Ahmed Awadallah

Dense Neural Networks

Introduction

The background for this case study is essentially nonexistent. We are given a dataset and told to create a model that saves them the most money. The rules for the money lost is that a false positive incurs a 35$ loss, a false negative incurs a 15$ loss, and True Positive/True Negative has a 0$ loss. Client doesn't care about feature interpretation, just performance.

Upon inspection of the data it's clear this is a binary classification problem. The dataset contains 160000 observations, 49 features, and one binary target.

Methods

The dataset is a single file that isn't large so it's completely loaded into the script.

The following was performed on the entire dataset.

Inspection of the data reveals three classes of data. There is data that is clearly numeric so those will remain numeric. There is data that is clearly categorical. Lastly there is data that is categorical but should be numeric.

The numeric data does not need any adjustments so is left as is during this stage.

Some of the categorical features have improper classes. For example x29 has a class labeled as Dev. The other classes for this feature are January, Feb, and Apr to name a few. It's clear this feature refers to months of the year so dev is most likely referring to December. This was corrected. Additionally the format was inconsistent, some months had their full name while others were abbreviations. The final format was to make the months have their full name. x24 had spelling errors so those were fixed as well as all classes starting with a capital letter. x30 had all classes start with a capital letter instead.

x32 and x37 fell into the "data that is categorical but should be numeric" category. Essentially they all encoded numeric data but it was encoded with a string. x32 had percentage information, 0.1% for example. The percentage sign was removed and the type was converted to float. There was one odd case where there was a -0.0% and 0.0% class. These two classes were coalesced into the singular class 0.0 and then converted to float. x37 had dollar information, $1000 for example. The dollar sign was removed and then the type was converted to float.

Overall plan moving forward is to do 10 fold cross validation with a neural network model. Data imputation, scaling, and one hot encoding is fitted and performed on the training data and then performed but not fitted on the test data for each fold.

The dataset contained NA values for every class. The amount was miniscule, no class had more than 0.03% of the data missing. The numeric variables were imputed using the mean and the categorical variables were imputed using the most frequent class.

Numeric data was scaled as the model being used is sensitive to scale.

Categorical data was one hot encoded as the model cannot interpret string data.

Data is transferred to a dataset object and then a data loader object. Dataloaders shuffle the data every epoch during training. The test is not shuffled as it is not necessary.

Model is a custom neural network whose details are specified in figure 1 under the Architecture section below. Neural network was chosen as the client wanted the best performance and did not care about interpretation. We have a large enough dataset for a neural network which provides the performance the client wants at the cost of interpretation.

The loss function is binary cross entropy loss as this is a binary classification problem. The weights for the classes are not even. Since it costs more to have a false positive than a false negative so more weight is given to predicting the positive class. Specifically the weight increase is the ratio of dollar cost for false positive over dollar cost for false negative which was 2.33.

The model optimizer is the adam optimizer.

A multi step learning rate scheduler is also used with the model. The learning is dropped twice by a factor of 10 at epoch 30 and epoch 50, overall the model runs for 100 epochs.

The model is trained with early stopping. If the model does not improve for 50 iterations training will stop and the model has completed training.

After training is complete for a fold the model is then used to output predictions for the out of fold data. Every folds out of fold prediction is added to this growing set.

Accuracy information is collected for each fold.

The first fold has their classification report and loss plots saved for later analysis.

Once all the folds are complete an average score is calculated, an overall out of fold classification report, and an overall out of fold dollar prediction score is computed.

Architecture

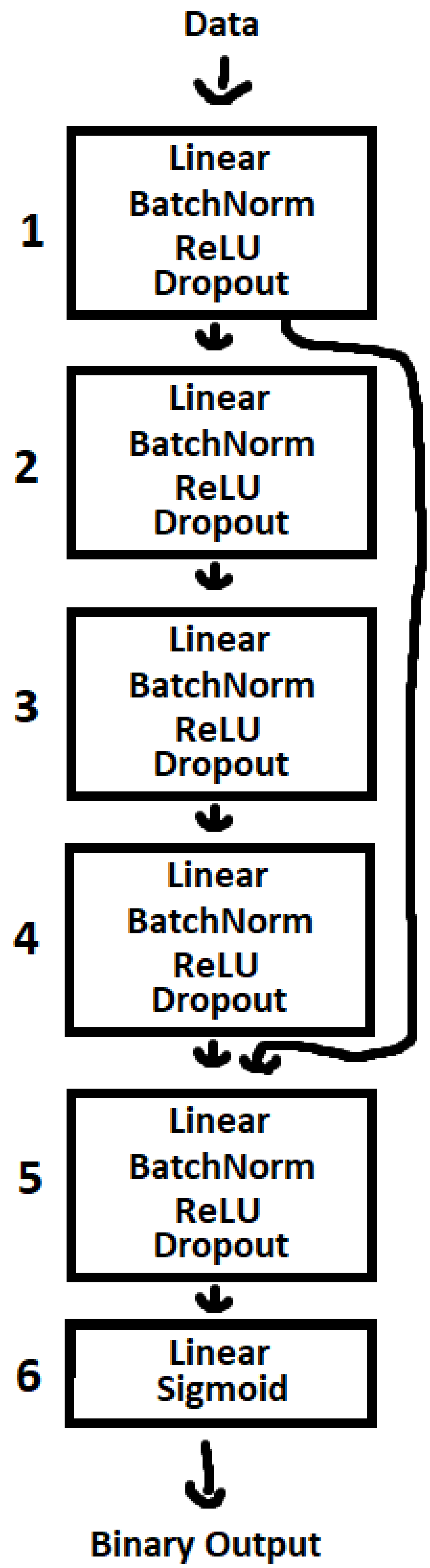

Figure 1: Neural Network Architecture

Architecture is overall 6 layers deep. The first 4 layers perform the same calculation of Linear > BatchNorm > ReLU > Dropout of 50%. Each layer reduces the hidden weights by a factor of 4. The first layer starts with 2048 weights.

The fifth layer is special as it acts as a skip connection layer so its weight size is the same as layer1 + layer4 and outputs the same hidden layer size as layer 4.

The sixth layer is the output layer which is just a linear layer on a single neuron with a sigmoid activation. This structure is required for the task of binary classification.

The linear segments are required in neural networks, the BatchNorm segments speed up the learning process, the ReLU activations in the hidden layers provide nonlinearity in the model, and the dropout layers prevent the model from overfitting.

Results

Train Loss Plot

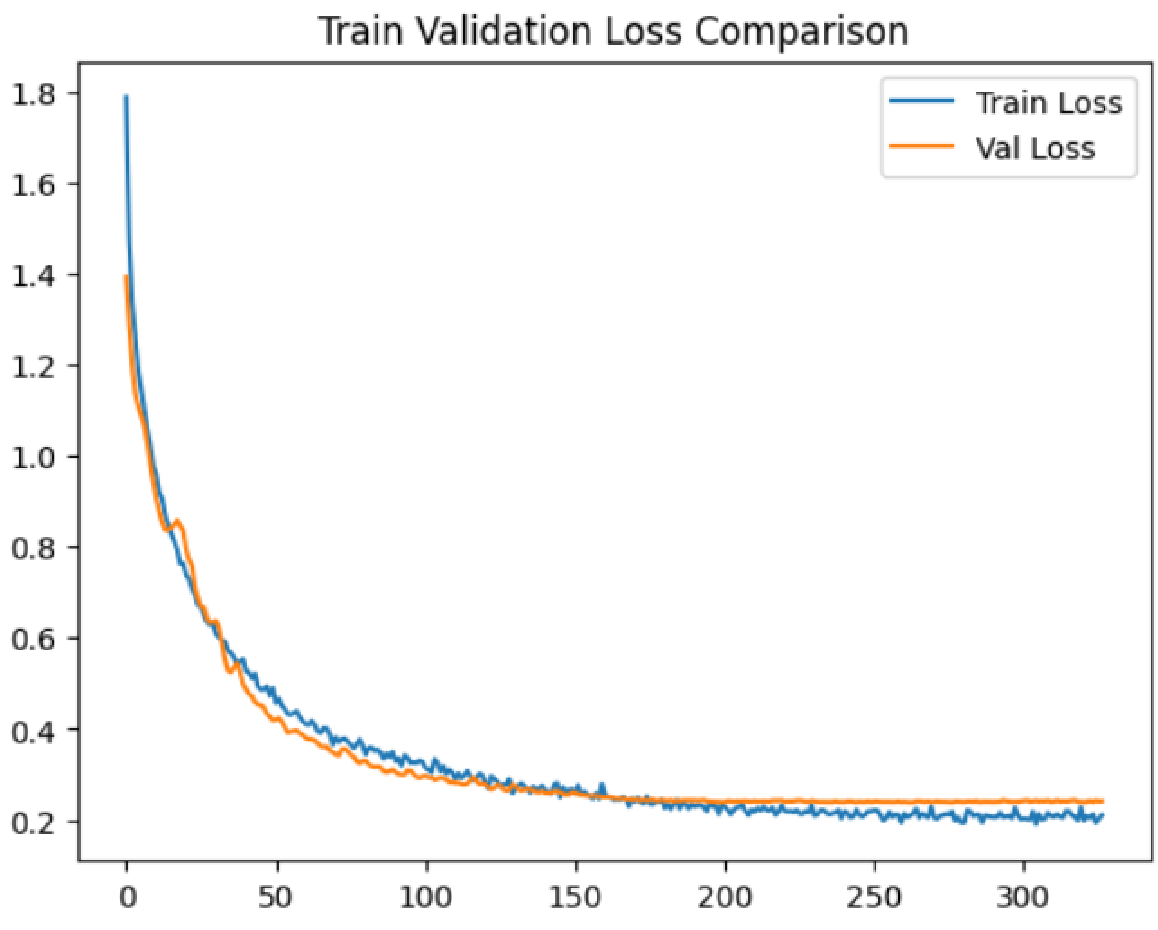

Figure 2: Loss Plot

This is a plot of a single folds train and validation losses. Overall good training curve that shows that the model learns quickly and maintains enough generalization for the validation curve to level out but not too much such that the training model overfits. Notably interesting behavior is that the validation curve actually has a lower loss than the training curve early on in training.

Out Of Fold Classification Report

```
OOF Classification Report
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
              precision    recall  f1-score   support

           0       0.97      0.98      0.97     95803
           1       0.97      0.96      0.96     64197

    accuracy                           0.97    160000
   macro avg       0.97      0.97      0.97    160000
weighted avg       0.97      0.97      0.97    160000
```

Figure 3: Out of Fold Classification Report

Above is the out of fold classification report for all folds. Overall the model does exceptionally well for both classes without much tradeoff for one or the other despite weighting the positive class more for the loss function. This likely indicates the classes being largely different but there is a small overlap region where it is ambiguous whether an observation belongs to either class 0 or class 1.

Out Of Fold USD Score

```
OOF Dollar Score
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model OOF Dollar Score: $117010
```

Figure 4: Dollar Score

Above is the dollar score my out of fold predictions got.

Conclusion

Overall the model had excellent performance. One thing that must be mentioned is that there is some minor variation in the results for a full run. I did a couple runs and picked the best score I personally saw. Since the client just wanted the best possible model trained on the data this felt like the optimal choice compared to the frequentist approach of doing multiple runs and giving a mean score with confidence intervals.

Code

Appendix A: Script

```python
import os
import sys
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report
import numpy as np
import torch
from torch import nn
from torch.utils.data import Dataset, DataLoader
from torch.optim.lr_scheduler import MultiStepLR
from sklearn.model_selection import KFold
from sklearn.preprocessing import OneHotEncoder


# Agnostic Device
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"


class CS7Dataset(Dataset):
    # Dataset class necessary for Dataloader
    def __init__(self, set):
        x = set[0]
        y = set[1]
        self.x = torch.tensor(x.values, dtype=torch.float32)
        self.y = torch.tensor(y.values, dtype=torch.float32)

    def __len__(self):
        return len(self.y)

    def __getitem__(self, idx):
        return self.x[idx], self.y[idx]
```

```python
class CS7NET(nn.Module):
    def __init__(self, input_features, hidden_feature_seed):
        super().__init__()

        l1_hidden = hidden_feature_seed
        self.layer1 = nn.Sequential(
            nn.Linear(input_features, l1_hidden),
            nn.BatchNorm1d(l1_hidden),
            nn.ReLU(),
            nn.Dropout(0.5)
        )

        l2_hidden = int(hidden_feature_seed/(2**2))
        self.layer2 = nn.Sequential(
            nn.Linear(l1_hidden, l2_hidden),
            nn.BatchNorm1d(l2_hidden),
            nn.ReLU(),
            nn.Dropout(0.5)
        )

        l3_hidden = int(l2_hidden/(2**2))
        self.layer3 = nn.Sequential(
            nn.Linear(l2_hidden, l3_hidden),
            nn.BatchNorm1d(l3_hidden),
            nn.ReLU(),
            nn.Dropout(0.5)
        )

        l4_hidden = int(l3_hidden/(2**2))
        self.layer4 = nn.Sequential(
            nn.Linear(l3_hidden, l4_hidden),
            nn.BatchNorm1d(l4_hidden),
            nn.ReLU(),
            nn.Dropout(0.5)
        )
```

```python
        self.skip_layer = nn.Sequential(
            nn.Linear(l4_hidden+l1_hidden, l4_hidden),
            nn.BatchNorm1d(l4_hidden),
            nn.ReLU(),
            nn.Dropout(0.5)
        )

        output_size = 1  # binary predictor
        self.output = nn.Sequential(
            nn.Linear(l4_hidden, output_size),
            nn.Sigmoid()
        )

    def forward(self, x):
        l1 = self.layer1(x)
        l2 = self.layer2(l1)
        l3 = self.layer3(l2)
        l4 = self.layer4(l3)
        l1l4 = torch.cat((l4, l1), 1)
        skip = self.skip_layer(l1l4)
        output = self.output(skip)

        return output


class UnexpectedFileStructureError(Exception):
    def __init__(self):
        msg = "This script expects a file path to a directory that contains "
        msg += "CS7 Dataset. The CS7 dataset is a "
        msg += "single csv labelled as final_project.csv "
        msg += "Did you specify a folder path that "
        msg += "contains this file?"
        super().__init__(msg)
```

```python
class MissingInputError(Exception):
    def __init__(self):
        msg = "This script expects atleast one input, a file directory that "
        msg += "contains the CS7 dataset. You input nothing. "
        msg += "Did you forget to add an argument with the script?"
        super().__init__(msg)


class ExcessiveInputError(Exception):
    def __init__(self):
        msg = "This script expects atleast one input and at most 2."
        msg += " The first argument is the file directory containing the data"
        msg += " and the second optional one is a flag specifying if you want"
        msg += " to train on all the data or a subset"
        msg += " You called this script with more than two arguments."
        msg += " Only use one or two and try again."
        super().__init__(msg)


class UnexpectedFlagError(Exception):
    def __init__(self):
        msg = "Flag input can only have a value of 0 or 1. Script received"
        msg += " something unexpected instead."
        super().__init__(msg)


class NonexistantFolderError(Exception):
    def __init__(self):
        msg = "Input folder does not exist"
        super().__init__(msg)

```

```python
def check_correct_file_structure(inputs):
    data_direc = inputs[1]
    expected_files = ["final_project.csv"]
    if os.path.exists(data_direc):
        for root, dirs, files in os.walk(data_direc):
            if files == []:
                raise UnexpectedFileStructureError
            for f in files:
                if f not in expected_files:
                    print(f)
                    raise UnexpectedFileStructureError
    else:
        raise NonexistantFolderError


def check_correct_flag_input_val(inputs):
    flag = inputs[2]
    if (flag != "1") and (flag != "0"):
        raise UnexpectedFlagError


def check_valid_inputs(inputs):
    if len(inputs) < 2:
        raise MissingInputError
    elif len(inputs) > 3:
        raise ExcessiveInputError


def load_data(inputs):
    data_direc = inputs[1]
    for root, dirs, files in os.walk(data_direc):
        for f in files:
            file_path = os.path.join(root, f)
            raw_data = pd.read_csv(file_path)
    return raw_data
```

```python
def get_flag(inputs):
    flag = inputs[2]
    if flag == "1":
        flag = 1
    elif flag == "0":
        flag = 0
    return flag


def train_val_test_split(x, y, test_size=0.1):
    x_train, x_val_test, y_train, y_val_test = train_test_split(
        x,
        y,
        test_size=(test_size*2))
    x_val, x_test, y_val, y_test = train_test_split(
        x_val_test,
        y_val_test,
        test_size=0.5)
    data_splits = {
        "train": (x_train, y_train),
        "val": (x_val, y_val),
        "test": (x_test, y_test)
    }
    return data_splits


def log_section(log_file, title="No Title", content="No Content"):
    log_file.write(title + "\n")
    log_file.write("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")
    log_file.write(content + "\n")
    log_file.write("\n")
```

```python
def train_model(model_package,
                train_loader,
                val_loader,
                num_epochs=1,
                early_stop_criterion=10,
                report_modifier=0.05,
                record_modifier=0.05,
                model_save_name="model.pth"):
    print("Starting model training...")

    # Important Variable Setup
    model = model_package[0]
    criterion = model_package[1]
    optimizer = model_package[2]
    scheduler = model_package[3]
    device = next(model.parameters()).device.type
    train_losses = []
    train_loss = 0
    val_losses = []
    val_loss = 0
    best_val_loss = 0
    early_stop_test_num = 0
    early_stop_criterion_met = False
    first = True
    n_total_steps = len(train_loader)
    report_freq = int(n_total_steps*report_modifier)
    if report_freq == 0:
        report_freq = 1
    record_freq = int(n_total_steps*record_modifier)
    if record_freq == 0:
        record_freq = 1
```

```python
    # Train Loop
    for epoch in range(num_epochs):
        if early_stop_criterion_met:
            break
        for i, (observations, labels) in enumerate(train_loader):
            if early_stop_criterion_met:
                break
            # This may be unnecesarily set but is here to avoid accidental
            # bugs where it is not set
            model.train()

            # Data to device
            observations = observations.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(observations)
            loss = criterion(outputs, labels)

            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # Loss Accumulation
            train_loss += loss.item()

            # Report Progress
            if (i+1) % report_freq == 0:
                progress_str = f'Epoch [{epoch+1}/{num_epochs}], '
                progress_str += f'Step [{i+1}/{n_total_steps}],'
                progress_str += f'Loss: {loss.item():.4f}'
                print(progress_str)

```

```python
            # Pass through validation set and record Val Loss and
            # Accumulated Train Loss Reset both when done
            if (i+1) % record_freq == 0:
                train_loss /= record_freq
                train_losses.append(train_loss)
                train_loss = 0

                val_loss = 0
                for i, (observations, labels) in enumerate(val_loader):
                    observations = observations.to(device)
                    labels = labels.to(device)
                    model.eval()
                    with torch.inference_mode():
                        outputs = model(observations)
                        loss = criterion(outputs, labels)
                        val_loss += loss.item()

                val_loss /= len(val_loader)
                val_losses.append(val_loss)

                if first:
                    best_val_loss = val_loss
                    first = False
                elif best_val_loss >= val_loss:
                    early_stop_test_num = 0
                    best_val_loss = val_loss
                elif best_val_loss < val_loss:
                    early_stop_test_num += 1
                    if early_stop_test_num == early_stop_criterion:
                        print("Training stopping early...")
                        early_stop_criterion_met = True
        scheduler.step()

    print('Finished Training, Saving model...')
    torch.save(model.state_dict(), model_save_name)
```

```python
def reformat_data(raw_df):
    # Explicitly Define Feature Types based on EDA
    targ_ftr = ["y"]
    categ_ftrs = ["x24", "x29", "x30"]
    bad_categ_ftrs = ["x32", "x37"]
    cts_ftrs = []
    for ftr in raw_df.columns:
        if (
                (ftr not in targ_ftr) and
                (ftr not in categ_ftrs) and
                (ftr not in bad_categ_ftrs)
                ):
            cts_ftrs.append(ftr)

    # Reformat improper feature values
    # x24 Replace euorpe with europe with grammer adjustments for the others
    raw_df['x24'] = raw_df['x24'].replace(
        ['euorpe', "asia", "america"],
        ['Europe', "Asia", "America"])
    # x29 Adjust Dev and make naming consistant (full month instead of abbrev)
    raw_df['x29'] = raw_df['x29'].replace(
        ["January", "Feb", "Mar", "Apr", "May", "Jun", "July",
         "Aug", "sept.", "Oct", "Nov", "Dev"],
        ["January", "February", "March", "April", "May", "June", "July",
         "August", "September", "October", "November", "December"])

    # x30 Adjust grammer
    raw_df['x30'] = raw_df['x30'].replace(
        ["monday", "tuesday", "wednesday", "thursday", "friday"],
        ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"])
```

```python
    # x32 Adjust value as categorical first (-0.0) and (0.0) as just 0 and
    # removing percentage sign then convert to numeric as its a better
    # representation
    raw_df['x32'] = raw_df['x32'].replace(
        ['-0.02%', '0.01%', '-0.0%', '-0.01%', '0.0%', '-0.03%', '0.02%',
         '0.03%', '-0.04%', '0.04%', '-0.05%', '0.05%'],
        ['-0.02', '0.01', '0.0', '-0.01', '0.0', '-0.03', '0.02',
         '0.03', '-0.04', '0.04', '-0.05', '0.05'])

    # x37 Remove $ char
    raw_df['x37'] = raw_df['x37'].str.strip('$')

    # Transform improper categorical vars into cts vars
    for ftr in bad_categ_ftrs:
        raw_df[ftr] = raw_df[ftr].astype("float")
    cts_ftrs.extend(bad_categ_ftrs)

    return raw_df, cts_ftrs, categ_ftrs
```

```python
def preprocess_data(train_df, test_df, cts_ftrs, categ_ftrs):
    # Impute Data
    cts_imputer = SimpleImputer(missing_values=np.nan, strategy="mean")
    train_df[cts_ftrs] = cts_imputer.fit_transform(train_df[cts_ftrs])

    categ_imputer = SimpleImputer(missing_values=np.nan,
                                  strategy="most_frequent")
    train_df[categ_ftrs] = categ_imputer.fit_transform(train_df[categ_ftrs])

    # Scale
    scaler = StandardScaler()
    train_df[cts_ftrs] = scaler.fit_transform(train_df[cts_ftrs])
    train_df.reset_index(drop=True, inplace=True)

    # One hot encode
    # train_df = pd.get_dummies(train_df)
    encoder = OneHotEncoder(sparse=False, handle_unknown='ignore')
    train_one_hot_categ = encoder.fit_transform(train_df[categ_ftrs])
    train_one_hot_categ = pd.DataFrame(train_one_hot_categ)
    train_df_final = pd.concat([train_one_hot_categ, train_df[cts_ftrs]],
                               axis=1)

    # Impute Data
    test_df[cts_ftrs] = cts_imputer.transform(test_df[cts_ftrs])

    test_df[categ_ftrs] = categ_imputer.transform(test_df[categ_ftrs])

    # Scale
    test_df[cts_ftrs] = scaler.transform(test_df[cts_ftrs])
    test_df.reset_index(drop=True, inplace=True)
```

```python
    # One hot encode
    # test_df = pd.get_dummies(test_df)
    test_one_hot_categ = encoder.transform(test_df[categ_ftrs])
    test_one_hot_categ = pd.DataFrame(test_one_hot_categ)
    test_df_final = pd.concat([test_one_hot_categ, test_df[cts_ftrs]], axis=1)

    return train_df_final, test_df_final


def nn_cross_val(raw_data, target_feature, early_stop=5, folds=2, log=None):
    oof_preds = []
    oof_true = []
    acc = 0
    y = raw_data[target_feature]
    x = raw_data.drop(target_feature, axis=1)
    x, cts_ftrs, categ_ftrs = reformat_data(x)
    first = True

    kf = KFold(n_splits=folds)
```

```python
    for id, (train_i, test_i) in enumerate(kf.split(x, y)):
        print("Fold:", id+1)
        x_train = x.loc[train_i, :]
        y_train = y.loc[train_i]
        x_test = x.loc[test_i, :]
        y_test = y.loc[test_i]
        x_train, x_test = preprocess_data(x_train,
                                          x_test,
                                          cts_ftrs,
                                          categ_ftrs)
        feature_count = x_train.shape[1]
        train_tup = (x_train, y_train)
        test_tup = (x_test, y_test)

        # Dataloader Creation
        batch_size = 32768
        train_set = CS7Dataset(train_tup)
        val_set = CS7Dataset(test_tup)
        train_loader = DataLoader(dataset=train_set,
                                  batch_size=batch_size,
                                  shuffle=True,
                                  num_workers=0)
        val_loader = DataLoader(dataset=val_set,
                                batch_size=256,  # Smaller for analysis
                                shuffle=True,
                                num_workers=0)

        # Model setup
        model = CS7NET(feature_count, 2048).to(DEVICE)
        weights = torch.FloatTensor([35.0/15.0]).to(DEVICE)
        criterion = nn.BCELoss(weight=weights)
        optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
        scheduler = MultiStepLR(optimizer, milestones=[30, 50], gamma=0.1)
        model_pckg = (model, criterion, optimizer, scheduler)
```

```python
        # Train Model
        train_losses, val_losses = train_model(
            model_package=model_pckg,
            train_loader=train_loader,
            val_loader=val_loader,
            num_epochs=100,
            early_stop_criterion=early_stop,
            record_modifier=0.0005,
        )

        model.eval()
        with torch.inference_mode():
            x_torch_test = torch.tensor(x_test.values, dtype=torch.float32)
            x_torch_test = x_torch_test.to(DEVICE)

            y_pred = model(x_torch_test).to("cpu")
            y_pred_np = y_pred.numpy()
            y_pred_np = np.where(y_pred_np >= 0.5, 1, 0).squeeze()
            y_test_np = y_test.squeeze()

            acc += np.sum(y_test_np == y_pred_np)/len(y_pred_np)
            oof_preds.extend(y_pred_np.tolist())
            oof_true.extend(y_test_np.tolist())
```

```python
                if first and (log is not None):
                    first = False

                    plt.plot(train_losses, label="Train Loss")
                    plt.plot(val_losses, label="Val Loss")
                    plt.title("Train Validation Loss Comparison")
                    plt.legend()
                    plt.savefig("loss_plot.png", bbox_inches='tight')
                    plt.clf()

                    classif_rep = classification_report(
                        y_test,
                        y_pred_np
                    )
                    log_section(log_file,
                                title="Model Classification Report",
                                content=classif_rep)

    acc /= folds
    return oof_preds, oof_true, acc
```

```python
if __name__ == '__main__':
    inputs = sys.argv
    check_valid_inputs(inputs)
    check_correct_file_structure(inputs)
    full_run_flag = 0
    if len(inputs) == 3:
        check_correct_flag_input_val(inputs)
        full_run_flag = get_flag(inputs)
    else:
        info_message = "This script by default runs on a small sample size"
        info_message += " by default to save"
        info_message += " computation time and memory.\n"
        info_message += " A full run can be computed by inputting 1 as"
        info_message += " the second script argument"
        print(info_message)

    report_type = ""
    if full_run_flag:
        print("Performing a full run...")
        report_type = "Full Report\n\n"
    else:
        print("Performing an example run...")
        report_type = "Example Report\n\n"

    # Log Start
    log_file = open("log.txt", "w")
    log_file.write("Case Study 7 Report\n\n")
    log_file.write(report_type)

    # Use input to load data
    raw_data = load_data(inputs)
```

```python
# Shuffle Data and sample
early_stop = 0
fold_count = 0
if full_run_flag:
    early_stop = 50
    fold_count = 10
    raw_data = raw_data.sample(frac=1)
else:
    early_stop = 10
    fold_count = 4
    raw_data = raw_data.sample(frac=0.01)
    raw_data.reset_index(drop=True, inplace=True)

# Explicitly Define Targ Feature
target_feature = ["y"]

# Cross Val
oof_pred, oof_true, cross_val_acc = nn_cross_val(raw_data,
                                                 target_feature,
                                                 early_stop=early_stop,
                                                 folds=fold_count,
                                                 log=log_file)

# Cross Val Accuracy
cross_val_acc_report = f"{fold_count} Fold Cross Validation Accuracy: "
cross_val_acc_report += f"{cross_val_acc}"
log_section(log_file,
            title="Model Accuracy Report",
            content=cross_val_acc_report)

```

```python
    # OOF Classification Report
    classif_rep = classification_report(
        oof_true,
        oof_pred
    )
    log_section(log_file,
                title="OOF Classification Report",
                content=classif_rep)

    # Report Score
    oof_pred = np.array(oof_pred)
    oof_true = np.array(oof_true)
    oof_combo = oof_pred - oof_true
    oof_combo[np.where(oof_combo == 1)] = 35
    oof_combo[np.where(oof_combo == -1)] = 15
    dollar_score = oof_combo.sum()
    dollar_score_report = f"Model OOF Dollar Score: ${dollar_score}"
    log_section(log_file,
                title="OOF Dollar Score",
                content=dollar_score_report)

    # Fin
    log_file.write("\n")
    log_file.close()
```