CAD for VLSI Design

Project Assignment 4

IR-drop Prediction

Instructor: Andy, Yu-Guang Chen Ph.D.

TA: 蔡書儀

Department/Class: Electrical Engineering 4A

Name: **陳緯亭**

Student ID Number: **109501201**

Contents

List Of Listings

1. The selection of training and validation sets.

The data was selected at random. With regard to the training set, 70 datasets were randomly selected. With regard to the validation set, 10 datasets were randomly selected. The objective is to perturb the datasets.

```
Training set is:  [29 28 59 79  8 53 17 30 68 69 27 18 54 19 44 36 25 15 58 43 66  4 50 22
 38 16 78 55 48 32 75 57 61 33 35  5 62 63 74 70 26  6 40 56  2 34 11 24
 13 52 76 64 65 14 41 23 73 45  1 71 72 46  0 10 77 12 31  9 51 20]
Validation set is:  [ 3  7 21 37 39 42 47 49 60 67]
```

Fig 1: The selection of training and validation sets

2. The feature selections and parameter settings

The feature was selected based on the model feature importances. The scores for MSE, RMSE, MAE, and CC were observed in order to determine which features were to be retained. The least important feature should be retained or deleted based on an assessment of its relative importance. The code can be observed in [Listing 1], and the result can be found in Figs [2] and [3].

```
        Feature  Importance
0             y    0.204408
1           SPR    0.164730
2          Reff    0.159653
3             x    0.116373
11  Ttransition    0.074426
8     Pinternal    0.064405
6             w    0.063943
7     Cell type    0.051081
10        Ipeak    0.034988
12        Cload    0.027908
5    TCinternal    0.021893
9         Pleak    0.015117
4      Tarrival    0.001075
Train MSE: 7.370561174035611e-05
Train RMSE: 0.00858519724527958
Train MAE: 0.00652119083669505
Train CC: 0.9355736736697305
Test MSE: 8.040160665132568e-05
Test RMSE: 0.008966694298978062
Test MAE: 0.0068158693019462835
Test CC: 0.9307824479114073
```

Fig 2: Features sort by importance

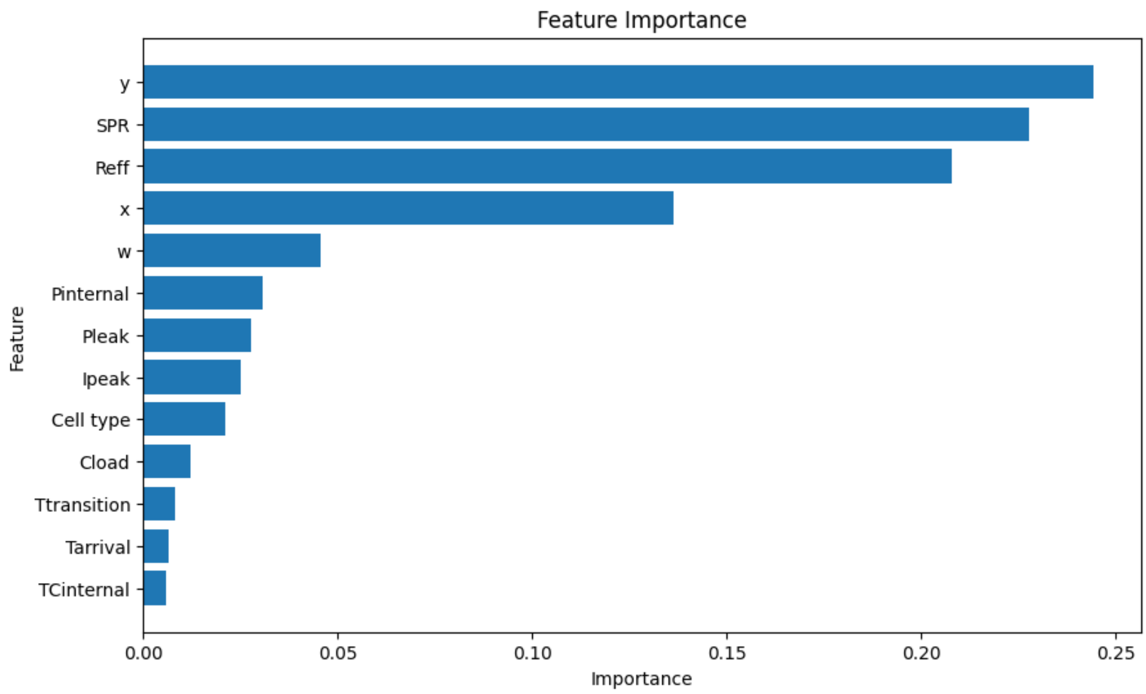

Fig 3: Features sort by importance

Listing 1: Features sort by importance

```python
import xgboost as xgb
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import random

# Define file path and filename
DataSet_Path = "/home/CAD112/PA4/Training/"
for i in range(1, 21):
    index = random.sample(range(1, 79), 1)[0]
    files = [f"MEMC_{index}.csv"]
    print(f"MEMC_{index}.csv loaded...")

# Extract features and labels
selected_features = ['y', 'SPR', 'Reff', 'x', 'Tarrival', 'TCinternal', 'w', 'Cell type', 'Pinternal', 'Pleak', 'Ipeak', 'Ttransition', 'Cload']
all_data = []

# Read and combine data
for file in files:
    data = pd.read_csv(os.path.join(DataSet_Path, file))
    all_data.append(data)

```

```python
    # Combine all data
    combined_data = pd.concat(all_data)

    # Extract features and labels
    X = combined_data[selected_features]
    y = combined_data["IR-drop"]

    # Split into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
        random_state=9527)

    # Train the model
    model = xgb.XGBRegressor(objective='reg:squarederror')
    model.fit(X_train, y_train)

    # Get feature importance
    importance = model.feature_importances_
    importance_df = pd.DataFrame({
        'Feature': selected_features,
        'Importance': importance
    })

    # Sort by importance
    importance_df = importance_df.sort_values(by='Importance', ascending=False)
    print(importance_df)

    # Predict and calculate scores
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    def score(y_true, y_pred):
        mse = np.mean((y_true - y_pred) ** 2)
        rmse = np.sqrt(mse)
        mae = np.mean(np.abs(y_true - y_pred))
        cc = np.corrcoef(y_true, y_pred)[0, 1]
        return mse, rmse, mae, cc

    train_mse, train_rmse, train_mae, train_cc = score(y_train, y_train_pred)
    test_mse, test_rmse, test_mae, test_cc = score(y_test, y_test_pred)

    print('Train MSE:', train_mse)
    print('Train RMSE:', train_rmse)
    print('Train MAE:', train_mae)
    print('Train CC:', train_cc)

    print('Test MSE:', test_mse)
    print('Test RMSE:', test_rmse)
    print('Test MAE:', test_mae)
    print('Test CC:', test_cc)
```

```
    # Visualize feature importance
    plt.figure(figsize=(10, 6))
    plt.barh(importance_df['Feature'], importance_df['Importance'])
    plt.xlabel('Importance')
    plt.ylabel('Feature')
    plt.title('Feature Importance')
    plt.gca().invert_yaxis()

    # Save the image
    plt.savefig('feature_importance.png', bbox_inches='tight')
    plt.show()
```

In order to ascertain the optimal configuration, I employed the methodology outlined in [Listing 1]. Through a process of trial and error, I identified the following features as the most suitable for the given context. The outcome of this process is illustrated in Fig [4]. [Listing 2]

```
['y', 'SPR', 'Reff', 'x', 'Tarrival', 'TCinternal', 'w', 'Cell type', 'Pinternal', 'Pleak', 'Ipeak', 'Ttransition',
 'Cload']
```

Fig 4: The selection of training and validation sets

Listing 2: Get features

```
# Get all input feature
def get_feature():
    # ****************************************************************
    # TODO 2: Select the features for training
    # Sample: feature_name = ["x", "y", "w", "h"]
    # Note that you should not put "IR-drop" as your training features

    feature_name = ['y','SPR','Reff','x','Tarrival','TCinternal','w','Cell type','
       Pinternal','Pleak','Ipeak','Ttransition','Cload']
    print(feature_name)
    np.save("./feature_name.npy",np.array(feature_name))
    # END of TODO 2
    # ****************************************************************
```

In order to define the model parameters, a series of trials were conducted, with the results recorded at each stage. This process was repeated until the desired outcome was achieved. Furthermore, it was observed that certain parameters had no impact on the results. For instance, the parameters ”subsample,” ”colsample_bytree,” ”colsample_bylevel,” ”colsample_bynode,” ”gamma,” and so forth. Consequently, these parameters were set to their default values. [Listing 3]

Listing 3: Define model parameters

```python
# ********************************************************************
# TODO 3: Define model parameters
# You can add more parameter settings if needed

param = {
    'max_depth': 8,  # Maximum depth of the tree
    'eta': 0.4,  # Learning rate
    'eval_metric': "rmse",  # Evaluation metric
    'objective': "reg:squarederror",  # Objective function
    'lambda': 1,  # L2 regularization
    'n_estimators': 500,  # Number of trees
    # 'subsample': 0.8,  # Subsample ratio
    # 'colsample_bytree': 0.8,  # Feature ratio per tree
    # 'colsample_bylevel': 0.8,  # Feature ratio per level
    # 'colsample_bynode': 0.8,  # Feature ratio per node
    # 'gamma': 0.1,  # Minimum loss reduction
    # 'min_child_weight': 1,  # Minimum sum of instance weight
    # 'alpha': 0.1  # L1 regularization
}
# END of TODO 3
# ********************************************************************
```

Following a comprehensive investigation, it was established that the optimal training iteration was 30. Consequently, the value of num_round was set to 50 in order to ensure a greater degree of possibility. [Listing 4]

Listing 4: Determine training itreation

```python
# TODO 4: Determine training itreation
num_round = 50
```

Furthermore, the early stopping rounds were adjusted in order to prevent overfitting during the training process. Fig [5]

```python
#  you can adjust this part if you want
model = xgb.train(params = param,
                  dtrain = dtrain,
                  num_boost_round = num_round,
                  evals = watchlist,
                  early_stopping_rounds = 5
                  )
```

Fig 5: Early stopping rounds

3. The evaluation results

```
MAE(mV):   21.69 mV
MaxE(mV):   152.15 mV
CC:   0.66
NRMSE(%):   3.71 %
```

Fig 6: The evaluation result

4. The hardness of this assignment / I overcome it

1. What are the distinctions between objective and evaluative metrics?
   The objective is to optimise the model. Evaluative metrics are employed to assess the model's effectiveness. Consequently, evaluative metrics exert no influence on the training of the model. xgboost objective 和 eval_metric 的區別

2. What parameters can be modified?
   XGB 系列-XGB 参数指南

3. Whether the tree_method can utilise gpu_hist.
   The output would be "WARNING: /src/learner.cc:248: No visible GPU is found, setting ‘gpu_id‘ to -1".

5. Bonus

The results demonstrated that the outcome was inferior to that of XGBoost. It appears that the parameters and features must be modified according to the specific technology in question. Furthermore, the LightGBM algorithm generates the following warning message: "No further splits with positive gain were observed, with the best gain being -inf for every training." This is the primary reason for the unsatisfactory results. If you're looking for more information, you can find it in [Listing 5], [Listing 6], [Listing 7], and [Listing 8].

```
all_cells_golden shape: (223829, 10)
MAE(mV):  153.77 mV
MaxE(mV):  392.38 mV
CC:  0.11
NRMSE(%):  20.39 %
```

Fig 7: The evaluation result of LightGBM

Listing 5: LightGBM_Training

```python
#!/usr/bin/env python
# coding: utf-8
# Lab for IR drop prediction using LightGBM

# Import packages
from sklearn.datasets import dump_svmlight_file
import lightgbm as lgb
import pandas as pd
import numpy as np
import os, shutil
import subprocess
import pickle
import time

# Circuit name
DESIGN = "MEMC"

# Debug setting
DEBUG = True

# Create directory to store model and data
p1 = subprocess.Popen(["mkdir -p ./model"], stdout=subprocess.PIPE, shell=True)
p1 = subprocess.Popen(["mkdir -p ./data/train/"], stdout=subprocess.PIPE, shell=True)
p1 = subprocess.Popen(["mkdir -p ./data/val"], stdout=subprocess.PIPE, shell=True)

# Wait until directory is created
p1.wait()

# ******************************************************************
# TODO 1: Set training and validation dataset
TRAINING_SET = np.random.choice(78, 20, replace=False)
# np.arange(10)
VALIDATION_SET = np.array([80])
print("Training set is: ", TRAINING_SET)
print("Validation set is: ", VALIDATION_SET)
# END of TODO 1
# ******************************************************************

# Get all input features
```

```
def get_feature():
    # ******************************************************************
    # TODO 2: Select the features for training
    feature_name = ['y', 'SPR', 'Reff', 'x', 'w', 'Pleak', 'h', 'Ipeak']
    print(feature_name)
    np.save("./feature_name.npy", np.array(feature_name))
    # END of TODO 2
    # ******************************************************************

# Save training data to temp files
def dump_file(raw_data, pattern_num, dir_name):
    feature_name = np.load("./feature_name.npy",allow_pickle=True)
    X = raw_data.loc[:, raw_data.columns.isin(feature_name)]

    # Golden IR-drop
    Y = raw_data["IR-drop"]
    dump_svmlight_file(X,Y,"./data/"+dir_name+"/"+DESIGN+"_"+str(pattern_num)+".dat")

# Split training & validation set
def load_data(training_set, validation_set):
    DataSet_Path = "/home/CAD112/PA4/Training/"
    print('Loading dataset...')
    train_name_dict = [str(i+1) for i in training_set]
    val_name_dict = [str(i) for i in validation_set]
    feature_name = np.load("./feature_name.npy", allow_pickle=True)

    train_X = []
    train_y = []
    val_X = []
    val_y = []

    for pattern_num in train_name_dict:
        FILE_STR = DESIGN + '_' + str(pattern_num)
        all_data = pd.read_csv(DataSet_Path + FILE_STR + '.csv')
        X_train = all_data[feature_name]
        y_train = all_data["IR-drop"]
        train_X.append(X_train)
        train_y.append(y_train)
        print(FILE_STR + '.csv loaded for training...')

    for pattern_num in val_name_dict:
        FILE_STR = DESIGN + '_' + str(pattern_num)
        all_data = pd.read_csv(DataSet_Path + FILE_STR + '.csv')
        X_val = all_data[feature_name]
        y_val = all_data["IR-drop"]
        val_X.append(X_val)
        val_y.append(y_val)
        print(FILE_STR + '.csv loaded for validation...')

    # Combine all training and validation data
```

```python
    X_train = pd.concat(train_X)
    y_train = pd.concat(train_y)
    X_val = pd.concat(val_X)
    y_val = pd.concat(val_y)

    dtrain = lgb.Dataset(X_train, label=y_train)
    dval = lgb.Dataset(X_val, label=y_val)

    return dtrain, dval


def training(dtrain, dval):
    # Define model parameter
    param = {
        'max_depth': 8,
        'learning_rate': 0.1,  # 通常學習率要較低
        'metric': "rmse",
        'objective': "regression",
        'lambda_l1': 1,
        'lambda_l2': 1,
        'tree_learner': 'serial',
        'num_leaves': 2**8,  # 這裡設置為 2^max_depth
        'min_data_in_leaf': 10,
        'feature_fraction': 0.8,
        'bagging_fraction': 0.8,
        'bagging_freq': 5,
        'verbose': 0,
        'min_gain_to_split': 0.01,  # 設置一個較小的 min_gain_to_split
        'early_stopping_round': 10,
    }

    # Start training model
    model = lgb.train(params=param,
                      train_set=dtrain,
                      valid_sets=dval)

    # Save the model
    model.save_model("./model/PA4_Model.cbm")

    return model


if __name__ == '__main__':
    # Get input feature
    get_feature()

    # Data preprocessing
    dtrain, dval = load_data(TRAINING_SET, VALIDATION_SET)

    # Train the model
```

```
    print("start training")
    start = time.time()
    model = training(dtrain, dval)
    end = time.time()

    print("total time: ", end - start)
    print("model produced successfully")

    # Clear all temp files
    shutil.rmtree("./data/train/")
    shutil.rmtree("./data/val/")
```

To evaluate the LightGBM, I have to modify the following three lines.

Listing 6: LightGBM_Evaluation change 1

```
dpredict = lgb.Dataset(X, feature_name=feature_name)
```

Listing 7: LightGBM_Evaluation change 2

```
# Require trained model to use below options
TRAINED_MODEL = "./model/PA4_Model.cbm"
```

Listing 8: LightGBM_Evaluation change 3

```
# Load the model
model = lgb.Booster(model_file=TRAINED_MODEL)
```

6. Suggestions

No.