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 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 you generate this model, you should perform and document initial data quality checks, exploratory data analysis, and all of the models you try 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 [1]: 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 [2]: h2o.init(ip='localhost', port=54321, nthreads=-1, max_mem_size='2G')
```

```
Checking whether there is an H2O instance running at http://localhost:54321... not found.
Attempting to start a local H2O server...
  Java Version: openjdk version "11.0.2" 2019-01-15; OpenJDK Runtime Environment 18.9 (build 1
  Starting server from /Users/miaocai/anaconda3/lib/python3.7/site-packages/h2o/backend/bin/h2
  Ice root: /var/folders/ng/t3l4gg0s0l398768705lv8vm0000gn/T/tmp8sc0eobc
  JVM stdout: /var/folders/ng/t3l4gg0s0l398768705lv8vm0000gn/T/tmp8sc0eobc/h2o_miaocai_started
  JVM stderr: /var/folders/ng/t3l4gg0s0l398768705lv8vm0000gn/T/tmp8sc0eobc/h2o_miaocai_started
  Server is running at http://127.0.0.1:54321
Connecting to H2O server at http://127.0.0.1:54321... successful.
Warning: Your H2O cluster version is too old (3 months and 16 days)! Please download and insta
```

```
--------------------------  ----------------------------------------
H2O cluster uptime:         01 secs
H2O cluster timezone:       America/Chicago
H2O data parsing timezone:  UTC
H2O cluster version:        3.22.1.3
H2O cluster version age:    3 months and 16 days !!!
H2O cluster name:           H2O_from_python_miaocai_k7904q
H2O cluster total nodes:    1
H2O cluster free memory:    2 Gb
H2O cluster total cores:    8
H2O cluster allowed cores:  8
H2O cluster status:         accepting new members, healthy
H2O connection url:         http://127.0.0.1:54321
H2O connection proxy:
H2O internal security:      False
H2O API Extensions:         XGBoost, Algos, AutoML, Core V3, Core V4
```

```
Python version:              3.7.1 final
--------------------------   ----------------------------------------
```

```
In [3]: d0 = h2o.import_file("insurance.csv")
        d0
```

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

```
Out[3]:
```

```
In [4]: 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[4]: [(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 [5]: 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 [6]: 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[6]:

```
                           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 T
bmi, mean (SD)                          0   30.4 (6.0)   30.9 (6.1)  0.090  Two Sample T
children, n (%) 0                       0  289 (43.7)   285 (42.2)   0.981        Chi-sq
                 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-sq
                 yes                       115 (17.4)   159 (23.5)
region, n (%)    northeast              0  161 (24.3)   163 (24.1)   0.933        Chi-sq
                 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.
```

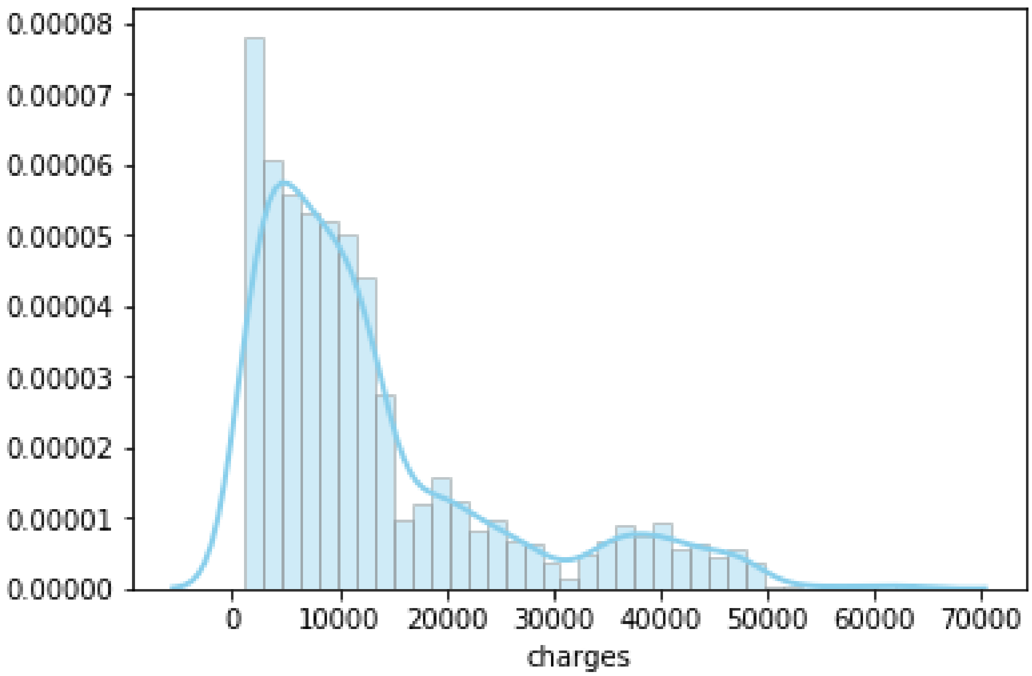
4.2 Descriptive analysis of the outcome variable charges

In [7]:

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

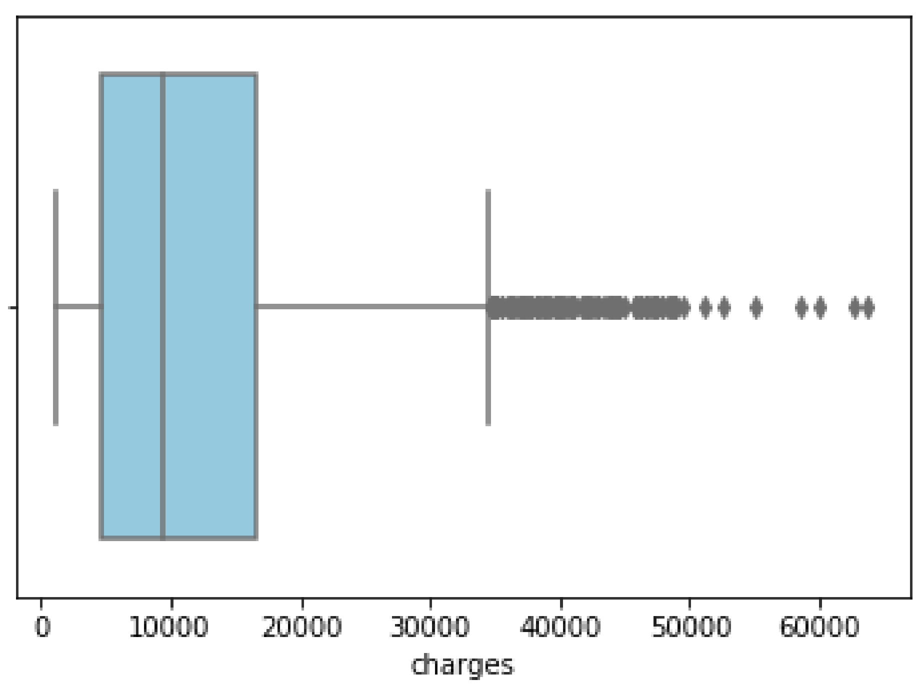
```
/Users/miaocai/anaconda3/lib/python3.7/site-packages/scipy/stats/stats.py:1713: FutureWarning:
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1badc160>




```
In [8]: sns.boxplot(d['charges'], color = 'skyblue')
```

```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1c00b978>
```



```
In [9]: d['charges'].describe()

Out[9]: 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 [10]: 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	871
mean_residual_deviance	1.29276e+08	1.25685e+07	1.22953e+08	1.02176e+08	1.53851e+08	1.2
mse	1.29276e+08	1.25685e+07	1.22953e+08	1.02176e+08	1.53851e+08	1.2
null_deviance	3.93265e+10	4.53807e+09	3.72839e+10	2.96943e+10	4.90307e+10	3.7
r2	0.110559	0.00589982	0.117462	0.114765	0.100491	0.1
residual_deviance	3.47427e+10	4.17128e+09	3.27055e+10	2.6157e+10	4.38475e+10	3.3
rmse	11342.6	557.598	11088.4	10108.2	12403.7	111
rmsle	0.945845	0.0343231	0.981383	0.914074	0.884154	1.0

Scoring History:

	timestamp	duration	iterations	negative_log_likelihood	objective
	2019-05-12 04:09:36	0.000 sec	0	1.96074e+11	1.46543e+08

4.4 Machine learning

4.4.1 Regularized regression

```
In [11]: 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_4_sid_b6da_model_python_1557652172496_1_model_5
1    [0.75]  Grid_GLM_py_4_sid_b6da_model_python_1557652172496_1_model_4
2     [0.5]  Grid_GLM_py_4_sid_b6da_model_python_1557652172496_1_model_3
3    [0.25]  Grid_GLM_py_4_sid_b6da_model_python_1557652172496_1_model_2
4     [0.0]  Grid_GLM_py_4_sid_b6da_model_python_1557652172496_1_model_1

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

```
In [12]: glm_best = glm_mod2.get_grid()[0]
         glm_best._model_json['output']['coefficients_table'].as_data_frame()
```

Out[12]:

	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 [13]: 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[13]:

```
In [14]: 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[14]:

4.4.2 Auto-ML

```
In [15]: 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[15]:

In [16]: `autom1_1.leader`

```
Model Details
=============
H2OXGBoostEstimator :  XGBoost
Model Key:  XGBoost_3_AutoML_20190512_040937


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

MSE: 12188391.73287312
RMSE: 3491.187725240956
MAE: 1819.1937206060816
RMSLE: 0.3021820933709707
Mean Residual Deviance: 12188391.73287312

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

MSE: 19380121.92359663
RMSE: 4402.285988392466
MAE: 2431.2812163864355
RMSLE: 0.40985324039544335
Mean Residual Deviance: 19380121.92359663

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

MSE: 19998514.578738146
RMSE: 4471.969876769984
MAE: 2408.4270772894056
RMSLE: 0.41859298826412966
Mean Residual Deviance: 19998514.578738146
Cross-Validation Metrics Summary:
```

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_
mae	2408.46	82.2232	2379.37	2247.67	2338.2	250
mean_residual_deviance	1.99967e+07	1.45611e+06	2.16905e+07	2.03256e+07	1.75627e+07	1.7
mse	1.99967e+07	1.45611e+06	2.16905e+07	2.03256e+07	1.75627e+07	1.7
r2	0.862954	0.0128701	0.876166	0.829337	0.880435	0.8
residual_deviance	1.99967e+07	1.45611e+06	2.16905e+07	2.03256e+07	1.75627e+07	1.7
rmse	4465.8	163.311	4657.31	4508.39	4190.79	42
rmsle	0.417898	0.0173632	0.393145	0.448762	0.416373	0.3

Scoring History:

	timestamp	duration	number_of_trees	training_rmse	training_mae	traini
	2019-05-12 04:09:49	1.424 sec	0	18177.7	13492.5	3.304
	2019-05-12 04:09:49	1.437 sec	5	14489.6	10517.1	2.099
	2019-05-12 04:09:49	1.445 sec	10	11629.2	8212.83	1.352
	2019-05-12 04:09:49	1.456 sec	15	9524.79	6461.63	9.072
	2019-05-12 04:09:49	1.465 sec	20	7934.06	5154.68	6.294
	2019-05-12 04:09:49	1.475 sec	25	6704.42	4165.21	4.494
	2019-05-12 04:09:49	1.487 sec	30	5943.49	3498.78	3.532
	2019-05-12 04:09:49	1.499 sec	35	5275.12	2966.53	2.782
	2019-05-12 04:09:49	1.512 sec	40	4770.26	2572.71	2.275
	2019-05-12 04:09:49	1.526 sec	45	4406.44	2309.07	1.941
	2019-05-12 04:09:49	1.542 sec	50	4171.37	2145.19	1.740
	2019-05-12 04:09:49	1.559 sec	55	3978.81	2038	1.583
	2019-05-12 04:09:49	1.577 sec	60	3815.2	1950.83	1.455
	2019-05-12 04:09:49	1.596 sec	65	3698.73	1890.03	1.368
	2019-05-12 04:09:50	1.617 sec	70	3620.38	1864.41	1.310
	2019-05-12 04:09:50	1.639 sec	75	3548.99	1837.89	1.259
	2019-05-12 04:09:50	1.663 sec	80	3501.17	1824.03	1.225
	2019-05-12 04:09:50	1.683 sec	81	3491.19	1819.19	1.218

Variable Importances:

variable	relative_importance	scaled_importance	percentage
smoker.no	4.10118e+11	1	0.499855
smoker.yes	1.4899e+11	0.363285	0.18159
bmi	1.35328e+11	0.329973	0.164939
age	1.02284e+11	0.2494	0.124664
children	1.0496e+10	0.0255926	0.0127926
region.northeast	4.24613e+09	0.0103534	0.00517522
sex.female	2.46772e+09	0.00601708	0.00300767
region.southeast	1.95259e+09	0.00476104	0.00237983

sex.male	1.7502e+09	0.00426755	0.00213316
region.southwest	1.43316e+09	0.00349449	0.00174674
region.northwest	1.40847e+09	0.00343429	0.00171665

Out[16]:

In [17]:
```
autom1_1.leader.cross_validation_metrics_summary()
```

Cross-Validation Metrics Summary:

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_
mae	2408.46	82.2232	2379.37	2247.67	2338.2	250
mean_residual_deviance	1.99967e+07	1.45611e+06	2.16905e+07	2.03256e+07	1.75627e+07	1.7
mse	1.99967e+07	1.45611e+06	2.16905e+07	2.03256e+07	1.75627e+07	1.7
r2	0.862954	0.0128701	0.876166	0.829337	0.880435	0.8
residual_deviance	1.99967e+07	1.45611e+06	2.16905e+07	2.03256e+07	1.75627e+07	1.7
rmse	4465.8	163.311	4657.31	4508.39	4190.79	42
rmsle	0.417898	0.0173632	0.393145	0.448762	0.416373	0.3

Out[17]:

4.5 Neural networks using kera and tensorflow

In [18]:
```
from keras import models
from keras import layers
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)
```

```
Using TensorFlow backend.
/Users/miaocai/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_split.py:2179: Fu
  FutureWarning)
WARNING:tensorflow:From /Users/miaocai/anaconda3/lib/python3.7/site-packages/tensorflow/python/
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /Users/miaocai/anaconda3/lib/python3.7/site-packages/tensorflow/python/
Instructions for updating:
Use tf.cast instead.
```

```
Out[18]: <keras.callbacks.History at 0x1a2d078f28>
```

```
In [19]: reg0.history.history['loss'][199]
```

```
Out[19]: 125999988.09172483
```

```
In [20]: reg0.history.history['val_loss'][199]
```

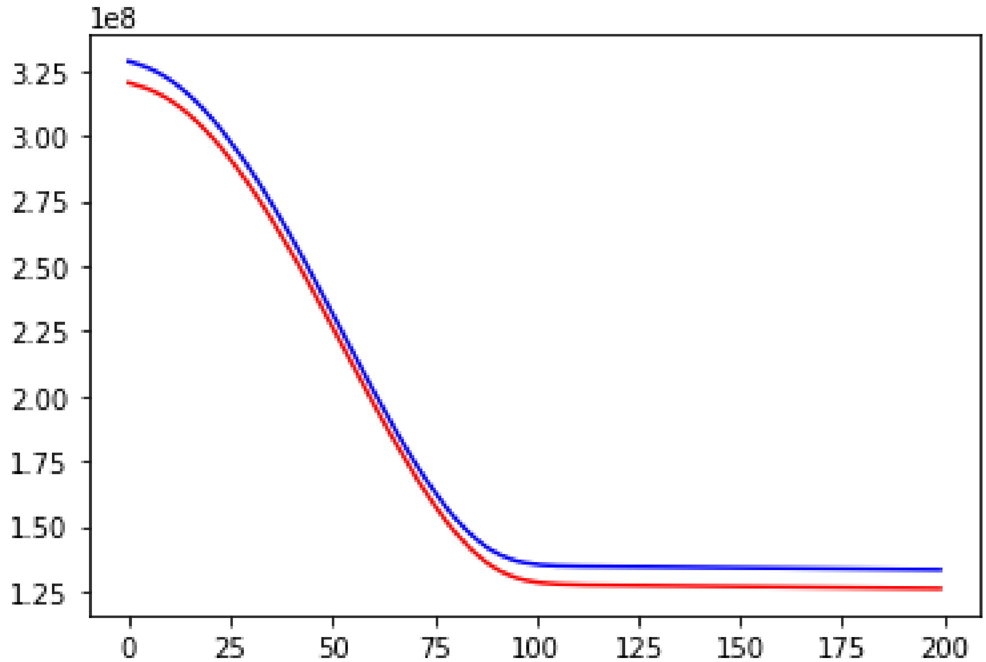
```
Out[20]: 133231693.99402985
```

```
In [21]: reg0.history.history['val_mean_absolute_error'][199]
```

```
Out[21]: 9078.39338123834
```

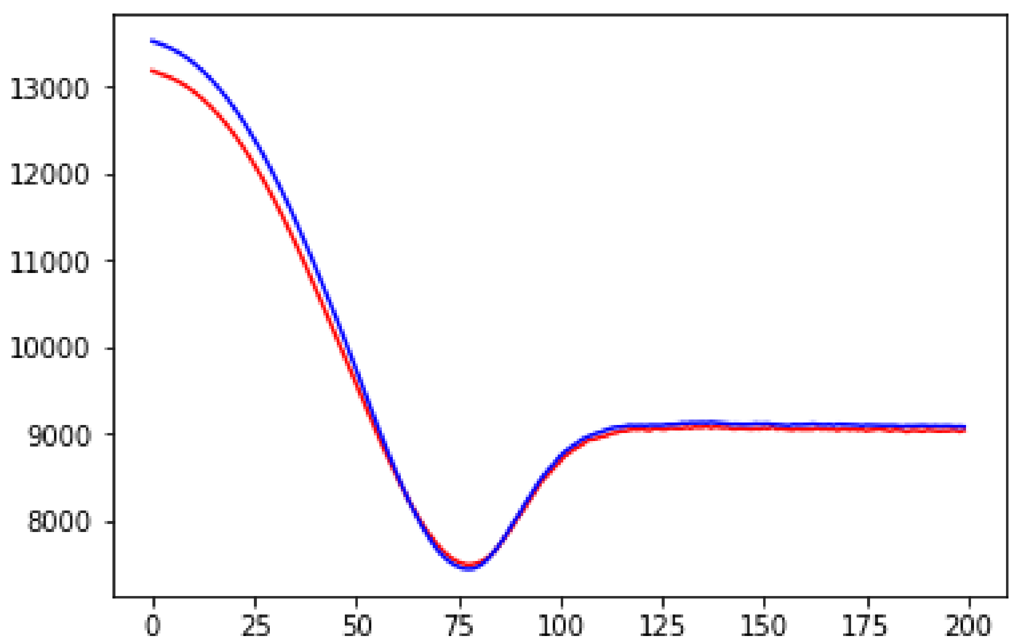
```
In [22]: plt.plot(reg0.history.history['loss'], color = 'red')
         plt.plot(reg0.history.history['val_loss'], color = 'blue')
```

```
Out[22]: [<matplotlib.lines.Line2D at 0x1a2d822ba8>]
```




```
In [23]: plt.plot(reg0.history.history['mean_absolute_error'], color = 'red')
         plt.plot(reg0.history.history['val_mean_absolute_error'], color = 'blue')
```

```
Out[23]: [<matplotlib.lines.Line2D at 0x1a2d7dbda0>]
```



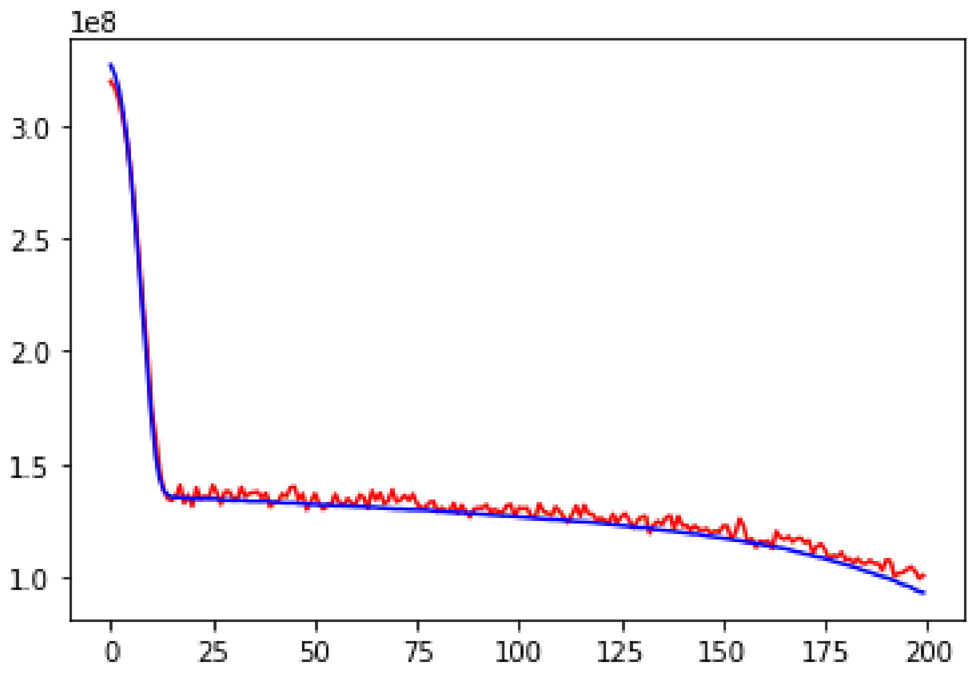
```
In [24]: def nn_model2():
             model = models.Sequential()
             model.add(layers.Dense(64, activation = 'relu',
                                    input_dim = X_train1.shape[1]))
             model.add(layers.Dropout(0.5))
             model.add(layers.Dense(64, activation = 'relu'))
             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)
```

```
WARNING:tensorflow:From /Users/miaocai/anaconda3/lib/python3.7/site-packages/keras/backend/tens
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

```
Out[24]: <keras.callbacks.History at 0x1a2d8c7048>
```

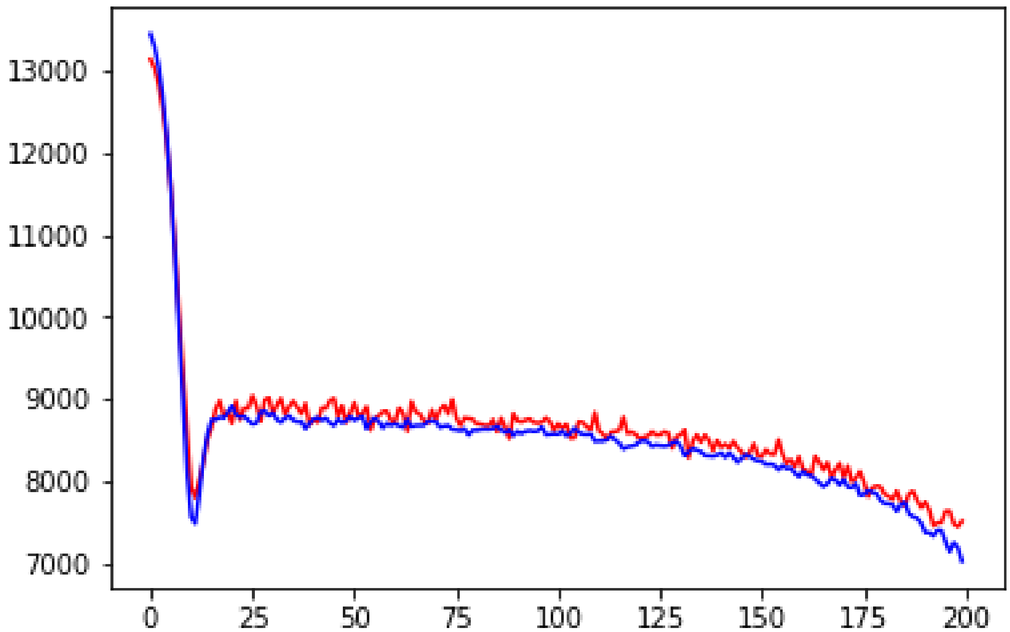
```
In [25]: plt.plot(reg2.history.history['loss'], color = 'red')
         plt.plot(reg2.history.history['val_loss'], color = 'blue')
```

```
Out[25]: [<matplotlib.lines.Line2D at 0x1a2dde9eb8>]
```



```
In [26]: plt.plot(reg2.history.history['mean_absolute_error'], color = 'red')
         plt.plot(reg2.history.history['val_mean_absolute_error'], color = 'blue')
```

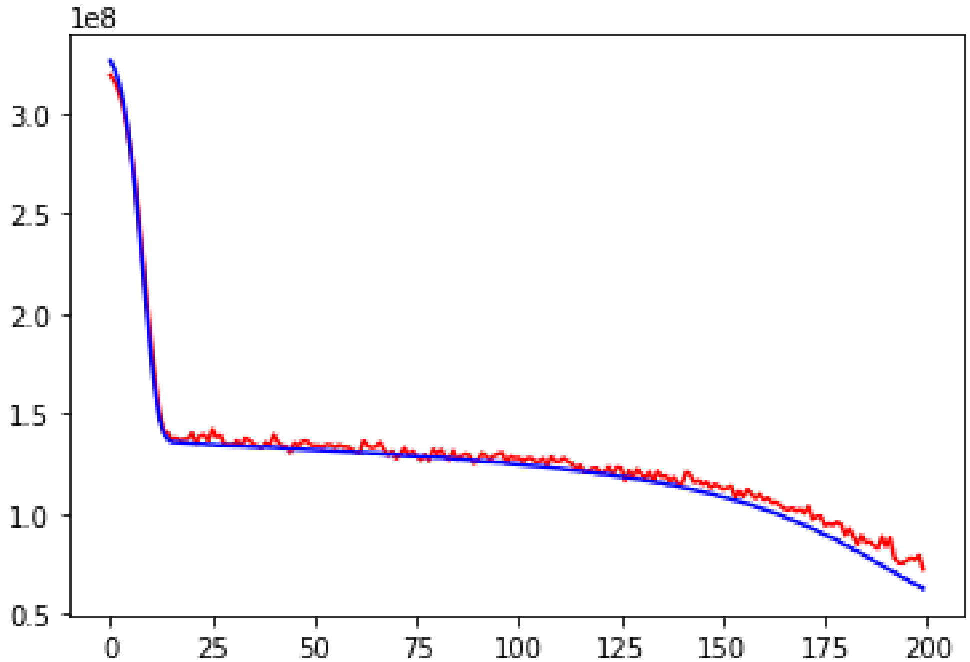
```
Out[26]: [<matplotlib.lines.Line2D at 0x1a2deb2550>]
```




```
In [27]: 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
```

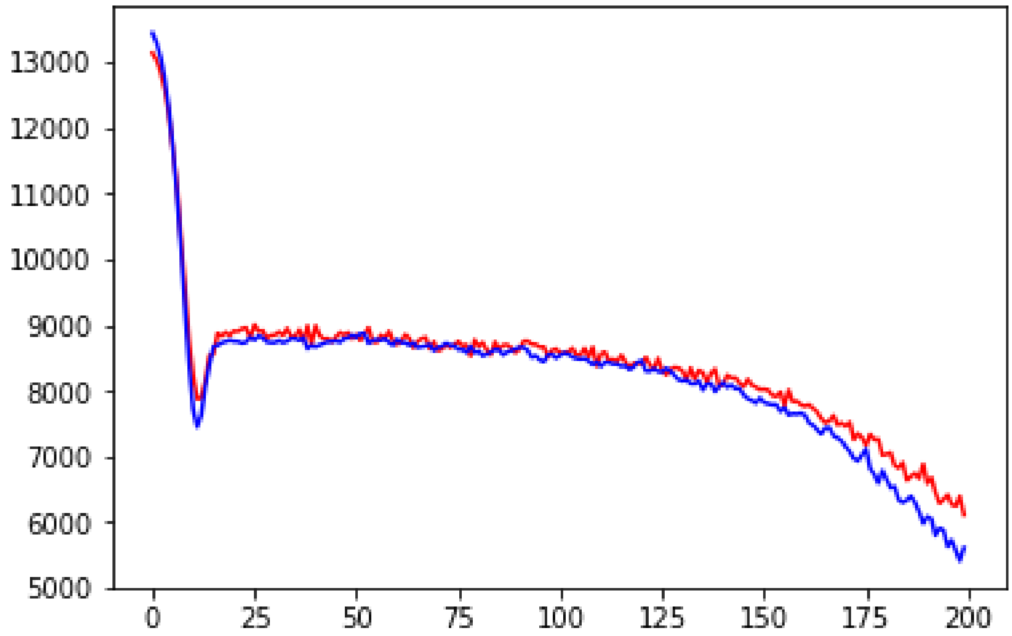
```
In [28]: reg3 = nn_model3(0.1)
         reg3.fit(X_train1, y_train1, epochs = 200,
                  validation_data = [X_valid1, y_valid1],
                 verbose = 0)
         plt.plot(reg3.history.history['loss'], color = 'red')
         plt.plot(reg3.history.history['val_loss'], color = 'blue')
```

```
Out[28]: [<matplotlib.lines.Line2D at 0x1a2e446208>]
```

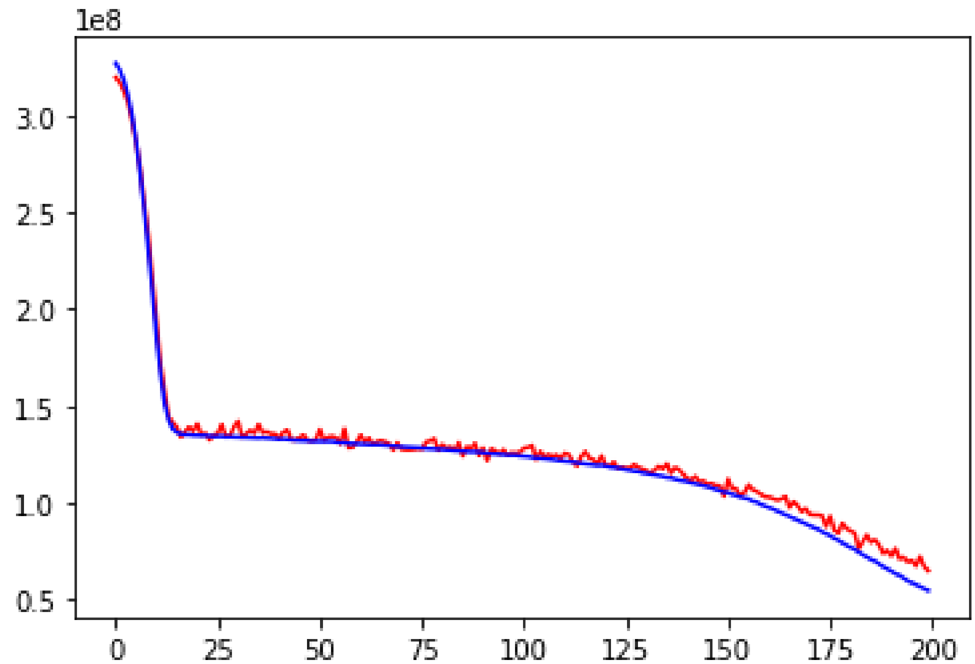



```
In [29]: plt.plot(reg3.history.history['mean_absolute_error'], color = 'red')
         plt.plot(reg3.history.history['val_mean_absolute_error'], color = 'blue')
```

```
Out[29]: [<matplotlib.lines.Line2D at 0x1a2e51b9e8>]
```

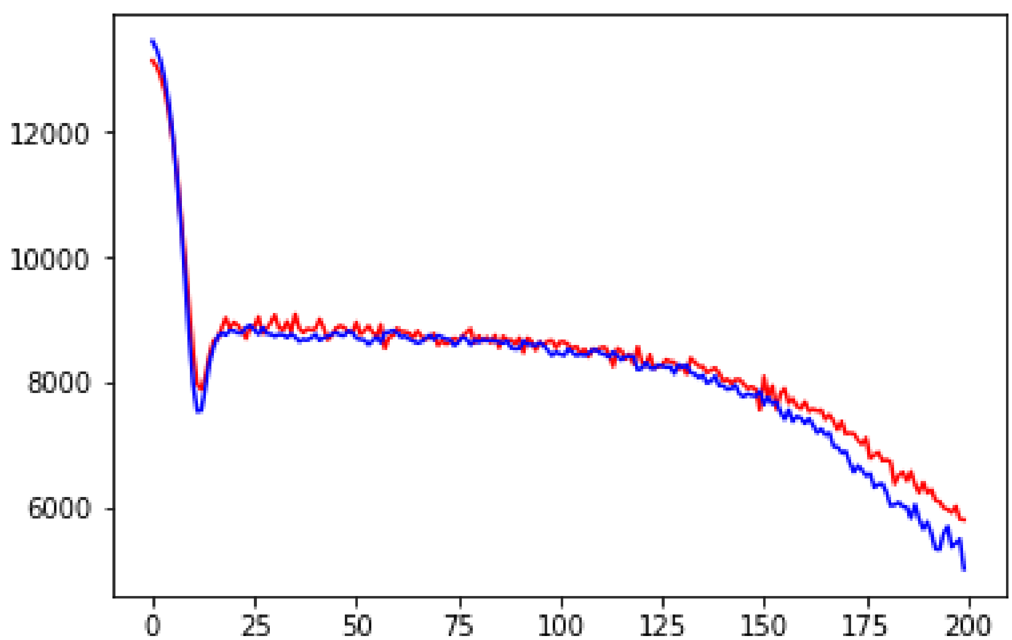



```
In [30]: reg3 = nn_model3(0.2)
         reg3.fit(X_train1, y_train1, epochs = 200,
                   validation_data = [X_valid1, y_valid1],
                  verbose = 0)
         plt.plot(reg3.history.history['loss'], color = 'red')
         plt.plot(reg3.history.history['val_loss'], color = 'blue')

Out[30]: [<matplotlib.lines.Line2D at 0x1a2eab65f8>]
```



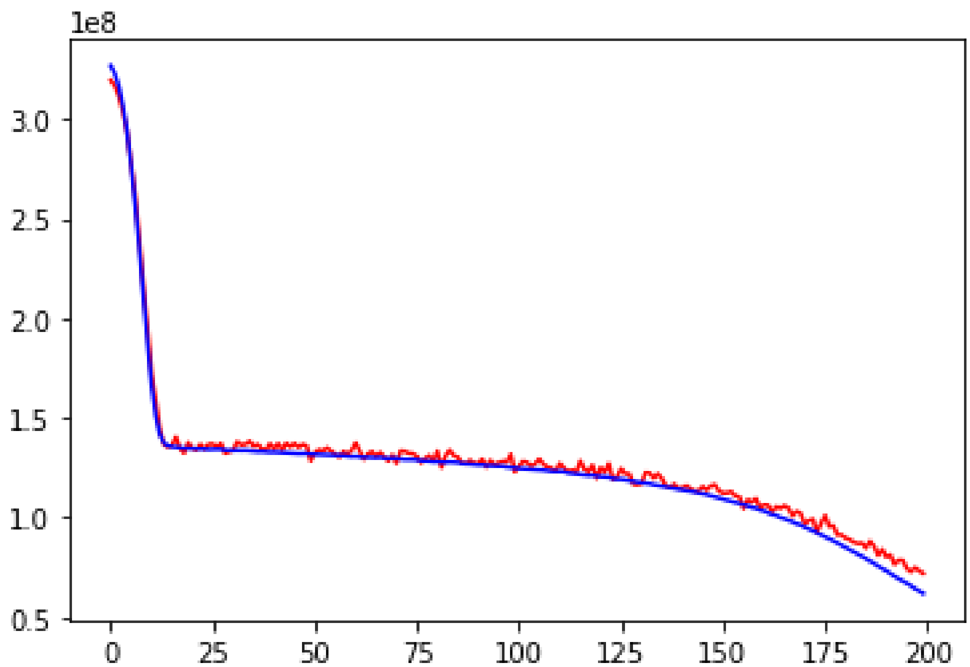

```
In [31]: plt.plot(reg3.history.history['mean_absolute_error'], color = 'red')
         plt.plot(reg3.history.history['val_mean_absolute_error'], color = 'blue')
```

```
Out[31]: [<matplotlib.lines.Line2D at 0x1a2eb15cf8>]
```



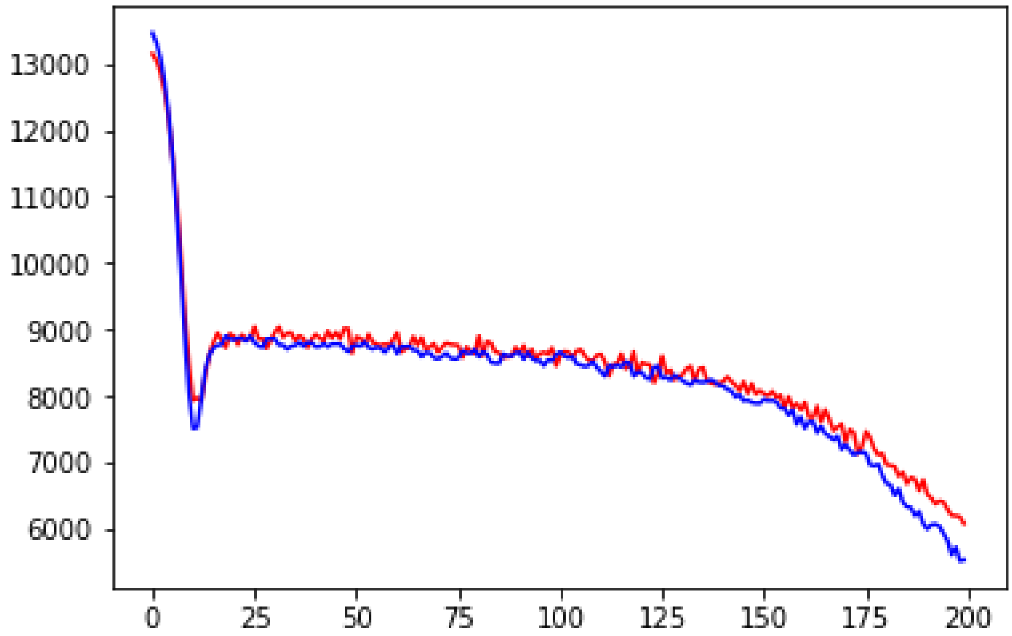
In [32]:
```
reg3 = nn_model3(0.3)
reg3.fit(X_train1, y_train1, epochs = 200,
          validation_data = [X_valid1, y_valid1],
         verbose = 0)
plt.plot(reg3.history.history['loss'], color = 'red')
plt.plot(reg3.history.history['val_loss'], color = 'blue')
```

Out[32]:
```
[<matplotlib.lines.Line2D at 0x1a2f0ebeb8>]
```



In [33]:
```
plt.plot(reg3.history.history['mean_absolute_error'], color = 'red')
plt.plot(reg3.history.history['val_mean_absolute_error'], color = 'blue')
```

Out[33]:
```
[<matplotlib.lines.Line2D at 0x1a2f156630>]
```



5 Conclusion

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```