HDS5230 Final Exam - programming - Miao Cai

May 12, 2019

1 HDS 5230 High Performance Computing

1.1 Final Exam - Progamming Part

Miao Cai miao.cai@slu.edu

2 Introduction

The big goal of this project is to use the provided dataset on health insurance charges to create a model that predicts charges as accurately as possible, based on the patient traits of age, sex, bmi, children, smoker, and region. As I generate this model, I performed and documented initial data quality checks, exploratory data analysis, and all of the models I tried to fit.

3 Methods summary

All the data cleaning, visualization, and modeling were conducted in Python, and this reported was wrote in jupyternotebook. The Python session and package version information is shown below.

In [38]:
```
import os
import sys
import pathlib
from tableone import TableOne
import pandas as pd
import numpy as np
import seaborn as sns
import h2o
from h2o.estimators.glm import H2OGeneralizedLinearEstimator

print(sys.version)
print("Pandas version: {0}".format(pd.__version__))
print("Numpy version:{0}".format(np.__version__))
print("Seaborn version:{0}".format(sns.__version__))
print("h2o version:{0}".format(h2o.__version__))
print("My working directory:\n" + os.getcwd())
```

```
3.7.1 (default, Dec 14 2018, 13:28:58)
[Clang 4.0.1 (tags/RELEASE_401/final)]
Pandas version: 0.23.4
Numpy version:1.15.4
Seaborn version:0.9.0
h2o version:3.22.1.3
My working directory:
/Users/miaocai/Dropbox/@2018 SPRING HDS5230 High performance computing/HDS5230Homework/Final e
```

3.0.1 Loss function

I pick the loss function as the mean absolute error (MAE). I chose this loss function since the outcome variable charges are highly right-skewed. Using the most commonly used mean square error (MSE) will not be as robust as MAE since the error will be squared, which is strongly influenced by outliers.

3.0.2 Initiate h2o and read data

```
In [39]: h2o.init(ip='localhost', port=54321, nthreads=-1, max_mem_size='2G')
```

```
Checking whether there is an H2O instance running at http://localhost:54321. connected.
Warning: Your H2O cluster version is too old (3 months and 17 days)! Please download and insta
```

```
--------------------------  ----------------------------------------
H2O cluster uptime:         18 hours 33 mins
H2O cluster timezone:       America/Chicago
H2O data parsing timezone:  UTC
H2O cluster version:        3.22.1.3
H2O cluster version age:    3 months and 17 days !!!
H2O cluster name:           H2O_from_python_miaocai_k7904q
H2O cluster total nodes:    1
H2O cluster free memory:    1.963 Gb
H2O cluster total cores:    8
H2O cluster allowed cores:  8
H2O cluster status:         locked, healthy
H2O connection url:         http://localhost:54321
H2O connection proxy:
H2O internal security:      False
H2O API Extensions:         XGBoost, Algos, AutoML, Core V3, Core V4
Python version:             3.7.1 final
--------------------------  ----------------------------------------
```

```
In [40]: d0 = h2o.import_file("insurance.csv")
         d0
```

```
Parse progress: || 100%
```

Out[40]:

```
In [41]: print("The shape of the DataFrame is:")
         print(d0.shape)
         list(zip(d0.nacnt(), d0.names))
```

```
The shape of the DataFrame is:
(1338, 7)
```

```
Out[41]: [(0.0, 'age'),
          (0.0, 'sex'),
          (0.0, 'bmi'),
          (0.0, 'children'),
          (0.0, 'smoker'),
          (0.0, 'region'),
          (0.0, 'charges')]
```

After reading the .csv file by using h2o, I found that there are no missing values in the data, which is great!

3.0.3 Split into train/validation/testing splits

We then need to convert some categorical variables into factors, and split the data into train, test, and validation sets.

```
In [42]: d0[['sex', 'smoker', 'region']] = d0[['sex', 'smoker', 'region']].asfactor()
         dtrain, dtest, dvalid = \
             d0.split_frame([0.7, 0.15], seed = 666)
```

4 Results

4.1 Summary statistics of model input variables

```
In [43]: d = pd.read_csv("insurance.csv")
         col_types = d.dtypes.to_dict()
         col_types['age'] = 'float64'
         d = pd.read_csv("insurance.csv", dtype=col_types)

         overall_table = TableOne(
             d, columns = ['age', 'bmi', 'children', 'smoker', 'region'],
             categorical = ['children', 'smoker', 'region'],
             groupby = 'sex', label_suffix=True, pval = True)
         overall_table
```

```
Out[43]:                           Grouped by sex
                                           isnull       female          male    pval
         variable        level
         n                                                 662           676
```

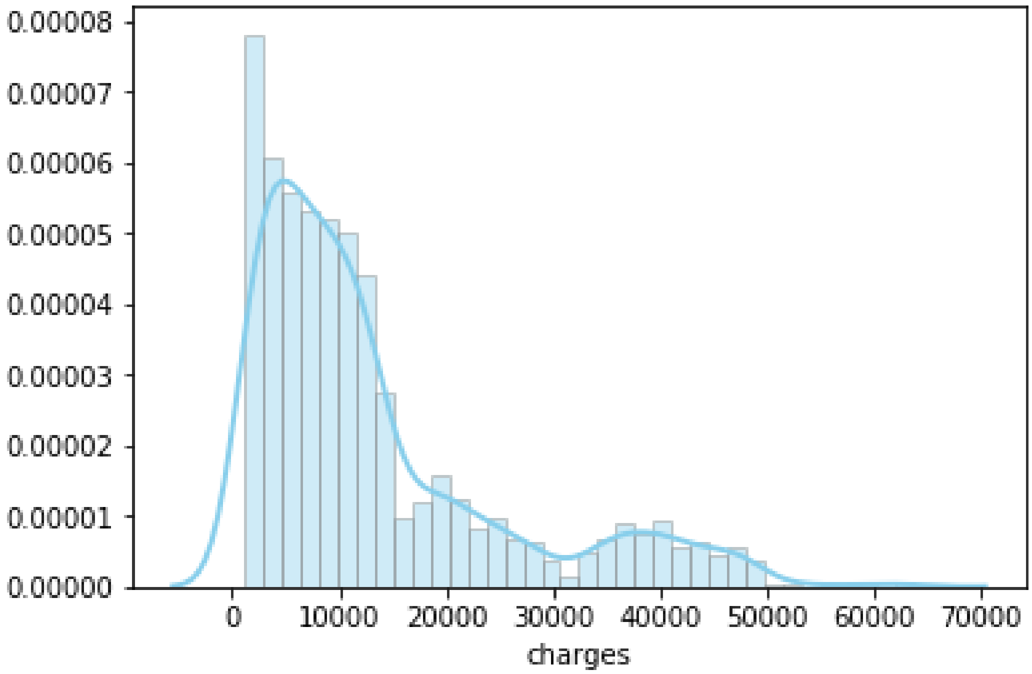
```
age, mean (SD)                   0  39.5 (14.1)  38.9 (14.1)  0.446  Two Sample
bmi, mean (SD)                   0   30.4 (6.0)   30.9 (6.1)  0.090  Two Sample
children, n (%) 0                0   289 (43.7)   285 (42.2)  0.981        Chi-
                1                    158 (23.9)   166 (24.6)
                2                    119 (18.0)   121 (17.9)
                3                     77 (11.6)    80 (11.8)
                4                      11 (1.7)     14 (2.1)
                5                       8 (1.2)     10 (1.5)
smoker, n (%)   no               0   547 (82.6)   517 (76.5)  0.007        Chi-
                yes                  115 (17.4)   159 (23.5)
region, n (%)   northeast        0   161 (24.3)   163 (24.1)  0.933        Chi-
                northwest            164 (24.8)   161 (23.8)
                southeast            175 (26.4)   189 (28.0)
                southwest            162 (24.5)   163 (24.1)
[1] Warning, Hartigan's Dip Test reports possible multimodal distributions for: age.
[2] Warning, test for normality reports non-normal distributions for: age.
```

4.2 Descriptive analysis of the outcome variable charges

In [44]:
```python
sns.distplot(d['charges'], hist=True, kde=True,
             bins=int(180/5), color = 'skyblue',
             hist_kws={'edgecolor':'grey'},
             kde_kws={'linewidth': 2})
```

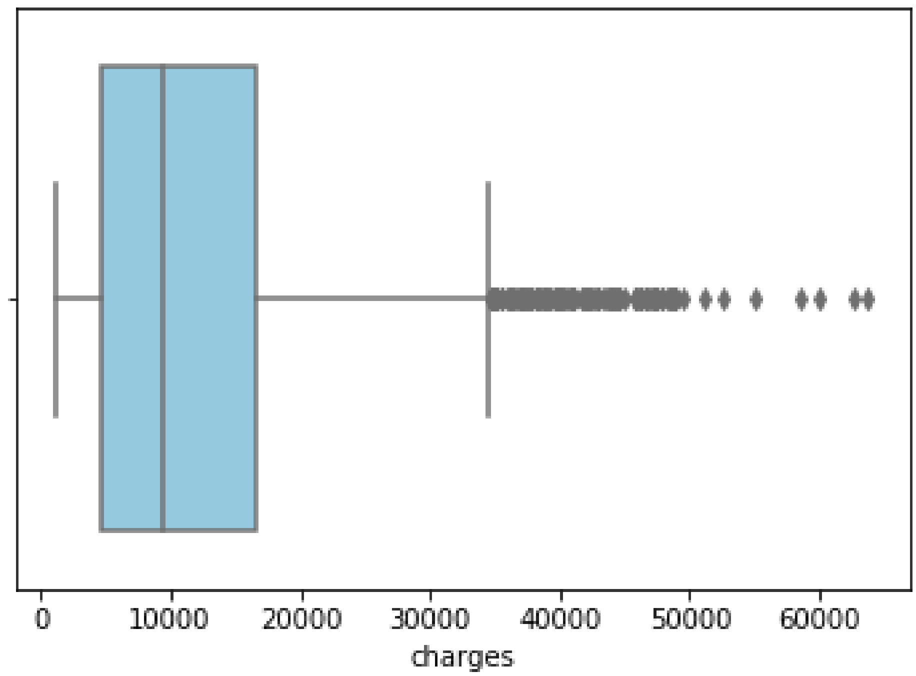
```
/Users/miaocai/anaconda3/lib/python3.7/site-packages/scipy/stats/stats.py:1713: FutureWarning:
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[44]: `<matplotlib.axes._subplots.AxesSubplot at 0x1a3997ec50>`



```
In [45]: sns.boxplot(d['charges'], color = 'skyblue')
```

```
Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x1a39a7ce80>
```




```
In [46]: d['charges'].describe()
```

```
Out[46]: count     1338.000000
         mean     13270.422265
         std      12110.011237
         min       1121.873900
         25%       4740.287150
         50%       9382.033000
         75%      16639.912515
         max      63770.428010
         Name: charges, dtype: float64
```

4.3 Cross-validation

Here I use the `H2OGeneralizedLinearEstimator` module in `h2o` package to perform cross-validation. I set the number of cross-validation as 5 by `nfolds=5`, in which 6 models will be built. The first 5 models (cross-validation models) are built on 80% of the training data, and a different 20% is held out for each of the 5 models. Then the main model is built on 100% of the training data.

```
In [47]: target_var = 'charges'
         input_var = ['age', 'sex', 'bmi',
                      'children', 'smoker', 'region']
         glm_mod0 = \
             H2OGeneralizedLinearEstimator(model_id = 'd_glm_0',
                                           family = 'gaussian',
                                           nfolds = 5,
                                           seed = 123)
         glm_mod0.train(x = input_var,
                        y = target_var,
                        training_frame = d0)
         glm_mod0.show()
```

```
glm Model Build progress: || 100%
Model Details
=============
H2OGeneralizedLinearEstimator :  Generalized Linear Modeling
Model Key:  d_glm_0


ModelMetricsRegressionGLM: glm
** Reported on train data. **

MSE: 125728309.44330035
RMSE: 11212.863570172445
MAE: 8471.397911142334
RMSLE: 0.9331157151413909
R^2: 0.14203674139265943
Mean Residual Deviance: 125728309.44330035
Null degrees of freedom: 1337
Residual degrees of freedom: 1326
Null deviance: 196074221532.74988
Residual deviance: 168224478035.13586
AIC: 28776.289622947068

ModelMetricsRegressionGLM: glm
** Reported on cross-validation data. **

MSE: 129830749.84711617
RMSE: 11394.32972346843
MAE: 8600.97548619905
RMSLE: 0.9464119017196216
R^2: 0.11404190751089327
Mean Residual Deviance: 129830749.84711617
Null degrees of freedom: 1337
Residual degrees of freedom: 1326
Null deviance: 196632689167.88873
Residual deviance: 173713543295.44144
```

```
AIC: 28819.25062652632
Cross-Validation Metrics Summary:
```

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_
mae	8592.55	295.311	8582.31	7818.53	8785.11	87
mean_residual_deviance	1.29276e+08	1.25685e+07	1.22953e+08	1.02176e+08	1.53851e+08	1.
mse	1.29276e+08	1.25685e+07	1.22953e+08	1.02176e+08	1.53851e+08	1.
null_deviance	3.93265e+10	4.53807e+09	3.72839e+10	2.96943e+10	4.90307e+10	3.
r2	0.110559	0.00589982	0.117462	0.114765	0.100491	0.
residual_deviance	3.47427e+10	4.17128e+09	3.27055e+10	2.6157e+10	4.38475e+10	3.
rmse	11342.6	557.598	11088.4	10108.2	12403.7	11
rmsle	0.945845	0.0343231	0.981383	0.914074	0.884154	1.

```
Scoring History:
```

	timestamp	duration	iterations	negative_log_likelihood	objective
	2019-05-12 22:43:32	0.000 sec	0	1.96074e+11	1.46543e+08

4.4 Machine learning

4.4.1 Regularized regression

```
In [48]: from h2o.grid.grid_search import H2OGridSearch

         alpha_values = {'alpha': [0, 0.25, 0.5, 0.75, 1]}
         glm_mod2 = H2OGridSearch(
             H2OGeneralizedLinearEstimator(family = 'gaussian',
                                           lambda_search = True,
                                           seed = 123), \
         hyper_params = alpha_values)

         glm_mod2.train(x = input_var,
                        y = target_var,
                        training_frame = dtrain,
                       validation_frame = dvalid)

         glm_mod2.show()
```

```
glm Grid Build progress: || 100%
     alpha                                                  model_ids  \
0     [1.0]  Grid_GLM_py_10_sid_929a_model_python_1557652172496_3_model_5
1    [0.75]  Grid_GLM_py_10_sid_929a_model_python_1557652172496_3_model_4
2     [0.5]  Grid_GLM_py_10_sid_929a_model_python_1557652172496_3_model_3
```

```
3    [0.25]  Grid_GLM_py_10_sid_929a_model_python_1557652172496_3_model_2
4     [0.0]  Grid_GLM_py_10_sid_929a_model_python_1557652172496_3_model_1

       residual_deviance
0    6.232294045221491E9
1   2.7448416102541004E10
2   2.7451841045226257E10
3   2.7452983289290577E10
4   2.7453233838145943E10
```

In [49]:
```python
glm_best = glm_mod2.get_grid()[0]
glm_best._model_json['output']['coefficients_table'].as_data_frame()
```

Out[49]:

	names	coefficients	standardized_coefficients
0	Intercept	-7206.776563	13492.999531
1	region.northeast	83.984427	83.984427
2	region.northwest	111.459665	111.459665
3	region.southeast	-192.376821	-192.376821
4	region.southwest	-70.256674	-70.256674
5	smoker.no	-5020.292788	-5020.292788
6	smoker.yes	18846.423625	18846.423625
7	sex.female	147.142857	147.142857
8	sex.male	-142.895434	-142.895434
9	age	255.041295	3604.495432
10	bmi	329.352274	2038.979286
11	children	477.291820	563.040100

In [50]:
```python
glm_best._model_json['output']['training_metrics']
```

```
ModelMetricsRegressionGLM: glm
** Reported on train data. **

MSE: 36244089.05252515
RMSE: 6020.306391914381
MAE: 4175.590832941357
RMSLE: NaN
R^2: 0.7557379707834732
Mean Residual Deviance: 36244089.05252515
Null degrees of freedom: 955
Residual degrees of freedom: 944
Null deviance: 141853194478.7002
Residual deviance: 34649349134.21404
AIC: 19378.942718761107
```

Out[50]:

In [51]:
```
glm_best._model_json['output']['validation_metrics']
```

```
ModelMetricsRegressionGLM: glm
** Reported on validation data. **

MSE: 30107700.701553095
RMSE: 5487.04845081152
MAE: 3827.8094066015087
RMSLE: 0.5274009774363978
R^2: 0.7699240420817869
Mean Residual Deviance: 30107700.701553095
Null degrees of freedom: 206
Residual degrees of freedom: 195
Null deviance: 27453554523.254715
Residual deviance: 6232294045.221491
AIC: 4178.040900338396
```

Out[51]:

4.4.2 Auto-ML

In [52]:
```
from h2o.automl.autoh2o import H2OAutoML

autom1_1 = H2OAutoML(max_runtime_secs = 60)
autom1_1.train(x = input_var,
               y = target_var,
               training_frame = dtrain,
               validation_frame = dvalid)
autom1_1.leaderboard
```

```
AutoML progress: || 100%
```

Out[52]:

In [53]:
```
autom1_1.leader
```

```
Model Details
=============
H2OXGBoostEstimator :  XGBoost
Model Key:  XGBoost_grid_1_AutoML_20190512_224333_model_2


ModelMetricsRegression: xgboost
** Reported on train data. **

MSE: 14528508.374148928
```

RMSE: 3811.6280477177897
MAE: 2022.629675446195
RMSLE: 0.33941648050977435
Mean Residual Deviance: 14528508.374148928

ModelMetricsRegression: xgboost
** Reported on validation data. **

MSE: 18054575.204083566
RMSE: 4249.067568783011
MAE: 2344.0509905976373
RMSLE: 0.38510422313361437
Mean Residual Deviance: 18054575.204083566

ModelMetricsRegression: xgboost
** Reported on cross-validation data. **

MSE: 19784318.805394568
RMSE: 4447.956700035936
MAE: 2432.7904463891705
RMSLE: 0.41318440454888644
Mean Residual Deviance: 19784318.805394568
Cross-Validation Metrics Summary:

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_
mae	2432.73	74.7078	2490.93	2286.08	2333.17	248
mean_residual_deviance	1.97816e+07	1.54879e+06	2.23829e+07	1.95263e+07	1.74532e+07	1.7
mse	1.97816e+07	1.54879e+06	2.23829e+07	1.95263e+07	1.74532e+07	1.7
r2	0.864809	0.0109681	0.872214	0.836048	0.88118	0.8
residual_deviance	1.97816e+07	1.54879e+06	2.23829e+07	1.95263e+07	1.74532e+07	1.7
rmse	4440.84	174.036	4731.05	4418.86	4177.71	416
rmsle	0.412534	0.0165132	0.403937	0.438595	0.418279	0.3

Scoring History:

	timestamp	duration	number_of_trees	training_rmse	training_mae
	2019-05-12 22:43:58	8.165 sec	0.0	18177.721142805927	13492.49952206152
	2019-05-12 22:43:58	8.176 sec	5.0	14425.196741755824	10417.29248066028
	2019-05-12 22:43:58	8.184 sec	10.0	11562.46886923096	8073.296127255491
	2019-05-12 22:43:58	8.195 sec	15.0	9388.860994598781	6304.479055620138
	2019-05-12 22:43:58	8.203 sec	20.0	7765.6711329633945	5018.130319252174
---	---	---	---	---	---
	2019-05-12 22:43:58	8.333 sec	80.0	3900.143246524783	2049.870979851758

```
2019-05-12 22:43:58  8.347 sec   85.0              3872.9667307722375  2046.1287794552088
2019-05-12 22:43:58  8.361 sec   90.0              3851.652535206911   2031.3362383343685
2019-05-12 22:43:58  8.376 sec   95.0              3828.628867905125   2024.7918644988886
2019-05-12 22:43:58  8.392 sec   99.0              3811.6280477177897  2022.629675446195
```

See the whole table with table.as_data_frame()
Variable Importances:

variable	relative_importance	scaled_importance	percentage
smoker.no	4.54323e+11	1	0.55736
bmi	1.48792e+11	0.327503	0.182537
age	1.02772e+11	0.226209	0.12608
smoker.yes	9.05798e+10	0.199373	0.111123
children	8.18582e+09	0.0180176	0.0100423
region.southeast	3.67247e+09	0.00808339	0.00450536
region.northeast	2.91581e+09	0.00641794	0.0035771
sex.female	1.9189e+09	0.00422366	0.0023541
region.southwest	1.10821e+09	0.00243926	0.00135954
sex.male	4.73685e+08	0.00104262	0.000581113
region.northwest	3.91876e+08	0.000862549	0.000480751

Out[53]:

In [54]:
```
autom1_1.leader.cross_validation_metrics_summary()
```

Cross-Validation Metrics Summary:

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_
mae	2432.73	74.7078	2490.93	2286.08	2333.17	248
mean_residual_deviance	1.97816e+07	1.54879e+06	2.23829e+07	1.95263e+07	1.74532e+07	1.7
mse	1.97816e+07	1.54879e+06	2.23829e+07	1.95263e+07	1.74532e+07	1.7
r2	0.864809	0.0109681	0.872214	0.836048	0.88118	0.8
residual_deviance	1.97816e+07	1.54879e+06	2.23829e+07	1.95263e+07	1.74532e+07	1.7
rmse	4440.84	174.036	4731.05	4418.86	4177.71	416
rmsle	0.412534	0.0165132	0.403937	0.438595	0.418279	0.3

Out[54]:

4.5 Neural networks using kera and tensorflow

In [55]:
```
from keras import models
from keras import layers
```

```python
from keras import optimizers
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

d = pd.read_csv("insurance.csv")
col_types = d.dtypes.to_dict()
col_types['age'] = 'float64'
d = pd.read_csv("insurance.csv", dtype=col_types)

d_dummy = pd.get_dummies(d, drop_first = True)
y = d_dummy['charges'].values
X = d_dummy.drop('charges', axis = 1).values

X_train0, X_test0 = train_test_split(
    X, train_size = 0.8, random_state = 123)
y_train0, y_test0 = train_test_split(
    y, train_size = 0.8, random_state = 123)
X_train1, X_valid1 = train_test_split(
    X, train_size = 0.75, random_state = 123)
y_train1, y_valid1 = train_test_split(
    y, train_size = 0.75, random_state = 123)

# logistic regression
def nn_model1():
    model = models.Sequential()
    model.add(layers.Dense(64, activation = 'relu',
                           input_dim = X_train1.shape[1]))
    model.add(layers.Dense(1,
                           activation = 'linear'))
    model.compile(optimizer = 'rmsprop',
                  loss = 'mse',
                  metrics = ['mae'])
    return model

reg0 = nn_model1()
reg0.fit(X_train1, y_train1, epochs = 200,
         validation_data = [X_valid1, y_valid1],
        verbose = 0)
```

```
/Users/miaocai/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:2179: Fu
  FutureWarning)
```

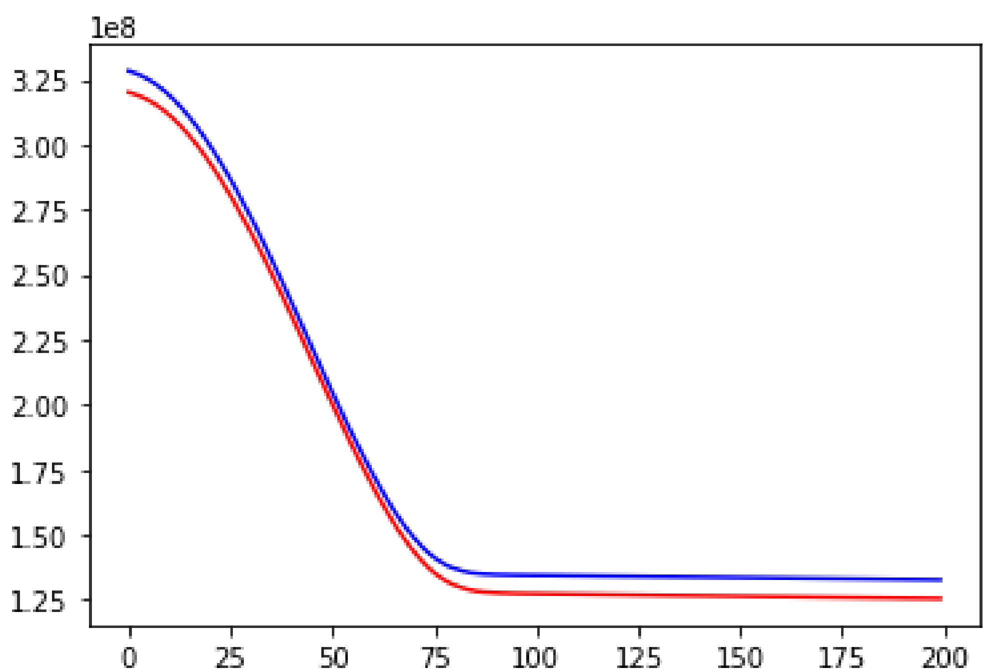
```
Out[55]: <keras.callbacks.History at 0x1a39ef4ef0>
```

```python
In [56]: reg0.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

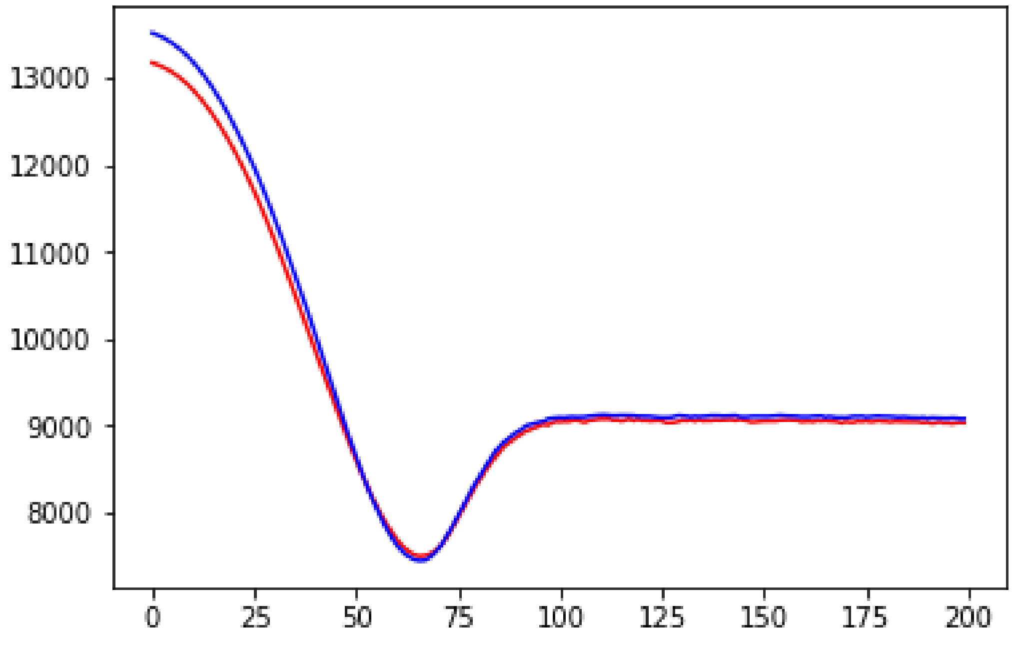
```
=================================================================
dense_18 (Dense)             (None, 64)                576
_________________________________________________________________
dense_19 (Dense)             (None, 1)                 65
=================================================================
Total params: 641
Trainable params: 641
Non-trainable params: 0
_________________________________________________________________
```

```
In [57]: reg0.history.history['loss'][199]

Out[57]: 125524612.95314057

In [58]: reg0.history.history['val_loss'][199]

Out[58]: 132704983.57014926

In [59]: reg0.history.history['val_mean_absolute_error'][199]

Out[59]: 9081.118927821828

In [60]: plt.plot(reg0.history.history['loss'], color = 'red')
         plt.plot(reg0.history.history['val_loss'], color = 'blue')

Out[60]: [<matplotlib.lines.Line2D at 0x1a3a48a470>]
```



```
In [61]: plt.plot(reg0.history.history['mean_absolute_error'], color = 'red')
         plt.plot(reg0.history.history['val_mean_absolute_error'], color = 'blue')
```

```
Out[61]: [<matplotlib.lines.Line2D at 0x1a3a4d5f60>]
```

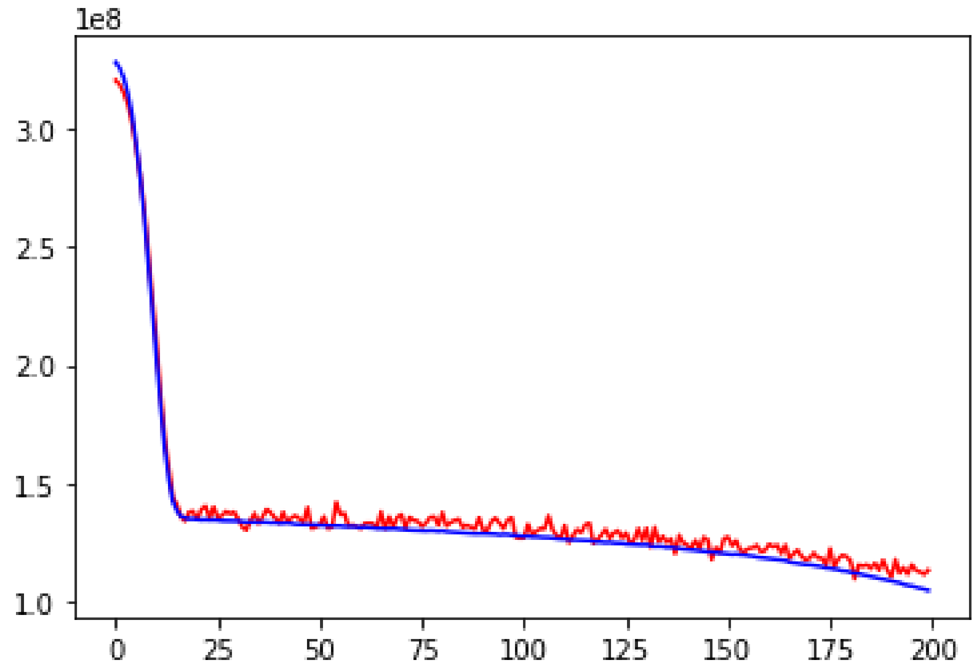



```
In [62]: def nn_model2():
             model = models.Sequential()
             model.add(layers.Dense(64, activation = 'relu',
                                    input_dim = X_train1.shape[1]))
             model.add(layers.Dropout(0.5))
             model.add(layers.Dense(64, activation = 'relu'))
             model.add(layers.Dropout(0.5))
             model.add(layers.Dense(1, activation = 'linear'))
             model.compile(optimizer = 'rmsprop',
                           loss = 'mse',
                           metrics = ['mae'])
             return model

         reg2 = nn_model2()
         reg2.fit(X_train1, y_train1, epochs = 200,
                  validation_data = [X_valid1, y_valid1],
                 verbose = 0)
```

```
Out[62]: <keras.callbacks.History at 0x1a3a4f16d8>
```

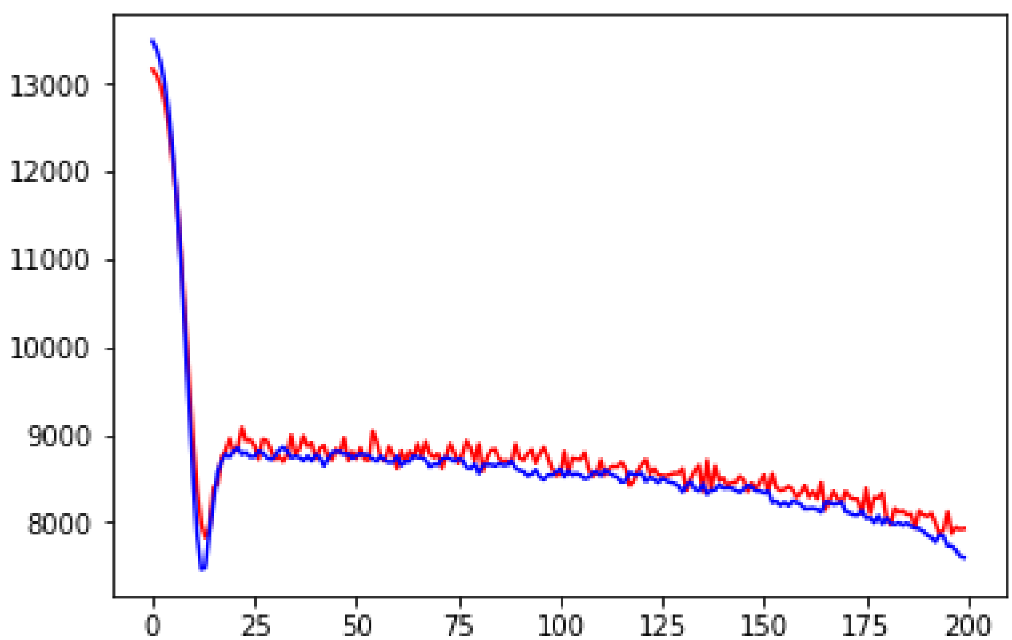
```
In [63]: plt.plot(reg2.history.history['loss'], color = 'red')
         plt.plot(reg2.history.history['val_loss'], color = 'blue')
```

Out[63]: [<matplotlib.lines.Line2D at 0x1a3a9d5748>]




```
In [64]: plt.plot(reg2.history.history['mean_absolute_error'], color = 'red')
         plt.plot(reg2.history.history['val_mean_absolute_error'], color = 'blue')
```

Out[64]: [<matplotlib.lines.Line2D at 0x1a3aa99e48>]

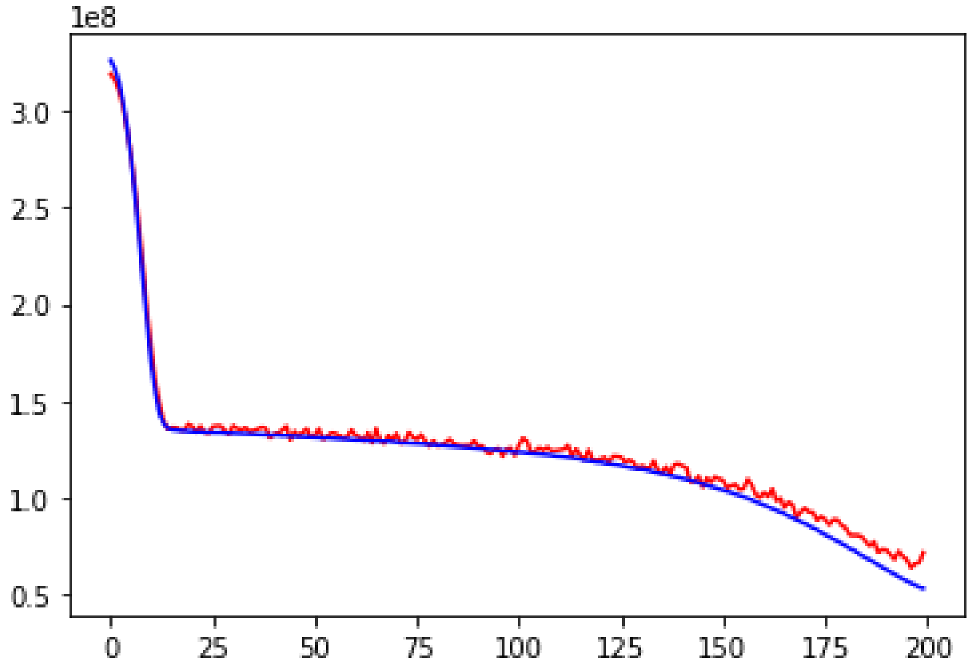


4.5.1 A more complex model with dropout, L1 regularization, smaller weight

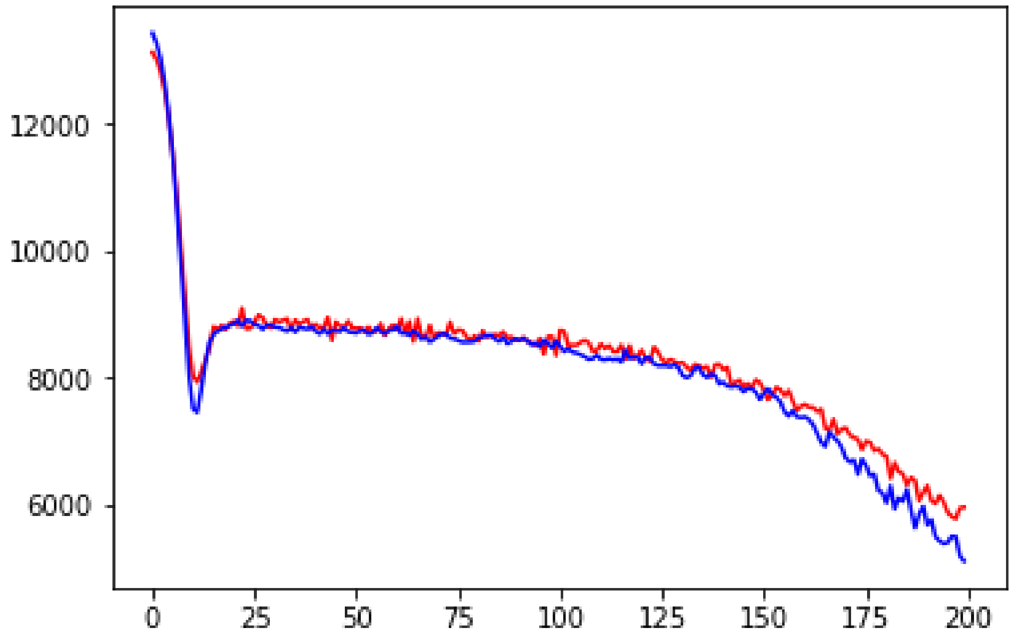
```
In [65]: from keras.regularizers import l1

         def nn_model3(l1_penalty):
             model = models.Sequential()
             model.add(layers.Dense(64, activation = 'relu',
                                    kernel_regularizer =
                                        l1(l1_penalty),
                                   input_dim = X_train1.shape[1]))
             model.add(layers.Dropout(0.5))
             model.add(layers.Dense(64, activation = 'relu',
                                   kernel_regularizer = l1(l1_penalty)))
             model.add(layers.Dropout(0.5))
             model.add(layers.Dense(1, activation = 'linear'))
             model.compile(optimizer = 'rmsprop',
                          loss = 'mse',
                          metrics = ['mae'])
             return model

In [66]: reg3 = nn_model3(0.1)
         reg3.fit(X_train1, y_train1, epochs = 200,
                  validation_data = [X_valid1, y_valid1],
                 verbose = 0)
         reg3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_23 (Dense)             (None, 64)                576
_________________________________________________________________
dropout_13 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_24 (Dense)             (None, 64)                4160
_________________________________________________________________
dropout_14 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_25 (Dense)             (None, 1)                 65
=================================================================
Total params: 4,801
Trainable params: 4,801
Non-trainable params: 0
_________________________________________________________________
```

```
In [67]: plt.plot(reg3.history.history['loss'], color = 'red')
         plt.plot(reg3.history.history['val_loss'], color = 'blue')

Out[67]: [<matplotlib.lines.Line2D at 0x1a3b03b128>]
```

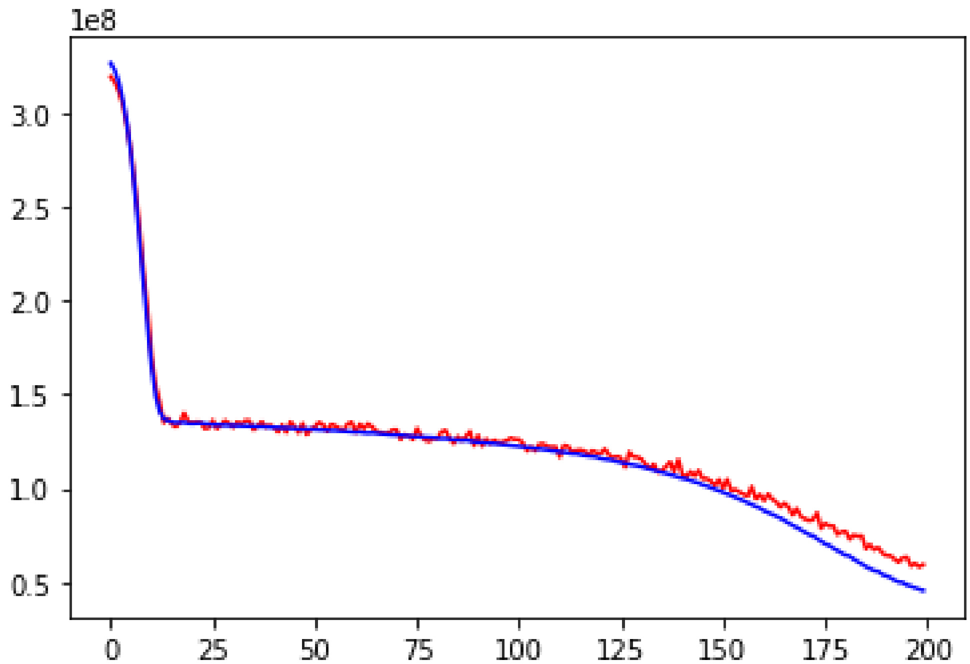



```
In [68]: plt.plot(reg3.history.history['mean_absolute_error'], color = 'red')
         plt.plot(reg3.history.history['val_mean_absolute_error'], color = 'blue')

Out[68]: [<matplotlib.lines.Line2D at 0x1a3b0ffc50>]
```

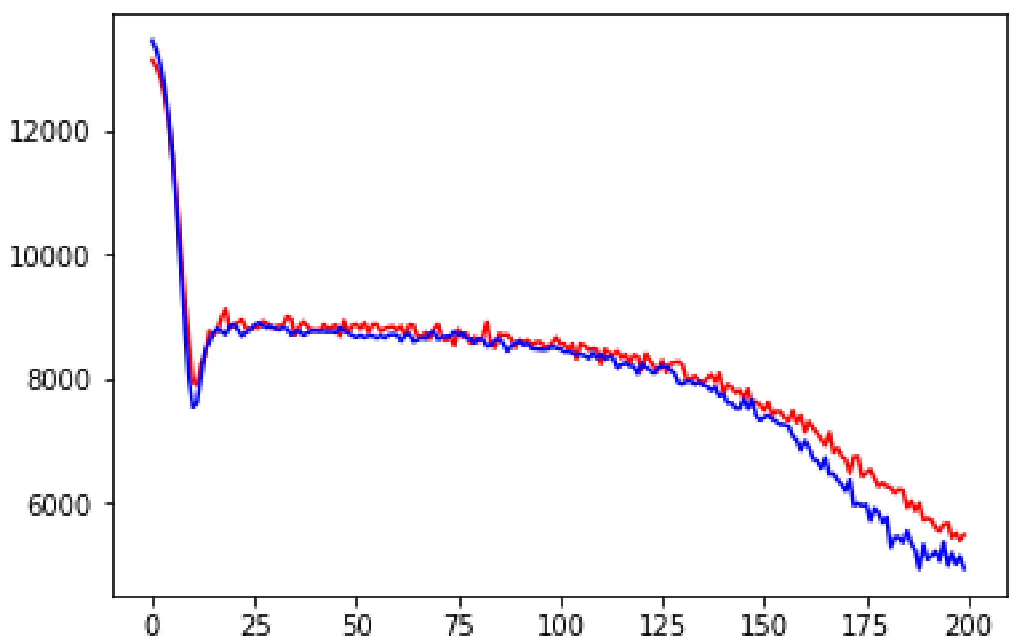



```
In [69]: reg3 = nn_model3(0.2)
         reg3.fit(X_train1, y_train1, epochs = 200,
                  validation_data = [X_valid1, y_valid1],
                 verbose = 0)
         plt.plot(reg3.history.history['loss'], color = 'red')
         plt.plot(reg3.history.history['val_loss'], color = 'blue')
```

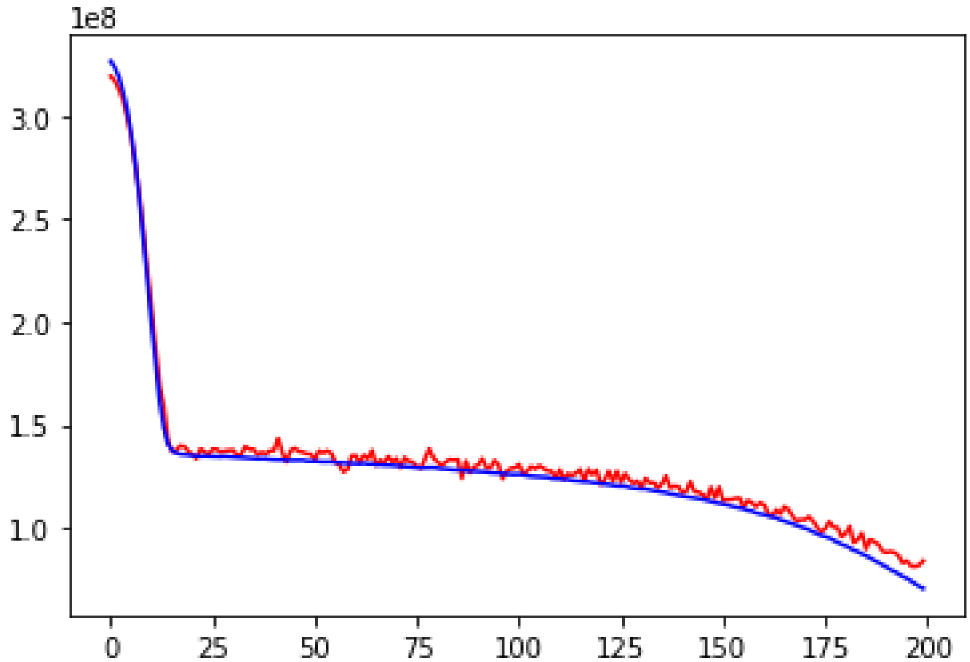
```
Out[69]: [<matplotlib.lines.Line2D at 0x1a3b6834a8>]
```



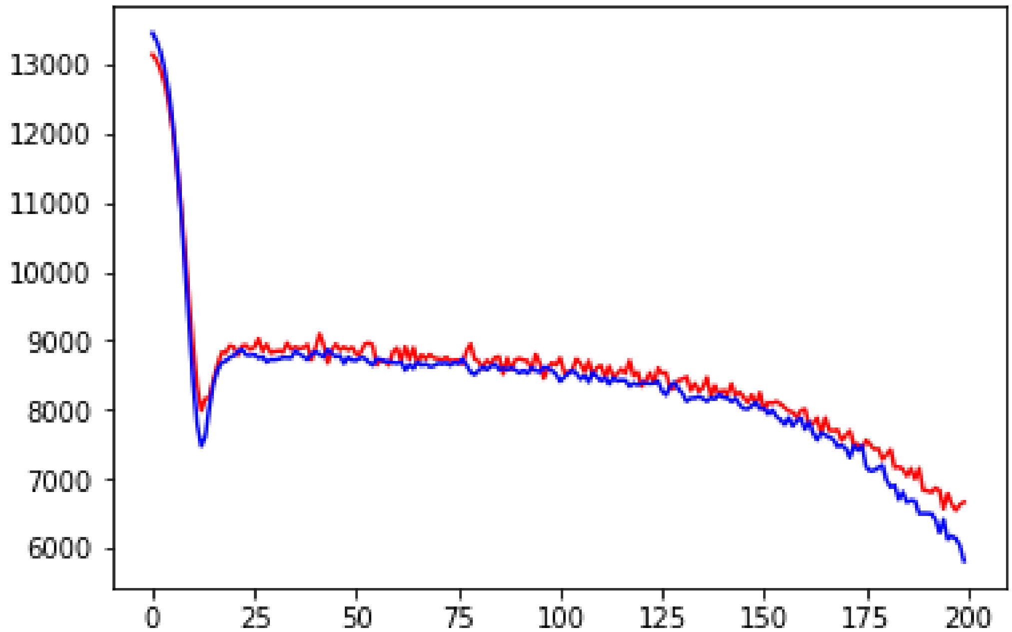

```
In [70]: plt.plot(reg3.history.history['mean_absolute_error'], color = 'red')
         plt.plot(reg3.history.history['val_mean_absolute_error'], color = 'blue')

Out[70]: [<matplotlib.lines.Line2D at 0x1a3b6e5c88>]
```



```
In [71]: reg3 = nn_model3(0.3)
         reg3.fit(X_train1, y_train1, epochs = 200,
                  validation_data = [X_valid1, y_valid1],
                 verbose = 0)
         plt.plot(reg3.history.history['loss'], color = 'red')
         plt.plot(reg3.history.history['val_loss'], color = 'blue')

Out[71]: [<matplotlib.lines.Line2D at 0x1a23530470>]
```




```
In [72]: plt.plot(reg3.history.history['mean_absolute_error'], color = 'red')
         plt.plot(reg3.history.history['val_mean_absolute_error'], color = 'blue')

Out[72]: [<matplotlib.lines.Line2D at 0x1a3a466e48>]
```



5 Conclusion

Based on the machine learning and deep learning models used above, I found that using XGBoost provided in `auto-ML` tool provided by `h2o` tends to have the best model prediction performance. With regard to deep learning models, a two-layer neural network with L1 regularization seems to have the best prediction performance.