# Part 2 Regression

* Programmatically downloads Q12005 and Q22005 origination data and pre-processes it.

import requests

import urllib

import os

from bs4 import BeautifulSoup

import http.cookiejar

import shutil

import pandas as pd

url = 'https://freddiemac.embs.com/FLoan/Data/download.php'

cookie={'PHPSESSID':'11vr6kgl2a9lt7b8if0mv08vq0'}

r = requests.post(url,cookies=cookie)

content=r.content

soup = BeautifulSoup(content,'lxml')

all\_href = soup.find\_all('a')

i=0;

dww=['Q12005','Q22005']

lod=len(dww)

url\_list=['https://freddiemac.embs.com/FLoan/Data/']\*lod

for href in all\_href:

for s in dww:

if s in href['href']:

url\_list[i]=url\_list[i]+href['href']

i+=1

if (not os.path.exists('datapart2')):

os.mkdir('datapart2')

os.chdir('datapart2')

for i in range(len(url\_list)):

r = requests.get(url\_list[i],cookies=cookie)

with open(url\_list[i][71:77]+'.zip','wb') as code:

code.write(r.content)

files= os.listdir()

for file in files:

for docu in dww:

if ('.zip' in file) and (docu in file):

shutil.unpack\_archive(file)

os.chdir('..')

Use cookies to get access to the download page, use BeautifulSoup to get content of the webpage, find the link we need, download the zip file, then unzip it by shutil package.

def preprocessing(data):

data['POSTAL CODE'].fillna(85200.0,inplace=True)

data['PREPAYMENT PENALTY MORTGAGE FLAG'].fillna('N',inplace=True)

data['METROPOLITAN STATISTICAL AREA'].fillna(16974.0,inplace=True)

data.drop(['SUPER CONFORMING FLAG'],axis=1,inplace=True)

data['CREDIT SCORE'].replace(to\_replace=9999, value=np.nan, inplace=True)

data['CREDIT SCORE'].fillna((data['CREDIT SCORE'].mean()), inplace=True)

data['FIRST TIME HOMEBUYER FLAG'].replace(to\_replace='9', value=np.nan, inplace=True)

data['FIRST TIME HOMEBUYER FLAG'].fillna('N', inplace=True)

data['Maturity\_year'] = [int(str(d)[:4]) for d in data['MATURITY DATE']]

del data['MATURITY DATE']

data['MORTGAGE INSURANCE PERCENTAGE'].replace(to\_replace=999,value=0,inplace=True)

data['NUMBER OF UNITS'].replace(to\_replace=99,value=1,inplace=True)

data['ORIGINAL COMBINED LOAN-TO-VALU'].replace(to\_replace=999,value=80,inplace=True)

data['ORIGINAL DEBT-TO-INCOME (DTI) RATIO'].replace(to\_replace=999,value=float('nan'),inplace=True)

data['ORIGINAL DEBT-TO-INCOME (DTI) RATIO'].fillna(data['ORIGINAL DEBT-TO-INCOME (DTI) RATIO'].mean(),inplace=True)

data['ORIGINAL LOAN-TO-VALUE'].replace(to\_replace=999,value=float('nan'),inplace=True)

data['ORIGINAL LOAN-TO-VALUE'].fillna(data['ORIGINAL LOAN-TO-VALUE'].mean(),inplace=True)

data.drop('PRODUCT TYPE',axis=1,inplace=True)

data.drop('LOAN SEQUENCE NUMBER',axis=1,inplace=True)

data['PROPERTY TYPE'].replace(to\_replace=99,value='SF',inplace=True)

data['LOAN PURPOSE'].replace(to\_replace=9,value='C',inplace=True)

data['NUMBER OF BORROWERS'].replace(to\_replace=99,value=2,inplace=True)

del data['FIRST PAYMENT DATE']

data.drop('ORIGINAL LOAN-TO-VALUE',axis=1,inplace=True)

data.drop('Maturity\_year',axis=1,inplace=True)

#dictionary for each column

cleanup\_nums = {"FIRST TIME HOMEBUYER FLAG":{"Y": 1, "N": 0},

"OCCUPANCY STATUS": {"P": 1, "S": 2, "I": 3 },

"CHANNE":{"T": 1, "R":2,"C":3,"B":4},

"PREPAYMENT PENALTY MORTGAGE FLAG":{"Y":1,"N":0},

"PROPERTY TYPE":{"SF":1,"PU":2,"CO":3,"MH":4,"CP":5,"99":99},

"LOAN PURPOSE":{"C":1,"P":2,"N":3},

"SELLER NAME":{"Other sellers": 0, "COUNTRYWIDE": 1, "TAYLOR,BEAN&WHITAKER": 2, "PROVIDENTFUNDINGASSO": 3, "USBANKNA": 4, "FIFTHTHIRDBANK": 5, "ABNAMROMTGEGROUP,INC": 6, "CHASEHOMEFINANCELLC": 7, "NATLCITYMTGECO": 8, "WELLSFARGOBANK,NA": 9, "GMACMTGECORP": 10, "WASHINGTONMUTUALBANK": 11, "FLAGSTARBANK,FSB": 12, "BANKOFAMERICA,NA": 13},

"PREPAYMENT PENALTY MORTGAGE FLAG":{"Y": 1, "N": 0},

"PROPERTY STATE":{"RI": 0, "OK": 1, "NY": 2, "MO": 3, "MN": 4, "IL": 5, "KY": 6, "WA": 7, "TX": 8, "FL": 9, "CA": 10, "IN": 11, "NJ": 12, "ID": 13, "TN": 14, "KS": 15, "MI": 16, "IA": 17, "MT": 18, "GA": 19, "OH": 20, "OR": 21, "ME": 22, "CT": 23, "WV": 24, "NH": 25, "VA": 26, "NC": 27, "AZ": 28, "NE": 29, "MD": 30, "MA": 31, "UT": 32, "CO": 33, "ND": 34, "PA": 35, "SC": 36, "DE": 37, "SD": 38, "WI": 39, "AL": 40, "AK": 41, "VT": 42, "LA": 43, "AR": 44, "NM": 45, "HI": 46, "DC": 47, "MS": 48, "NV": 49, "GU": 50, "WY": 51, "PR": 52, "VI": 53},

"SERVICER NAME":{"USBANKNA": 0, "Other servicers": 1, "PNCMTGESERVICES,INC": 2, "WELLSFARGOBANK,NA": 3, "WASHINGTONMUTUALBANK": 4, "PNCBANK,NATL": 5, "NATLCITYMTGECO": 6, "JPMORGANCHASEBANK,NA": 7, "NATIONSTARMTGELLCDBA": 8, "COUNTRYWIDE": 9, "BANKOFAMERICA,NA": 10, "BACHOMELOANSERVICING": 11, "CITIMORTGAGE,INC": 12, "PROVIDENTFUNDINGASSO": 13, "ABNAMROMTGEGROUP,INC": 14, "FIFTHTHIRDBANK": 15, "GMACMORTGAGE,LLC": 16},

"OCCUPANCY STATUS":{"P": 1, "S": 2, "I": 3 },

"CHANNE":{"T": 1, "R":2,"C":3,"B":4}

}

data.replace(cleanup\_nums, inplace=True)

#d\_seller = {"Other sellers": 0, "COUNTRYWIDE": 1, "TAYLOR,BEAN&WHITAKER": 2, "PROVIDENTFUNDINGASSO": 3, "USBANKNA": 4, "FIFTHTHIRDBANK": 5, "ABNAMROMTGEGROUP,INC": 6, "CHASEHOMEFINANCELLC": 7, "NATLCITYMTGECO": 8, "WELLSFARGOBANK,NA": 9, "GMACMTGECORP": 10, "WASHINGTONMUTUALBANK": 11, "FLAGSTARBANK,FSB": 12, "BANKOFAMERICA,NA": 13}

#d\_service = {"USBANKNA": 0, "Other servicers": 1, "PNCMTGESERVICES,INC": 2, "WELLSFARGOBANK,NA": 3, "WASHINGTONMUTUALBANK": 4, "PNCBANK,NATL": 5, "NATLCITYMTGECO": 6, "JPMORGANCHASEBANK,NA": 7, "NATIONSTARMTGELLCDBA": 8, "COUNTRYWIDE": 9, "BANKOFAMERICA,NA": 10, "BACHOMELOANSERVICING": 11, "CITIMORTGAGE,INC": 12, "PROVIDENTFUNDINGASSO": 13, "ABNAMROMTGEGROUP,INC": 14, "FIFTHTHIRDBANK": 15, "GMACMORTGAGE,LLC": 16}

#d\_seller.setdefault(data["SELLER NAME"], default=0)

#d\_service.setdefault(data["SERVICER NAME"], default=0)

for i in range(len(data['SELLER NAME'])):

if (not(str(data['SELLER NAME'][i]).isdigit())):

data['SELLER NAME'][i]=0

for i in range(len(data['SERVICER NAME'])):

if (not(str(data['SERVICER NAME'][i]).isdigit())):

data['SERVICER NAME'][i]=1

return data

Preprocessing the data.

* Builds a Regression model for the interest rate using Q12005 data as training data (col 13) Does variable selection to choose the best Regression model using Forward, Backward, Stepwise and Exhaustive search methods. Validates against the Q22005 datasets. Computes MAE, RMS, MAPE for training and testing datasets

def mean\_absolute\_percentage\_error(y\_true, y\_pred):

y\_true, y\_pred = np.array(y\_true), np.array(y\_pred)

return np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100

#Linear Regression

lin\_reg = LinearRegression()

lin\_reg.fit(train\_x,train\_y)

pred\_y = lin\_reg.predict(test\_x)

print("Score for train: "+str(lin\_reg.score(train\_x,train\_y)))

print("Score: "+str(lin\_reg.score(test\_x,test\_y)))

print("RMS: "+str(sqrt(metrics.mean\_squared\_error(test\_y,pred\_y))))

print("MAPE: "+str(mean\_absolute\_percentage\_error(test\_y,pred\_y)))

print("R2: "+str(metrics.r2\_score(test\_y,pred\_y)))

print("MAE: "+str(metrics.mean\_absolute\_error(test\_y,pred\_y)))

Above is an example of linear regression model. We tried several different models include Linear Regression, Lasso, Decision Tree Regression, Ridge Regression, Bayesion Ridge.

* Repeat this using Random Forest & Neural Network algorithms.

Random Forest

rf\_reg = RandomForestRegressor()

rf\_reg.fit(train\_x,train\_y)

pred\_y = rf\_reg.predict(test\_x)

predvali\_y = rf\_reg.predict(vali\_x)

print("Score for train: "+str(rf\_reg.score(train\_x,train\_y)))

print("Score: "+str(rf\_reg.score(test\_x,test\_y)))

print("Score for Validates: "+str(rf\_reg.score(vali\_x,vali\_y)))

print("RMS: "+str(sqrt(metrics.mean\_squared\_error(test\_y,pred\_y))))

print("MAPE: "+str(mean\_absolute\_percentage\_error(test\_y,pred\_y)))

print("R2: "+str(metrics.r2\_score(test\_y,pred\_y)))

print("MAE: "+str(metrics.mean\_absolute\_error(test\_y,pred\_y)))

Neural Network

transformer = Normalizer().fit(train\_x)

mlp\_train\_x = pd.DataFrame(transformer.transform(train\_x))

mlp\_train\_x.columns = train\_x.columns

mlp\_test\_x = Normalizer().fit(test\_x)

mlp\_test\_x = pd.DataFrame(transformer.transform(test\_x))

mlp\_test\_x.columns = test\_x.columns

mlp\_vali\_x = Normalizer().fit(vali\_x)

mlp\_vali\_x = pd.DataFrame(transformer.transform(vali\_x))

mlp\_vali\_x.columns = vali\_x.columns

#mlp

mlp\_reg2 = MLPRegressor(hidden\_layer\_sizes=(300,20),learning\_rate='adaptive',solver='adam',random\_state=3,learning\_rate\_init=0.001,max\_iter=400)

mlp\_reg.fit(mlp\_train\_x,train\_y)

pred\_y = mlp\_reg.predict(mlp\_test\_x)

predvali\_y = mlp\_reg.predict(mlp\_vali\_x)

print("Score for train: "+str(mlp\_reg.score(mlp\_train\_x,train\_y)))

print("Score: "+str(mlp\_reg.score(mlp\_test\_x,test\_y)))

print("Score for Validates: "+str(mlp\_reg.score(mlp\_vali\_x,vali\_y)))

print("RMS: "+str(sqrt(metrics.mean\_squared\_error(test\_y,pred\_y))))

print("MAPE: "+str(mean\_absolute\_percentage\_error(test\_y,pred\_y)))

print("R2: "+str(metrics.r2\_score(test\_y,pred\_y)))

print("MAE: "+str(metrics.mean\_absolute\_error(test\_y,pred\_y)))

* Try TPOT, H20.Ai and AutoSKLearn Automl algorithms

tpot = TPOTRegressor(generations=3,scoring='r2',population\_size=10)

tpot.fit(train\_x,train\_y)

tpot.score(test\_x,test\_y)

tpot.export('tpot\_assignment3.py')

h2o.init()

train = train\_x.copy()

train['ORIGINAL INTEREST RATE']=train\_y

train = h2o.H2OFrame(train)

test = test\_x.copy()

test['ORIGINAL INTEREST RATE']=test\_y

test = h2o.H2OFrame(test)

vali = vali\_x.copy()

vali['ORIGINAL INTEREST RATE']=vali\_y

vali = h2o.H2OFrame(vali)

aml = H2OAutoML(max\_runtime\_secs = 600, seed = 1, project\_name = "lending\_club")

aml.train(y = 'ORIGINAL INTEREST RATE', training\_frame = train, leaderboard\_frame = test)

aml.leaderboard.head()

* Choose the best model amongst the different types of algorithms.

The best model we get is Gradient Boosting Regression.

gbm\_reg = GradientBoostingRegressor(alpha=0.85, learning\_rate=0.1, loss="huber", max\_depth=10, max\_features=0.7, min\_samples\_leaf=18, min\_samples\_split=19, n\_estimators=100, subsample=0.4,verbose=2)

gbm\_reg.fit(train\_x,train\_y)

pred\_y = gbm\_reg.predict(test\_x)

print("Score for train: "+str(gbm\_reg.score(train\_x,train\_y)))

print("Score: "+str(gbm\_reg.score(test\_x,test\_y)))

print("RMS: "+str(sqrt(metrics.mean\_squared\_error(test\_y,pred\_y))))

print("MAPE: "+str(mean\_absolute\_percentage\_error(test\_y,pred\_y)))

print("R2: "+str(metrics.r2\_score(test\_y,pred\_y)))

print("MAE: "+str(metrics.mean\_absolute\_error(test\_y,pred\_y)))

* You are asked to do what-if analysis had your algorithm used in various scenarios:
* Financial crisis
* Run your algorithm for 4 rolling quarters and report your findings and discuss it in your report. (i.e Use Q12007, Q22007, Q32007, Q42007 for training and predict for Q22007, Q32007, Q42007, Q12008)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Train data | Validation data | RMS | MAPE | R2 | MAE |
| Q12007 | Q22007 | 0.3352401243090923 | 3.916927487675391 | 0.2684 | 0.2501980157915151 |
| Q22007 | Q32007 | 0.4791132294150997 | 5.923953649333251 | -0.5436 | 0.40251641096175517 |
| Q32007 | Q42007 | 0.46435446061447644 | 6.178674941610631 | -0.1010 | 0.38000884041383104 |
| Q42007 | Q12008 | 0.554393207159252 | 8.351641591136772 | -0.2060 | 0.47045738324609065 |

We found that the accuracy of the model drops rapidly from 2ed quarter, because of the influence of financial crisis

* Run your algorithm 2 years later (i.e, 2009 for all 4 quarters)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Train data | Validation data | RMS | MAPE | R2 | MAE |
| Q12009 | Q22009 | 0.2878 | 4.5366 | 0.3000 | 0.2194 |
| Q22009 | Q32009 | 0.4095 | 6.2581 | 0.0150 | 0.3313 |
| Q32009 | Q42009 | 0.3088 | 5.2760 | 0.2751 | 0.2540 |
| Q42009 | Q12010 | 0.2044 | 3.1715 | 0.6824 | 0.1572 |

Compare to above, the model performs pretty good if there isn’t a financial crisis.

* Economic boom (1999, 2013)
* Discuss your design and results in a report. Would you recommend using this model for the next quarter? Justify

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Train data | Validation data | RMS | MAPE | R2 | MAE |
| Q11999 | Q21999 | 0.4169 | 4.2585 | -0.2088 | 0.3132 |
| Q21999 | Q31999 | 0.7315 | 8.0781 | -1.7689 | 0.6418 |

The model performs bad during a economic boom. I don’t recommend using this model for the next quarter, because the interest rate changes with the economic conditions. We need to consider the economic conditions when built our model. Only predicting interest rates in the next quarter based on information from the origination data from the prior quarter won’t get a good result in many cases.

* Regime change (2016) from election

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Train data | Validation data | RMS | MAPE | R2 | MAE |
| Q12016 | Q22016 | 0.2688 | 5.9976 | 0.6034 | 0.2205 |
| Q22016 | Q32016 | 0.2628 | 5.9924 | 0.6442 | 0.2123 |
| Q32016 | Q42016 | 0.2765 | 5.3891 | 0.6754 | 0.2026 |
| Q42016 | Q12017 | 0.5821 | 12.3629 | -0.5579 | 0.5300 |

The model works well in the first three quarters, but when election happened in 4th quarter, the accuracy drop rapidly.