<00:00, 4.17s/it]
Epoch [1/10]    Train_Loss: 0.2127      Train_Accuracy: 90.84%
Epoch [2/10]    Train_Loss: 0.1895      Train_Accuracy: 91.77%
Epoch [3/10]    Train_Loss: 0.1756      Train_Accuracy: 92.45%
Epoch [4/10]    Train_Loss: 0.1600      Train_Accuracy: 93.26%
Epoch [5/10]    Train_Loss: 0.1500      Train_Accuracy: 93.71%
Epoch [6/10]    Train_Loss: 0.1390      Train_Accuracy: 94.38%
Epoch [7/10]    Train_Loss: 0.1293      Train_Accuracy: 94.84%
Epoch [8/10]    Train_Loss: 0.1227      Train_Accuracy: 95.20%
Epoch [9/10]    Train_Loss: 0.1170      Train_Accuracy: 95.47%
Epoch [10/10]   Train_Loss: 0.1111      Train_Accuracy: 95.72%
Training complete!
learning rate = 0.006
Epoch: 100%                                         10/10 [00:42<00:00, 4.25s/it]
Epoch [1/10]    Train_Loss: 0.2099      Train_Accuracy: 90.92%
Epoch [2/10]    Train_Loss: 0.1889      Train_Accuracy: 91.79%
Epoch [3/10]    Train_Loss: 0.1758      Train_Accuracy: 92.32%
Epoch [4/10]    Train_Loss: 0.1662      Train_Accuracy: 92.94%
Epoch [5/10]    Train_Loss: 0.1548      Train_Accuracy: 93.42%
Epoch [6/10]    Train_Loss: 0.1445      Train_Accuracy: 93.91%
Epoch [7/10]    Train_Loss: 0.1346      Train_Accuracy: 94.41%
Epoch [8/10]    Train_Loss: 0.1278      Train_Accuracy: 94.86%
```

```python
import matplotlib.pyplot as plt
fig = plt.figure(figsize =(20, 15))

e_num = [value for value in range(0, 10)]

for i in range(10):
    plt.plot(np.array(e_num), np.array(result[i]), label = lr_list[i])
    for x,y in zip(np.array(e_num), np.array(result[i])):
        plt.annotate(str(round(y,2)),xy=(x,y))

plt.xlabel('Number of Epoch')
plt.ylabel('Accuracy(%)')
plt.title('Accuracy: Learning Rate')
plt.legend()
plt.show()
```



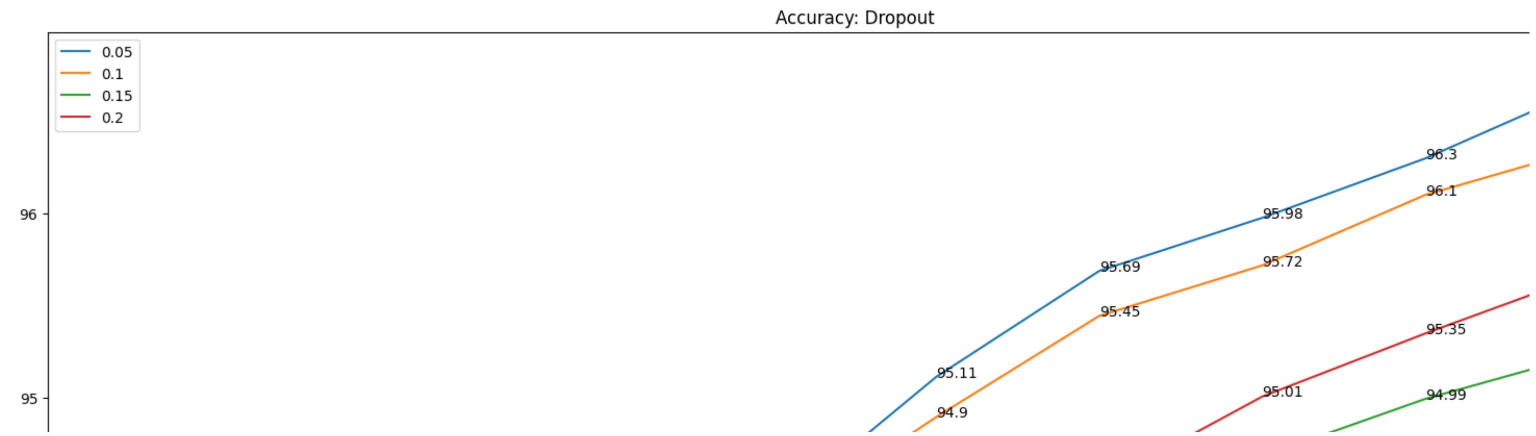

```
# change dropout
dropout_list = [0.05, 0.1, 0.15, 0.2]
dropout_acc = []
for d in dropout_list:
  print(f"dropout = {d}")
  model, acc = optimize(epochs, default_num_channels, default_kernel_size, d, default_learning_rate)
  dropout_acc.append(acc)
```

```
dropout = 0.05
Epoch: 100%                                    10/10 [00:42<00:00, 4.20s/it]
Epoch [1/10]    Train_Loss: 0.2102      Train_Accuracy: 90.90%
Epoch [2/10]    Train_Loss: 0.1851      Train_Accuracy: 91.83%
Epoch [3/10]    Train_Loss: 0.1715      Train_Accuracy: 92.50%
Epoch [4/10]    Train_Loss: 0.1530      Train_Accuracy: 93.52%
Epoch [5/10]    Train_Loss: 0.1366      Train_Accuracy: 94.38%
Epoch [6/10]    Train_Loss: 0.1217      Train_Accuracy: 95.11%
Epoch [7/10]    Train_Loss: 0.1109      Train_Accuracy: 95.69%
Epoch [8/10]    Train_Loss: 0.1026      Train_Accuracy: 95.98%
Epoch [9/10]    Train_Loss: 0.0947      Train_Accuracy: 96.30%
Epoch [10/10]   Train_Loss: 0.0882      Train_Accuracy: 96.69%
Training complete!
dropout = 0.1
Epoch: 100%                                    10/10 [00:42<00:00, 4.26s/it]
Epoch [1/10]    Train_Loss: 0.2088      Train_Accuracy: 90.99%
Epoch [2/10]    Train_Loss: 0.1846      Train_Accuracy: 91.90%
Epoch [3/10]    Train_Loss: 0.1717      Train_Accuracy: 92.50%
Epoch [4/10]    Train_Loss: 0.1540      Train_Accuracy: 93.46%
Epoch [5/10]    Train_Loss: 0.1388      Train_Accuracy: 94.31%
Epoch [6/10]    Train_Loss: 0.1269      Train_Accuracy: 94.90%
Epoch [7/10]    Train_Loss: 0.1164      Train_Accuracy: 95.45%
Epoch [8/10]    Train_Loss: 0.1078      Train_Accuracy: 95.72%
Epoch [9/10]    Train_Loss: 0.1010      Train_Accuracy: 96.10%
Epoch [10/10]   Train_Loss: 0.0951      Train_Accuracy: 96.36%
Training complete!
dropout = 0.15
```

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize =(20, 15))

for i in range(4):
        plt.plot(np.array(e_num), np.array(dropout_acc[i]), label = dropout_list[i])
        for x,y in zip(np.array(e_num), np.array(dropout_acc[i])):
            plt.annotate(str(round(y,2)),xy=(x,y))

plt.xlabel('Number of Epoch')
plt.ylabel('Accuracy(%)')
plt.title('Accuracy: Dropout')
plt.legend()
plt.show()
```




```
# num_channels
num_channels_list = [[32], [32, 64]]
acc_k = []
for c in range(len(num_channels_list)):
  print(f"channels_list = {c}")
  model, acc = optimize(epochs, num_channels_list[c], default_kernel_size, default_dropout, default_learning_rate)
  acc_k.append(acc)
```

```
channels_list = 0
Epoch: 100%                                    10/10 [00:32<00:00, 3.31s/it]
Epoch [1/10]    Train_Loss: 0.2308     Train_Accuracy: 90.44%
Epoch [2/10]    Train_Loss: 0.2042     Train_Accuracy: 91.45%
Epoch [3/10]    Train_Loss: 0.1916     Train_Accuracy: 92.08%
Epoch [4/10]    Train_Loss: 0.1756     Train_Accuracy: 92.86%
Epoch [5/10]    Train_Loss: 0.1610     Train_Accuracy: 93.64%
Epoch [6/10]    Train_Loss: 0.1513     Train_Accuracy: 94.11%
Epoch [7/10]    Train_Loss: 0.1425     Train_Accuracy: 94.58%
Epoch [8/10]    Train_Loss: 0.1345     Train_Accuracy: 95.02%
Epoch [9/10]    Train_Loss: 0.1298     Train_Accuracy: 95.21%
Epoch [10/10]   Train_Loss: 0.1262     Train_Accuracy: 95.38%
Training complete!
channels_list = 1
Epoch: 100%                                    10/10 [00:42<00:00, 4.29s/it]
Epoch [1/10]    Train_Loss: 0.2165     Train_Accuracy: 90.66%
Epoch [2/10]    Train_Loss: 0.1916     Train_Accuracy: 91.72%
Epoch [3/10]    Train_Loss: 0.1816     Train_Accuracy: 92.08%
Epoch [4/10]    Train_Loss: 0.1694     Train_Accuracy: 92.69%
Epoch [5/10]    Train_Loss: 0.1602     Train_Accuracy: 93.22%
Epoch [6/10]    Train_Loss: 0.1474     Train_Accuracy: 93.96%
Epoch [7/10]    Train_Loss: 0.1370     Train_Accuracy: 94.48%
Epoch [8/10]    Train_Loss: 0.1306     Train_Accuracy: 94.78%
Epoch [9/10]    Train_Loss: 0.1233     Train_Accuracy: 95.11%
Epoch [10/10]   Train_Loss: 0.1181     Train_Accuracy: 95.37%
Training complete!
```

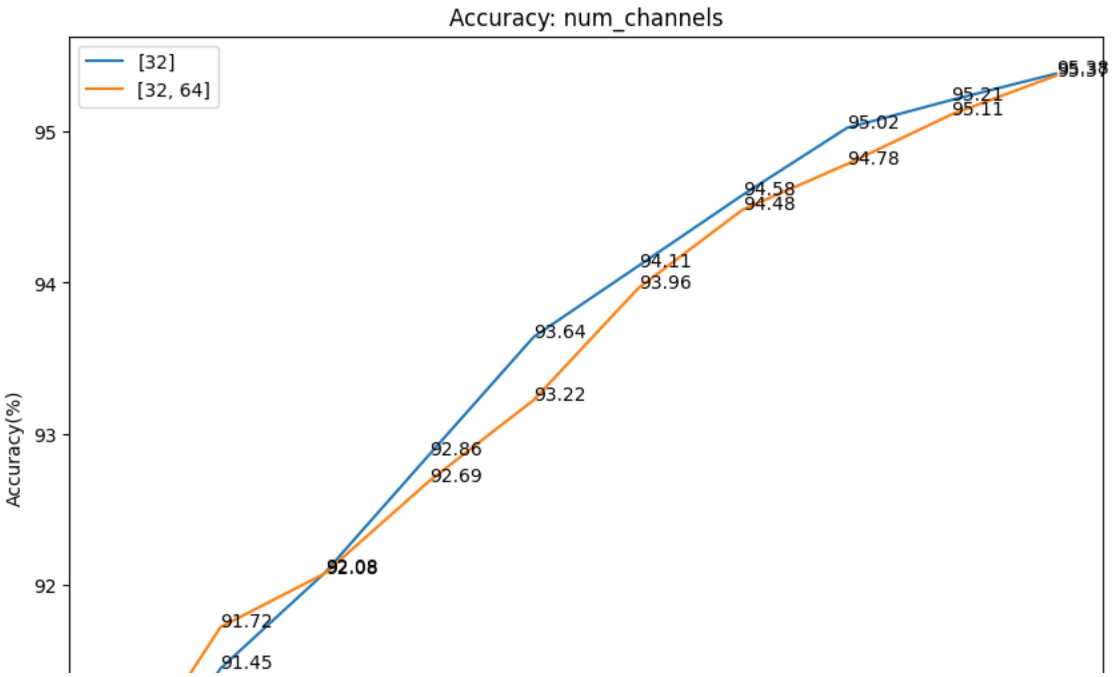
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10, 8))

for i in range(2):
    ax.plot(np.array(e_num), np.array(acc_k[i]), label=num_channels_list[i])
    for x, y in zip(np.array(e_num), np.array(acc_k[i])):
        ax.annotate(str(round(y, 2)), xy=(x, y))

ax.set_xlabel('Number of Epoch')
ax.set_ylabel('Accuracy(%)')
ax.set_title('Accuracy: num_channels')
ax.legend()

plt.show()
```



cross validation

```
import itertools
from sklearn.model_selection import KFold
from torch.utils.data import DataLoader, SubsetRandomSampler
```

```
def create_param_combs(params):
    # Cite: https://stackoverflow.com/questions/44887695/execute-function-on-all-possible-combinations-of-parameters
    return [dict(zip(params, vals)) for vals in itertools.product(*params.values())]
```

```
# Cite: https://medium.com/dataseries/k-fold-cross-validation-with-pytorch-and-sklearn-d094aa00105f
def cross_val(tune_params, dataset, k):
    param_combs = create_param_combs(tune_params)

    opt_params = {}
    opt_loss = np.Inf

    skf = KFold(n_splits=k, shuffle=True)

    for comb in param_combs:
        print('Cross Validation with Params = {0}'.format(comb))
        loss_k = []
        for i, (train_index, val_index) in enumerate(skf.split(np.arange(len(dataset)))):
            train_sampler = SubsetRandomSampler(train_index)
            val_sampler = SubsetRandomSampler(val_index)

            batch_size = comb['batch_size']
            train_loader = DataLoader(dataset, batch_size=batch_size, sampler=train_sampler)
            val_loader = DataLoader(dataset, batch_size=batch_size, sampler=val_sampler)

            # TCN
            tcn_model = TemporalConvNet(feature_num, comb['num_channels'], comb['kernel_size'], comb['dropout'])

            # Binary cross-entropy loss for binary classification
            loss_function = nn.BCEWithLogitsLoss()

            # Optimizer
            optimizer = torch.optim.Adam(tcn_model.parameters(), lr=comb['lr'])
            epochs = 20
            train_record, test_record = train(tcn_model, train_loader, loss_function, optimizer, epochs, val_loader)

            loss_k.append(test_record['loss'][-1])

        mean_loss = np.mean(loss_k)
        if mean_loss < opt_loss:
            opt_params = comb
            opt_loss = mean_loss

    return opt_params
```

```
params = {
    'lr': [0.01, 0.001, 0.0001],
    'batch_size': [64],
    'num_channels': [[32, 64]],
    'kernel_size': [2],
    'dropout': [0.1, 0.15, 0.2, 0.25, 0.3]
}

opt_params = cross_val(params, train_data, 3)
```

```
Epoch [7/20]    Train_Loss: 0.1848      Train_Accuracy: 92.18%
Epoch [7/20]    Test_Loss: 0.1864       Test_Accuracy: 91.87%
Epoch [8/20]    Train_Loss: 0.1809      Train_Accuracy: 92.24%
Epoch [8/20]    Test_Loss: 0.1838       Test_Accuracy: 91.97%
Epoch [9/20]    Train_Loss: 0.1775      Train_Accuracy: 92.36%
Epoch [9/20]    Test_Loss: 0.1815       Test_Accuracy: 92.19%
Epoch [10/20]   Train_Loss: 0.1745      Train_Accuracy: 92.70%
Epoch [10/20]   Test_Loss: 0.1793       Test_Accuracy: 92.23%
Epoch [11/20]   Train_Loss: 0.1696      Train_Accuracy: 92.89%
Epoch [11/20]   Test_Loss: 0.1769       Test_Accuracy: 92.31%
Epoch [12/20]   Train_Loss: 0.1676      Train_Accuracy: 92.91%
Epoch [12/20]   Test_Loss: 0.1772       Test_Accuracy: 92.42%
Epoch [13/20]   Train_Loss: 0.1630      Train_Accuracy: 93.28%
Epoch [13/20]   Test_Loss: 0.1722       Test_Accuracy: 92.59%
Epoch [14/20]   Train_Loss: 0.1605      Train_Accuracy: 93.30%
Epoch [14/20]   Test_Loss: 0.1704       Test_Accuracy: 92.75%
Epoch [15/20]   Train_Loss: 0.1569      Train_Accuracy: 93.45%
Epoch [15/20]   Test_Loss: 0.1689       Test_Accuracy: 92.81%
Epoch [16/20]   Train_Loss: 0.1537      Train_Accuracy: 93.64%
Epoch [16/20]   Test_Loss: 0.1675       Test_Accuracy: 92.93%
Epoch [17/20]   Train_Loss: 0.1507      Train_Accuracy: 93.87%
Epoch [17/20]   Test_Loss: 0.1657       Test_Accuracy: 93.05%
Epoch [18/20]   Train_Loss: 0.1467      Train_Accuracy: 94.01%
Epoch [18/20]   Test_Loss: 0.1641       Test_Accuracy: 93.11%
Epoch [19/20]   Train_Loss: 0.1434      Train_Accuracy: 94.16%
Epoch [19/20]   Test_Loss: 0.1620       Test_Accuracy: 93.13%
Epoch [20/20]   Train_Loss: 0.1403      Train_Accuracy: 94.27%
Epoch [20/20]   Test_Loss: 0.1607       Test_Accuracy: 93.33%
Training complete!
```

Epoch: 100% 20/20 [01:29<00:00, 4.48s/it]

```
Epoch [1/20]    Train_Loss: 0.3011      Train_Accuracy: 89.58%
Epoch [1/20]    Test_Loss: 0.2228       Test_Accuracy: 89.62%
Epoch [2/20]    Train_Loss: 0.2273      Train_Accuracy: 90.25%
Epoch [2/20]    Test_Loss: 0.2018       Test_Accuracy: 91.08%
Epoch [3/20]    Train_Loss: 0.2136      Train_Accuracy: 91.18%
Epoch [3/20]    Test_Loss: 0.1959       Test_Accuracy: 91.45%
Epoch [4/20]    Train_Loss: 0.2057      Train_Accuracy: 91.49%
Epoch [4/20]    Test_Loss: 0.1904       Test_Accuracy: 91.48%
Epoch [5/20]    Train_Loss: 0.1995      Train_Accuracy: 91.63%
Epoch [5/20]    Test_Loss: 0.1867       Test_Accuracy: 91.61%
Epoch [6/20]    Train_Loss: 0.1946      Train_Accuracy: 91.87%
Epoch [6/20]    Test_Loss: 0.1834       Test_Accuracy: 91.73%
Epoch [7/20]    Train_Loss: 0.1915      Train_Accuracy: 91.98%
Epoch [7/20]    Test_Loss: 0.1807       Test_Accuracy: 92.01%
Epoch [8/20]    Train_Loss: 0.1876      Train_Accuracy: 92.16%
Epoch [8/20]    Test_Loss: 0.1786       Test_Accuracy: 91.90%
Epoch [9/20]    Train_Loss: 0.1839      Train_Accuracy: 92.29%
Epoch [9/20]    Test_Loss: 0.1765       Test_Accuracy: 92.11%
Epoch [10/20]   Train_Loss: 0.1813      Train_Accuracy: 92.46%
Epoch [10/20]   Test_Loss: 0.1745       Test_Accuracy: 92.17%
Epoch [11/20]   Train_Loss: 0.1777      Train_Accuracy: 92.66%
Epoch [11/20]   Test_Loss: 0.1714       Test_Accuracy: 92.33%
Epoch [12/20]   Train_Loss: 0.1739      Train_Accuracy: 92.84%
Epoch [12/20]   Test_Loss: 0.1699       Test_Accuracy: 92.47%
Epoch [13/20]   Train_Loss: 0.1699      Train_Accuracy: 93.04%
Epoch [13/20]   Test_Loss: 0.1677       Test_Accuracy: 92.56%
Epoch [14/20]   Train_Loss: 0.1662      Train_Accuracy: 93.31%
Epoch [14/20]   Test_Loss: 0.1662       Test_Accuracy: 92.68%
Epoch [15/20]   Train_Loss: 0.1626      Train_Accuracy: 93.58%
Epoch [15/20]   Test_Loss: 0.1635       Test_Accuracy: 92.86%
Epoch [16/20]   Train_Loss: 0.1602      Train_Accuracy: 93.54%
Epoch [16/20]   Test_Loss: 0.1626       Test_Accuracy: 92.95%
Epoch [17/20]   Train_Loss: 0.1554      Train_Accuracy: 93.84%
Epoch [17/20]   Test_Loss: 0.1597       Test_Accuracy: 93.26%
Epoch [18/20]   Train_Loss: 0.1519      Train_Accuracy: 93.98%
Epoch [18/20]   Test_Loss: 0.1576       Test_Accuracy: 93.33%
Epoch [19/20]   Train_Loss: 0.1484      Train_Accuracy: 94.10%
Epoch [19/20]   Test_Loss: 0.1564       Test_Accuracy: 93.46%
Epoch [20/20]   Train_Loss: 0.1462      Train_Accuracy: 94.19%
Epoch [20/20]   Test_Loss: 0.1538       Test_Accuracy: 93.62%
Training complete!
```

Epoch: 100% 20/20 [01:29<00:00, 4.52s/it]

```
Epoch [1/20]    Train_Loss: 0.2984      Train_Accuracy: 89.08%
Epoch [1/20]    Test_Loss: 0.2118       Test_Accuracy: 91.26%
Epoch [2/20]    Train_Loss: 0.2161      Train_Accuracy: 91.09%
Epoch [2/20]    Test_Loss: 0.1979       Test_Accuracy: 91.45%
Epoch [3/20]    Train_Loss: 0.2041      Train_Accuracy: 91.41%
Epoch [3/20]    Test_Loss: 0.1917       Test_Accuracy: 91.66%
Epoch [4/20]    Train_Loss: 0.1956      Train_Accuracy: 91.69%
Epoch [4/20]    Test_Loss: 0.1885       Test_Accuracy: 91.63%
Epoch [5/20]    Train_Loss: 0.1916      Train_Accuracy: 91.84%
Epoch [5/20]    Test_Loss: 0.1853       Test_Accuracy: 91.86%
Epoch [6/20]    Train_Loss: 0.1868      Train_Accuracy: 91.96%
Epoch [6/20]    Test_Loss: 0.1827       Test_Accuracy: 92.09%
Epoch [7/20]    Train_Loss: 0.1831      Train_Accuracy: 92.23%
Epoch [7/20]    Test_Loss: 0.1811       Test_Accuracy: 92.11%
```