CS376 Term Project

20170130 Kim Aujin 20160288 Bae Hanseong
20160487 Lee Yongwook 20170491 Lee Jaejun

December 14, 2018

1 Model Descriptions

Our model is a model that combined binary encoding and Gradient Boosting Regression tree.We first filled blank data with median of that data. After, we used binary encoding to deal with categorical data, simple transformation of data to deal with date/degree, special treatment for date data, and then used Gradient Boosting regression tree to get the final result.

To handle categorical data, we used binary encoding. We also changed date data to six digit integer to handle date data, degree to sin/cos values. Since we may need to predict future prices, we took out date data from original data set, and assumed that the price is linearly proportional to dates. Last, we took logarithm for all prices so that we can decrease the loss created by gradient boosting.

Table 1: data transforming equations

categorical data	**degree data**	**date data**
X to Y (Y:binary representation of X)	θ to (cos θ, sin θ)	YYYY-MM-DD to YYYY+MM/12+DD/365

After we finished processing data, we used Gradient Boosting regression tree as main model. We trained the tree with processed data, and used modified 5-folding using modular operation to verify whether our model fits well, and to tune the hyper-parameters.

2 Unique Methods

2.1 Gradient Boosting Regression Tree with logarithm

Our unique method is the Gradient Boosting Regression Tree with logarithm. We took logarithm for all prices before training, so that we can decrease the loss created in Gradient Boosting Algorithm. Taking logarithm may be a common method, and so does gradient boosting. However, combining these two to decrease the loss is our own idea, which in so we think this is our unique method.

3 Libraries

3.1 Numpy

Version: 1.15.1
Purpose: numerical computation on datas.

3.2 scikit-learn

Version: 0.20.1
Purpose:To import Gradient Boosting Regressor, Imputer, etc.

3.3 pandas

Version: 0.23.4
Purpose:To deal with csv files.

4 Source codes

```
import numpy as np
import pandas as pd
import gc
import warnings
import sys
import time
import os
import math
from math import isnan
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.preprocessing import Imputer
from sklearn.preprocessing import MinMaxScaler
FILE_NAME =1 ## input file 이름
MAKEY_SKIP = 2 ## makey procedure가 이미 실행 된 경우
    skip할 지 결정
KIND_OF_ENC = 3 ## 어떤 encoder를 사용할지 결정(only non
    used)
ENCODER_SKIP = 4 ## encoder가 이미 실행 된 경우 skip할 지
    결정
IMPUTE_SKIP = 5 ## impute이 이미 실행 된 경우 skip할 지
    결정
```

```
M_FEATURES = 20
CONST_DATE_INDEX = 13
TREE_DEPTH = 10
def log_y(y): ##y 행렬 원소들 각각 log
    logy = np.zeros((length(y),1))
    for i in range(length(y)):
        logy[i,0] = math.log(y[i])*10000
    return pd.DataFrame(logy)

def exp_y(y): ##y 행렬 원소들 각각 exp
    expy = np.zeros((length(y),1))
    for i in range(length(y)):
        expy[i,0] = math.exp(y[i]/10000)
    return expy

def sindeg(x): ## deg로 값을 input받는 sin 함수.
    return math.sin(x*math.pi/180)
def cosdeg(x): ## deg로 값을 input받는 cos 함수.
    return math.cos(x*math.pi/180)

def handle_deg(application_train):
    ## given data의 각도 값을 sin과 cos으로 바꾸어 각도가
        연속적으로 변화하게 만든다.
    deg = application_train[["9"]]
    deg1 = deg.values
    sincos = np.zeros((length(deg1),2))
    for i in range(length(deg1)):
        sincos[i,0] = sindeg(deg1[i])
        sincos[i,1] = cosdeg(deg1[i])
    array1 = application_train.values
    temp1 =  np.concatenate((array1[:,:9],sincos),axis=1)
    result = np.concatenate((temp1,array1[:,10:]),axis=1)
    return pd.DataFrame(result)

def cutConcat(array1,col_index): ## given ndarray and int
    , returns ndarray
    return np.concatenate((array1[:,:col_index],array1[:,
        col_index+1:]),axis=1)
def length(array1): ## given ndarray, returns the length
    of array
    return array1.shape[0]
def width(array1): ## given ndarray, returns the width of
    array
    return array1.shape[1]
def filename(): ## 경로 string "FILE_NAME\KIND_OF_ENC\"를
    만든다.
    return sys.argv[FILE_NAME][:-4]+'/'+sys.argv[
        KIND_OF_ENC]
def dateColumn2Float(dataframe, col_index):
    ## 날짜 string은 일반적인 계산이 불가능하므로, float
        로 변환하여(1일 = 1/365년)
    ## float처럼 계산할 수 있게 만든다.
    array1 = dataframe.values
    for i in range(length(array1)):
        if (type(array1[i, col_index]) is float):
            if (isnan(array1[i, col_index])):
                pass
        else:
            date = pd.to_datetime(array1[i,col_index].
                replace("-",""), format='%Y%m%d')
            new_year_day = pd.Timestamp(year=date.year,
                month=1, day=1)
            day_of_year = (date - new_year_day).days + 1
            array1[i, col_index] = day_of_year/365 + date
                .year
    return pd.DataFrame(array1)
def makey(application_train):
    ## 결과값을 데이터에서 미리 분리해 둬서 return하는 함
        수이다.
    print("Making␣y...")
    if(sys.argv[MAKEY_SKIP] == "Y"):
        y = pd.read_csv(filename()+'/y.csv')
        print("y=")
        print(y)
        gc.collect()
        return y
    else:
        y = application_train[["price"]]
        application_train.drop("price", axis=1, inplace=
            True)
        print("y=")
        print(y)
        y.to_csv(filename()+'/y.csv', index=False)
        gc.collect()
        return y

def enc_selector(application_train):
    ## categorical data를 handle하기 위해서, 어떤 encoder
        를 사용할 지 선택하는 함수였지만,
    ## 이를 모델에 적용시켰을 때, binary encoder를 사용했
        을 때의 train error가 가장 작아서
    ## binary encoder만을 사용하기로 했다.
    if(sys.argv[KIND_OF_ENC] == "bin"):
        print("Binary␣encoder␣selected.")
        return binaryNd2f(application_train)
    if(sys.argv[KIND_OF_ENC] == "non"):
        print("Not␣encoding␣selected")
        return notEncode(application_train)

def binaryNd2f(application_train):
    ## categorical data의 각각의 value에 그 column의 최소
        값을 빼서
    ## 이를 이진법으로 나타내고(당연히 categorical data는
        모두 integer value를 가진다)
    ## 각 자리와 그 자리수를 feature 한 줄과 그 value로
        바꾸는 encoding이다.
    ## ex) 이 column의 최소값이 3, 최대값이 8, i번째 row
        의 값이 6이면
    ##      6 - 3 = 011으로 바꾸어, 0, 1, 1로 바꾼다.
    if(sys.argv[ENCODER_SKIP] == "Y"):
        print("Skipping␣binary␣encoder")
        gc.collect()
        total_data_df = pd.read_csv(filename()+'/concat_b
            .csv')
        return total_data_df
    print("Executing␣binary␣encoder..")
    ## handle degree
    application_train = handle_deg(application_train)
    gc.collect()
    cat_features = [4, 5, 6, 7, 14, 18]
    non_cat_features = []
    for i in range(application_train.shape[1]):
        if i in cat_features:
            pass
        else:
            non_cat_features.append(i)
    cat_max = []
    cat_min = []
    dims = []
    for i in cat_features:
        temp_max = -123456
        temp_min = 123456
        for j in range(length(application_train)):
            if(np.isnan(application_train.iloc[j, i])):
                pass
            else:
                if(temp_max < application_train.iloc[j, i
                    ]):
```

```
                    temp_max = application_train.iloc[j,
                        i]
                if(temp_min > application_train.iloc[j, i
                    ]):
                    temp_min = application_train.iloc[j,
                        i]
        cat_max.append(temp_max)
        cat_min.append(temp_min)
        dims.append(int(math.ceil(np.log2(temp_max−
            temp_min+1))))
    binenc_info = np.concatenate((np.array(cat_min), np.
        array(dims)))
    pd.DataFrame(binenc_info).to_csv(filename()+'/
        binenc_info.csv', index=False)
    non_cat_train = application_train.iloc[:,
        non_cat_features].values
    for i in range(6):
        binencoded = np.zeros((length(application_train),
            dims[i]))
        for j in range(length(application_train)):
            temp = application_train.iloc[j, cat_features
                [i]]−cat_min[i]
            for k in range(int(math.ceil(dims[i]))):
                binencoded[j, k] = temp%2
                if(np.isnan(temp)):
                    pass
                else:
                    temp = math.floor(temp/2)
        non_cat_train = np.concatenate((non_cat_train,
            binencoded), axis=1)
    bin_concated=pd.DataFrame(non_cat_train)
    bin_concated = dateColumn2Float(bin_concated, 0)
    bin_concated = dateColumn2Float(bin_concated,
        CONST_DATE_INDEX)
    print("done.")
    bin_concated.to_csv(filename()+'/concat_b.csv', index
        =False)
    print(bin_concated)
    return bin_concated

def notEncode(application_train):
    ## encoding을 하지 않고, given data의 날짜를 float로
        고치기만 하는 함수.
    if(sys.argv[ENCODER_SKIP] == "Y"):
        print("Skipping␣binary␣encoder")
        gc.collect()
        total_data_df = pd.read_csv(filename()+'/concat_n
            .csv')
        return total_data_df
    print("Not␣encoder..")
    ## handle degree
    application_train = handle_deg(application_train)
    gc.collect()
    application_train=dateColumn2Float(application_train,
         0) ##0번째 feature은 날짜.
    application_train = dateColumn2Float(
        application_train, CONST_DATE_INDEX + 6)
    print("done")
    application_train = pd.DataFrame(application_train)
    ## 완료된 결과를 filename()+'/concat_n.csv'에 저장한
        다.
    application_train.to_csv(filename()+'/concat_n.csv',
        index=False)
    return application_train

def impute(total_data_df):
    ## sklearn의 Imputer함수를 이용해서, 빈 칸을 채우는
        함수.
    ## 중간값을 빈 칸에 채운다.
    print("Imputing...")
    if(sys.argv[IMPUTE_SKIP] == "Y"):
        gc.collect()
        total_data_df = pd.read_csv(filename()+'/impute.
            csv')
        return total_data_df
    else:
        gc.collect()
        total_data_df = Imputer(strategy='median').
            fit_transform(total_data_df)
        print("done.")
        print(total_data_df)
        print("Saving␣data␣on␣impute.csv")
        pd.DataFrame(total_data_df).to_csv(filename()+'/
            impute.csv', index=False)
        return total_data_df

def gbr(total_data_df, y, testx, testy):
    ## make a model using gradient boosting regressor
        tree
    print("gradientboostingregressor")
    ## maximum depth of tree is 10
    GBR = GradientBoostingRegressor(n_estimators = 100,
        max_depth = TREE_DEPTH)
    GBR.fit(total_data_df,log_y(y)) ## y의 log를 취한다.
    acc=accuracy(testy,exp_y(GBR.predict(testx))) ## 다시
        exp로 원래 값으로 복원한다.
    print("Accuracy␣—>␣", acc)
    return acc

def accuracy(real, pred):
    ## calculate accuracy between real data and predicted
        data
    ## accuracy is calculated as same as homework slide
    total = 0
    for i in range(len(real)):
        total=total+abs((real[i]-pred[i])/real[i])
    return 1-1/len(real)*total
def fold5(total_data_df, y):
    ## beautiful code implementing 5-fold cross
       vallidation
    X = total_data_df.values
    ynd = y.values
    dim_mody = width(X)
    list0 =[]
    list1 =[]
    list2 =[]
    list3 =[]
    list4 =[]
    for i in range(len(X)):
        if(i%5==0):
            list0.append(i)
        if(i%5==1):
            list1.append(i)
        if(i%5==2):
            list2.append(i)
        if(i%5==3):
            list3.append(i)
        if(i%5==4):
            list4.append(i)
    listn4 =sorted(list0 + list1 + list2 + list3)
    listn3 =sorted(list0 + list1 + list2 + list4)
    listn2 =sorted(list0 + list1 + list3 + list4)
    listn1 =sorted(list0 + list2 + list4 + list3)
    listn0 =sorted(list1 + list2 + list3 + list4)
    print("yndn0␣=␣", ynd[listn0,:])
    print("yndn1␣=␣", ynd[listn1,:])
    print("yndn2␣=␣", ynd[listn2,:])
    print("yndn3␣=␣", ynd[listn3,:])
    print("yndn4␣=␣", ynd[listn4,:])
    acc0 = gbr(X[listn0,:], ynd[listn0,:],X[list0,:],ynd[
        list0,:])
    acc1 = gbr(X[listn1,:], ynd[listn1,:],X[list1,:],ynd[
        list1,:])
    acc2 = gbr(X[listn2,:], ynd[listn2,:],X[list2,:],ynd[
        list2,:])
    acc3 = gbr(X[listn3,:], ynd[listn3,:],X[list3,:],ynd[
        list3,:])
    acc4 = gbr(X[listn4,:], ynd[listn4,:],X[list4,:],ynd[
        list4,:])
    print((acc0+acc1+acc2+acc3+acc4)/5)
def predict(train_x, y):
    ## 전체 data_train.csv의 값으로
        GradientBoostingRegressor model을 만들고
    ## data_test.csv에 이를 적용한다.
    test_x = pd.read_csv("data_test.csv")
    GBR = GradientBoostingRegressor(n_estimators = 100,
        max_depth = TREE_DEPTH)
    GBR.fit(train_x,log_y(y.values)) ## y의 log를 취한다.
    test_data = exp_y(GBR.predict(impute(enc_selector(
        test_x)))) ## exp로 원래 값으로 복원한다.
    pd.DataFrame(test_data).to_csv(filename()+'/return.
        csv', index=False)
start = time.time()
if(not os.path.exists(filename())):
    os.makedirs(filename())
gc.collect()
warnings.filterwarnings("ignore")

application_train = pd.read_csv(sys.argv[FILE_NAME])
y=makey(application_train)

total_data_df = enc_selector(application_train)
gc.collect()

total_data_df = pd.DataFrame(impute(total_data_df))
gc.collect()
fold5(total_data_df, y)
predict(total_data_df, y)
end = time.time()
run_time = end - start
print(run_time)
```

5 Performance

It uses the 5-fold cross validation for the training. Using modular tool, the original train data is partitioned into 5 equal-sized subsamples. The reason for using modularity is to make subsample's distribution equally.

Table 2: Validation result using 5-fold cross validation without unique method

	fold 1	**fold 2**	**fold 3**	**fold 4**	**fold 5**	**mean**
Accuracy	0.948515	0.958869	0.948604	0.948661	0.948705	0.948671

Execution time is 3757sec.

Table 3: validation result using 5-fold cross validation with unique method

	fold 1	**fold 2**	**fold 3**	**fold 4**	**fold 5**	**mean**
Accuracy	0.949924	0.950214	0.949900	0.949579	0.950801	0.950084

Execution time is 4377 sec.

Performance was improved by 0.14% when using the unique method. Because base model have already large accuracy, unique method makes quite impactive changes. This program takes very long time, but we can reduce excution time to change hyperparameter. We can find optimal performance between execution time and accuracy through changing hyperparameter.