all_state_capstone

March 18, 2018

0.0.1 Importing data

To get started two sets of data will be imported and will be analyzed to figure what will be the best way to preprocess them. After running the code below we can see that both data sets have category data that uses letters and continues data that uses numbers. The training data is comprised of 188,318 rows and 132 columns and the test data is comprised of 125,546 rows and 131 columns. The training data has an extra column for loss. This loss is what the insurance company pays out and is what we want to predict. For the purpose of this project we will be using this data set to train and test the model. The algorithm that found will be used to predict the loss value for the test_data set.

```
In [2]: import numpy as np
        import pandas as pd
        from IPython.display import display


        train_data = pd.read_csv("train.csv")
        test_data = pd.read_csv("test.csv")

        display(train_data.head())
        print train_data.shape

        display(test_data.head())
        print test_data.shape
```

```
   id cat1 cat2 cat3 cat4 cat5 cat6 cat7 cat8 cat9   ...        cont6  \
0   1    A    B    A    B    A    A    A    A    B   ...     0.718367
1   2    A    B    A    A    A    A    A    A    B   ...     0.438917
2   5    A    B    A    A    B    A    A    A    B   ...     0.289648
3  10    B    B    A    B    A    A    A    A    B   ...     0.440945
4  11    A    B    A    B    A    A    A    A    B   ...     0.178193

      cont7    cont8    cont9   cont10    cont11    cont12    cont13  \
0  0.335060  0.30260  0.67135  0.83510  0.569745  0.594646  0.822493
1  0.436585  0.60087  0.35127  0.43919  0.338312  0.366307  0.611431
2  0.315545  0.27320  0.26076  0.32446  0.381398  0.373424  0.195709
3  0.391128  0.31796  0.32128  0.44467  0.327915  0.321570  0.605077
4  0.247408  0.24564  0.22089  0.21230  0.204687  0.202213  0.246011
```

```
      cont14     loss
0  0.714843  2213.18
1  0.304496  1283.60
2  0.774425  3005.09
3  0.602642   939.85
4  0.432606  2763.85

[5 rows x 132 columns]
```

```
(188318, 132)
```

```
   id cat1 cat2 cat3 cat4 cat5 cat6 cat7 cat8 cat9    ...        cont5  \
0   4    A    B    A    A    A    A    A    A    B    ...     0.281143
1   6    A    B    A    B    A    A    A    A    B    ...     0.836443
2   9    A    B    A    B    B    A    B    A    B    ...     0.718531
3  12    A    A    A    A    B    A    A    A    A    ...     0.397069
4  15    B    A    A    A    A    B    A    A    A    ...     0.302678

      cont6     cont7    cont8    cont9   cont10    cont11    cont12  \
0  0.466591  0.317681  0.61229  0.34365  0.38016  0.377724  0.369858
1  0.482425  0.443760  0.71330  0.51890  0.60401  0.689039  0.675759
2  0.212308  0.325779  0.29758  0.34365  0.30529  0.245410  0.241676
3  0.369930  0.342355  0.40028  0.33237  0.31480  0.348867  0.341872
4  0.398862  0.391833  0.23688  0.43731  0.50556  0.359572  0.352251

     cont13    cont14
0  0.704052  0.392562
1  0.453468  0.208045
2  0.258586  0.297232
3  0.592264  0.555955
4  0.301535  0.825823

[5 rows x 131 columns]
```

```
(125546, 131)
```

In [3]:
```
train_data.skew()
```

Out[3]:
```
id        -0.002155
cont1      0.516424
cont2     -0.310941
cont3     -0.010002
cont4      0.416096
cont5      0.681622
```

```
cont6      0.461214
cont7      0.826053
cont8      0.676634
cont9      1.072429
cont10     0.355001
cont11     0.280821
cont12     0.291992
cont13     0.380742
cont14     0.248674
loss       3.794958
dtype: float64
```

0.0.2 Drop data not needed

After looking at the data there are a few changes that need to be made. First the column for "id" needs to be dropped because it does not add any value to our model. Second the "loss" column needs to be separated from the features.

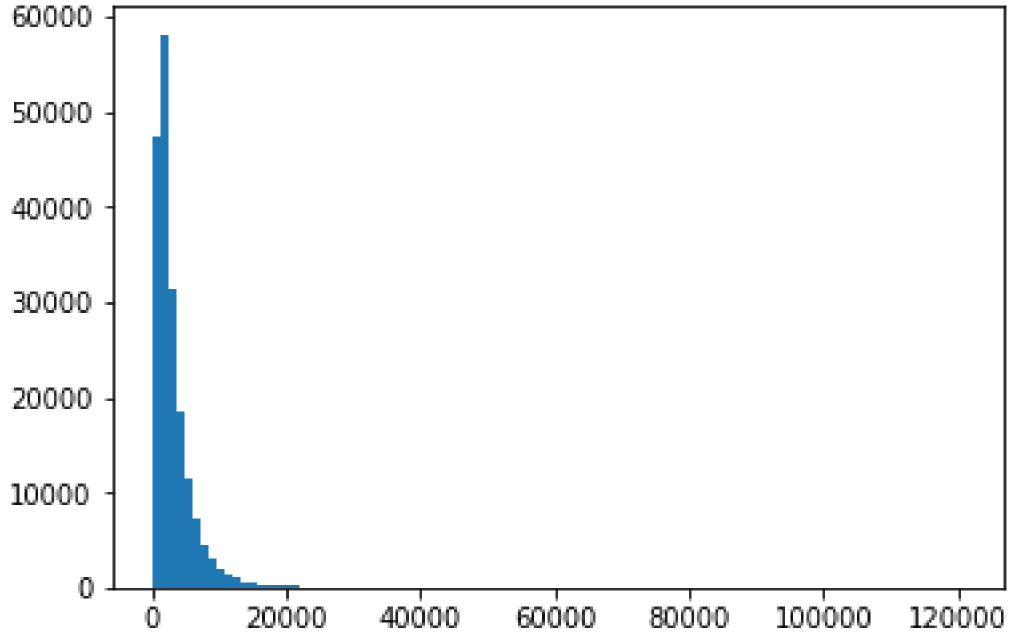
```
In [4]: train_columns = train_data.columns.values
        train_columns = pd.DataFrame(train_columns)

        loss = train_data["loss"]
        ids = train_data["id"]

        train_data_raw = train_data.drop(["id","loss"], axis = 1)


        test_data_id = test_data["id"]
        test_data_raw = test_data.drop(["id"], axis = 1)


        #make sure only ids and loss columsn were dropped
        print train_data_raw.shape
        print test_data_raw.shape
```

```
(188318, 130)
(125546, 130)
```

0.0.3 Loss data

The loss data is plotted on a histogram chart and we can see that the data is skewed to the right so it needs to be normalized. Regression algorithms can be sensitive to the distribution of values and can results in the model underperforming if the data is not normally distributed.

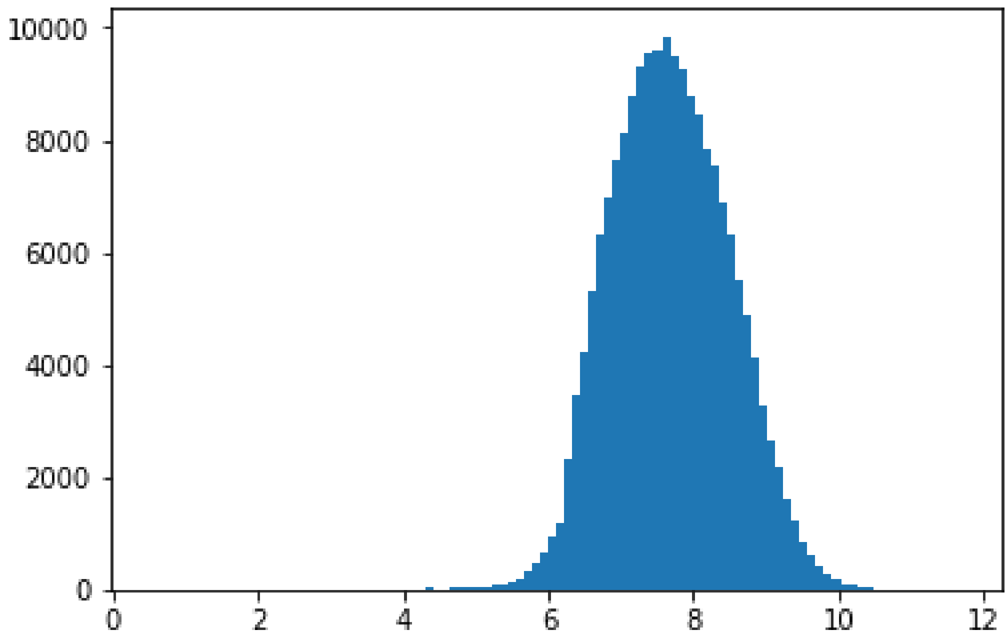
```
In [5]: import matplotlib.pyplot as pl
        pl.hist(loss,bins = 100)
        pl.show()
```



0.0.4 Normalizing the Data

Logarithmic transformation is applied to the "loss" data so that it does not negatively affect the performance of the learning algorithm. Apply the transformation will reduce the range of the values.

```
In [6]: loss_log_transformed = loss.apply(lambda x: np.log(x + 1))

        pl.hist(loss_log_transformed,bins = 100)
        pl.show()
```



0.0.5 Splitting the data

There are two different types of data in this data set, which are numeric and non-numeric. Below the numeric and non-numeric data will be separated.

```
In [7]: split = 116
        cont_data = train_data_raw.iloc[:,split:]
        display(cont_data.head())
        print cont_data.shape

        split = 116
        cont_test_data = test_data_raw.iloc[:,split:]
        display(cont_test_data.head())
        print cont_test_data.shape
```

```
      cont1     cont2     cont3     cont4     cont5     cont6     cont7  \
0  0.726300  0.245921  0.187583  0.789639  0.310061  0.718367  0.335060
1  0.330514  0.737068  0.592681  0.614134  0.885834  0.438917  0.436585
2  0.261841  0.358319  0.484196  0.236924  0.397069  0.289648  0.315545
3  0.321594  0.555782  0.527991  0.373816  0.422268  0.440945  0.391128
4  0.273204  0.159990  0.527991  0.473202  0.704268  0.178193  0.247408

     cont8    cont9   cont10    cont11    cont12    cont13    cont14
0  0.30260  0.67135  0.83510  0.569745  0.594646  0.822493  0.714843
1  0.60087  0.35127  0.43919  0.338312  0.366307  0.611431  0.304496
2  0.27320  0.26076  0.32446  0.381398  0.373424  0.195709  0.774425
```

```
3  0.31796  0.32128  0.44467  0.327915  0.321570  0.605077  0.602642
4  0.24564  0.22089  0.21230  0.204687  0.202213  0.246011  0.432606
```

```
(188318, 14)
```

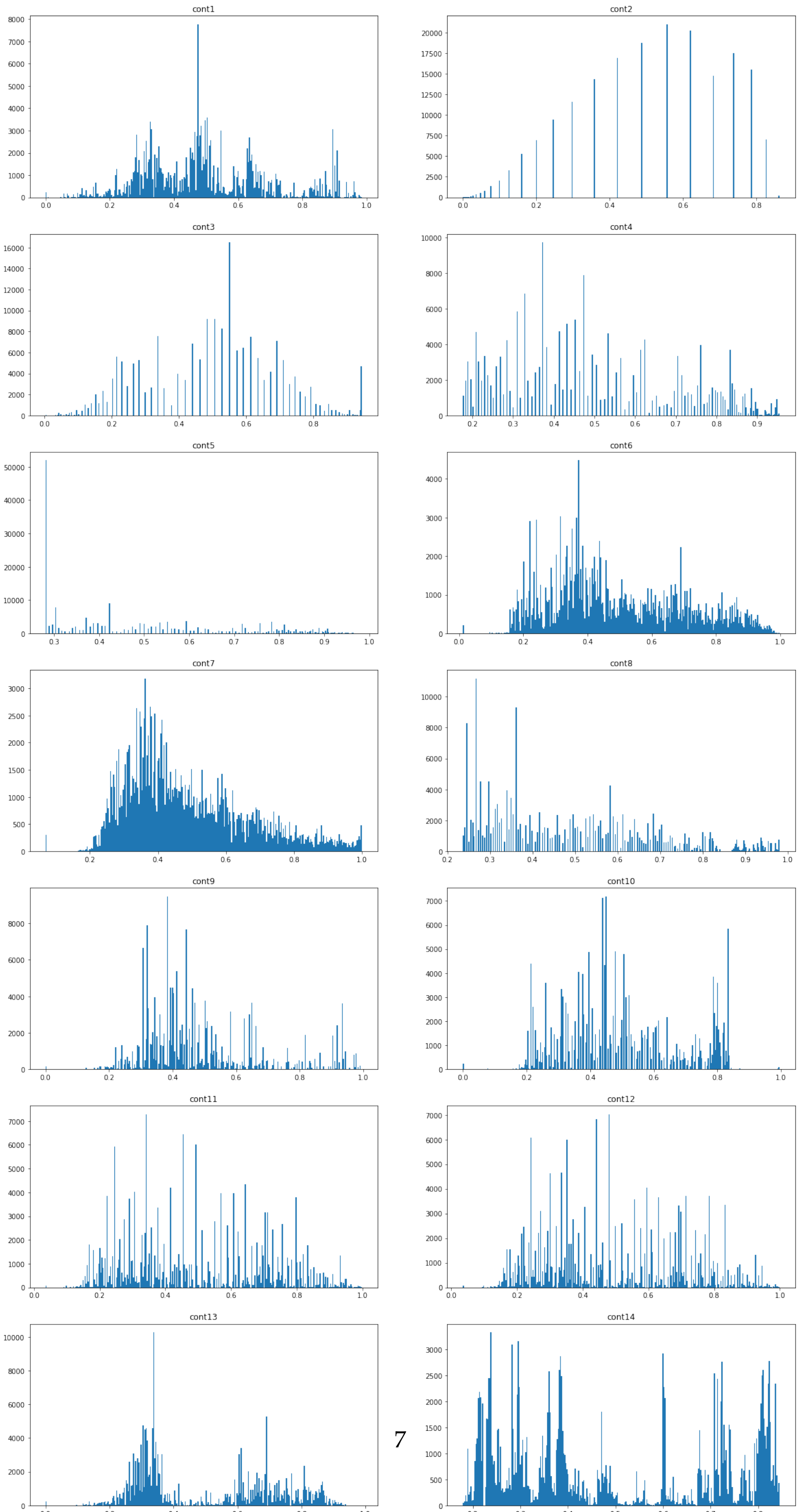
```
      cont1     cont2     cont3     cont4     cont5     cont6     cont7  \
0  0.321594  0.299102  0.246911  0.402922  0.281143  0.466591  0.317681
1  0.634734  0.620805  0.654310  0.946616  0.836443  0.482425  0.443760
2  0.290813  0.737068  0.711159  0.412789  0.718531  0.212308  0.325779
3  0.268622  0.681761  0.592681  0.354893  0.397069  0.369930  0.342355
4  0.553846  0.299102  0.263570  0.696873  0.302678  0.398862  0.391833

     cont8    cont9   cont10    cont11    cont12    cont13    cont14
0  0.61229  0.34365  0.38016  0.377724  0.369858  0.704052  0.392562
1  0.71330  0.51890  0.60401  0.689039  0.675759  0.453468  0.208045
2  0.29758  0.34365  0.30529  0.245410  0.241676  0.258586  0.297232
3  0.40028  0.33237  0.31480  0.348867  0.341872  0.592264  0.555955
4  0.23688  0.43731  0.50556  0.359572  0.352251  0.301535  0.825823
```

```
(125546, 14)
```

0.0.6 Numeric Data

Histograms are plotted to what is going on within the continuous data. None of the categories seem to have a normal distribution

```
In [8]: #plot continues data

        cont_data_columns = cont_data.columns.values
        fig, axarr = pl.subplots(nrows=7, ncols=2,figsize=(20, 40))

        #columns
        index_arr = -1
        for i in range(7):
            for j in range(2):
                index_arr = index_arr + 1
                axarr[i, j].hist(cont_data[cont_data_columns[index_arr]], bins = 300)
                axarr[i, j].set_title(cont_data_columns[index_arr])


        pl.show()
```


cont1
cont2
cont3
cont4
cont5
cont6
cont7
cont8
cont9
cont10
cont11
cont12
cont13
cont14

0.0.7 Hot key encoding

Typically learning algorithms expect input to be numeric so all non-numeric data must be converted into numeric data. There are many ways to do so but for this project we will be using one-hot encoding. This method creates dummy variables for each possible category of each non-numeric feature.

```
In [9]: cat_data = train_data_raw.iloc[:,:split]
        display(cat_data.head())
        print cat_data.shape

        cat_test_data = test_data_raw.iloc[:,:split]
        display(cat_test_data.head())
        print cat_test_data.shape
```

```
  cat1 cat2 cat3 cat4 cat5 cat6 cat7 cat8 cat9 cat10  ...   cat107 cat108  \
0    A    B    A    B    A    A    A    A    B     A  ...        J      G
1    A    B    A    A    A    A    A    A    B     B  ...        K      K
2    A    B    A    A    B    A    A    A    B     B  ...        F      A
3    B    B    A    B    A    A    A    A    B     A  ...        K      K
4    A    B    A    B    A    A    A    A    B     B  ...        G      B

  cat109 cat110 cat111 cat112 cat113 cat114 cat115 cat116
0     BU     BC      C     AS      S      A      O     LB
1     BI     CQ      A     AV     BM      A      O     DP
2     AB     DK      A      C     AF      A      I     GK
3     BI     CS      C      N     AE      A      O     DJ
4      H      C      C      Y     BM      A      K     CK

[5 rows x 116 columns]


(188318, 116)


  cat1 cat2 cat3 cat4 cat5 cat6 cat7 cat8 cat9 cat10  ...   cat107 cat108  \
0    A    B    A    A    A    A    A    A    B     A  ...        L      K
1    A    B    A    B    A    A    A    A    B     A  ...        F      B
2    A    B    A    B    B    A    B    A    B     B  ...        G      A
3    A    A    A    A    B    A    A    A    A     A  ...        K      K
4    B    A    A    A    A    B    A    A    A     A  ...        E      B

  cat109 cat110 cat111 cat112 cat113 cat114 cat115 cat116
0     BI     BC      A      J     AX      A      Q     HG
1     BI     CO      E      G      X      A      L     HK
2     BI     CS      C      U     AE      A      K     CK
```

```
3     BI     CR      A     AY     AJ      A      P     DJ
4     AB     EG      A      E      I      C      J     HA

[5 rows x 116 columns]
```

```
(125546, 116)
```

In [10]:
```
len(np.unique(cat_data['cat1']))
unique_var = []
for column in cat_data.columns.values:
    variables = len(np.unique(cat_data[column]))
    unique_var.append(variables)

#    print "{} unique lables in {}".format(variables,column)
```

In [11]:
```
unique_var2 =  list(set(unique_var))
sorted_var = np.sort(unique_var2)
for uniquer in sorted_var:
    n = unique_var.count(uniquer)
    print "There is {} unique categories in {} of the lables".format(uniquer,n)
```

```
There is 2 unique categories in 72 of the lables
There is 3 unique categories in 4 of the lables
There is 4 unique categories in 12 of the lables
There is 5 unique categories in 3 of the lables
There is 7 unique categories in 4 of the lables
There is 8 unique categories in 3 of the lables
There is 9 unique categories in 1 of the lables
There is 11 unique categories in 1 of the lables
There is 13 unique categories in 1 of the lables
There is 15 unique categories in 1 of the lables
There is 16 unique categories in 2 of the lables
There is 17 unique categories in 2 of the lables
There is 19 unique categories in 2 of the lables
There is 20 unique categories in 2 of the lables
There is 23 unique categories in 1 of the lables
There is 51 unique categories in 1 of the lables
There is 61 unique categories in 1 of the lables
There is 84 unique categories in 1 of the lables
There is 131 unique categories in 1 of the lables
There is 326 unique categories in 1 of the lables
```

In [13]:
```
cat_data_df = cat_data.apply(lambda x: pd.factorize(x)[0])

cat_test_data_df = cat_test_data.apply(lambda x: pd.factorize(x)[0])

display(cat_data_df.head())
```

```
   cat1  cat2  cat3  cat4  cat5  cat6  cat7  cat8  cat9  cat10   ...    \
0     0     0     0     0     0     0     0     0     0      0   ...
1     0     0     0     1     0     0     0     0     0      1   ...
2     0     0     0     1     1     0     0     0     0      1   ...
3     1     0     0     0     0     0     0     0     0      0   ...
4     0     0     0     0     0     0     0     0     0      1   ...

   cat107  cat108  cat109  cat110  cat111  cat112  cat113  cat114  cat115  \
0       0       0       0       0       0       0       0       0       0
1       1       1       1       1       1       1       1       0       0
2       2       2       2       2       1       2       2       0       1
3       1       1       1       3       0       3       3       0       0
4       3       3       3       4       0       4       1       0       2

   cat116
0       0
1       1
2       2
3       3
4       4

[5 rows x 116 columns]
```

0.0.8 Shuffle and split data

Now that the data has been processed the training data will be split into testing and training in which 80% is used for training and 20% is used for testing.

```
In [14]: cat_cont_data = pd.concat([cat_data_df, cont_data], axis=1)
         display(cat_cont_data.head())


         cat_cont_test_data = pd.concat([cat_test_data_df, cont_test_data], axis=1)
         display(cat_cont_test_data.head())
```

```
   cat1  cat2  cat3  cat4  cat5  cat6  cat7  cat8  cat9  cat10    ...     \
0     0     0     0     0     0     0     0     0     0      0    ...
1     0     0     0     1     0     0     0     0     0      1    ...
2     0     0     0     1     1     0     0     0     0      1    ...
3     1     0     0     0     0     0     0     0     0      0    ...
4     0     0     0     0     0     0     0     0     0      1    ...

      cont5     cont6     cont7    cont8    cont9   cont10    cont11  \
0  0.310061  0.718367  0.335060  0.30260  0.67135  0.83510  0.569745
1  0.885834  0.438917  0.436585  0.60087  0.35127  0.43919  0.338312
2  0.397069  0.289648  0.315545  0.27320  0.26076  0.32446  0.381398
3  0.422268  0.440945  0.391128  0.31796  0.32128  0.44467  0.327915
```

```
4  0.704268  0.178193  0.247408  0.24564  0.22089  0.21230  0.204687

     cont12    cont13    cont14
0  0.594646  0.822493  0.714843
1  0.366307  0.611431  0.304496
2  0.373424  0.195709  0.774425
3  0.321570  0.605077  0.602642
4  0.202213  0.246011  0.432606

[5 rows x 130 columns]


   cat1  cat2  cat3  cat4  cat5  cat6  cat7  cat8  cat9  cat10    ...     \
0     0     0     0     0     0     0     0     0     0      0    ...
1     0     0     0     1     0     0     0     0     0      0    ...
2     0     0     0     1     1     0     1     0     0      1    ...
3     0     1     0     0     1     0     0     0     1      0    ...
4     1     1     0     0     0     1     0     0     1      0    ...

      cont5     cont6     cont7    cont8    cont9   cont10    cont11  \
0  0.281143  0.466591  0.317681  0.61229  0.34365  0.38016  0.377724
1  0.836443  0.482425  0.443760  0.71330  0.51890  0.60401  0.689039
2  0.718531  0.212308  0.325779  0.29758  0.34365  0.30529  0.245410
3  0.397069  0.369930  0.342355  0.40028  0.33237  0.31480  0.348867
4  0.302678  0.398862  0.391833  0.23688  0.43731  0.50556  0.359572

     cont12    cont13    cont14
0  0.369858  0.704052  0.392562
1  0.675759  0.453468  0.208045
2  0.241676  0.258586  0.297232
3  0.341872  0.592264  0.555955
4  0.352251  0.301535  0.825823

[5 rows x 130 columns]
```

```
In [19]: from sklearn.cross_validation import train_test_split

         X_train, X_test, y_train, y_test = train_test_split(cat_cont_data, loss_log_transformed

         print "Training set has {} samples.".format(X_train.shape[0])
         print "Testing set has {} samples.".format(X_test.shape[0])
```

```
Training set has 141238 samples.
Testing set has 47080 samples.
```

The code block below will allow for automatically testing different repressors using different data sizes.

In [16]:
```
from sklearn.metrics import mean_absolute_error
from time import time

def train_predict(learner, sample_size, X_train, y_train, X_test, y_test):
    results = {}
    start = time()
    learner = learner.fit(X_train[:sample_size], y_train[:sample_size])
    end = time()

    results['train_time'] = end - start

    start = time()
    predictions_test = learner.predict(X_test)
    predictions_train = learner.predict(X_train[:300])
    end = time()

    results['pred_time'] = end - start

    results['acc_train'] = mean_absolute_error(np.exp(y_train[:300]), np.exp(prediction

    results['acc_test'] = mean_absolute_error(np.exp(y_test), np.exp(predictions_test))

    print "{} trained on {} samples.".format(learner.__class__.__name__, sample_size)
    print "Trained score = {:.2f} and test score = {:.2f}".format(results['acc_train'],
    print "Trained time = {:.2f} and pred time = {:.2f}".format(results['train_time'],r

    return results
```

0.0.9 Learning Algorithms

Three different repressors will be taken into consideration linear regression, random forest regression, and extreme gradient regression. The mean absolute error of a un-tuned Linear regression model will be used as our bench mark that we need to beat. Mean absolute error is a measure of the average difference between the actual loss and predicted loss.

In [18]:
```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor



clf_A = LinearRegression()
#clf_B = RandomForestRegressor(random_state = 10)
clf_C = XGBRegressor(random_state = 10)

sample_1 = len(X_train) / 100
sample_10 = len(X_train) / 10
```

```
sample_100 = len(X_train) / 1

results = {}

for clf in [clf_A,clf_C]:
    clf_name = clf.__class__.__name__
    results[clf_name] = {}
    for i, samples in enumerate([sample_1, sample_10, sample_100]):
        results[clf_name][i] = train_predict(clf, samples, X_train, y_train, X_test, y_
```

```
LinearRegression trained on 1412 samples.
Trained score = 1262.83 and test score = 30697.67
Trained time = 0.02 and pred time = 0.04
LinearRegression trained on 14123 samples.
Trained score = 1287.68 and test score = 1431.73
Trained time = 0.09 and pred time = 0.02
LinearRegression trained on 141238 samples.
Trained score = 1303.10 and test score = 1334.36
Trained time = 1.21 and pred time = 0.02
XGBRegressor trained on 1412 samples.
Trained score = 1068.56 and test score = 1285.11
Trained time = 0.67 and pred time = 0.23
XGBRegressor trained on 14123 samples.
Trained score = 1224.03 and test score = 1232.22
Trained time = 7.07 and pred time = 0.23
XGBRegressor trained on 141238 samples.
Trained score = 1231.10 and test score = 1217.24
Trained time = 74.02 and pred time = 0.22
```

0.0.10 XGBoosting

XGboosting stands for extreme gradient boosting algorithm and is part of a group of learning algorithms called ensemble methods. There are three types of ensemble methods and XGBoosting is a part of the boosting class of ensemble methods. XGBoosting runs multiple decision tree algorithms and each of which learns to fix the prediction errors of a prior model in the chain.

In other words, this can be compared to how a basketball player shoots a basketball. If the basketball player misses the shot then the next shot the basketball player takes is corrected based on how far off his first shot was from the goal. The basketball player will continue to adjust this technique until he makes perfect shots.

```
In [21]: import xgboost as xgb

         dtrain = xgb.DMatrix(data=X_train, label=y_train)
         dtest = xgb.DMatrix(data=X_test, label=y_test)
```

```
In [25]: from sklearn.metrics import mean_absolute_error
         from time import time
```

```
from sklearn.metrics import make_scorer


xgb_params = {
    'objective': 'reg:linear',
}

scorer = make_scorer(mean_absolute_error)


results = {}
start = time()
bst_cv1 = xgb.train(params=xgb_params, dtrain=dtrain, num_boost_round=50,
                                  feval=scorer, maximize=False)
end = time()
results['train_time'] = end - start

start = time()
dpred_train = xgb.DMatrix(X_train)
dpred_test = xgb.DMatrix(X_test)

predictions_train = bst_cv1.predict(dtrain)
predictions_test = bst_cv1.predict(dtest)
end = time()
results['pred_time'] = end - start

y_tr = dtrain.get_label()
y_te = dtest.get_label()


results['acc_train'] = mean_absolute_error(np.exp(y_tr), np.exp(predictions_train))

results['acc_test'] = mean_absolute_error(np.exp(y_te), np.exp(predictions_test))
print "params {}".format(xgb_params)
print "Trained score = {:.2f} and test score = {:.2f}".format(results['acc_train'],resu
print "Trained time = {:.2f} and pred time = {:.2f}".format(results['train_time'],resul
```

```
params {'objective': 'reg:linear'}
Trained score = 1108.09 and test score = 1178.91
Trained time = 77.09 and pred time = 0.83
```

In [ ]:
```
from sklearn.metrics import mean_absolute_error
from time import time
from sklearn.metrics import make_scorer


xgb_params = {'eta': 0.3,
```

```
            'objective': 'reg:linear',
        }

        scorer = make_scorer(mean_absolute_error)

        max_depth = [6,7,8,9,10,11,12,13,14]
        child_weight = [4,6,8,10,12,14,16,18]

        for depth in max_depth:
            xgb_params['max_depth'] = depth
            for weight in child_weight:
                xgb_params['min_child_weight'] = weight
                results = {}
                start = time()
                bst_cv1 = xgb.train(params=xgb_params, dtrain=dtrain, num_boost_round=50,
                                    feval=scorer, maximize=False)
                end = time()
                results['train_time'] = end - start

                start = time()

                predictions_train = bst_cv1.predict(dtrain)
                predictions_test = bst_cv1.predict(dtest)
                end = time()
                results['pred_time'] = end - start

                y_tr = dtrain.get_label()
                y_te = dtest.get_label()


                results['acc_train'] = mean_absolute_error(np.exp(y_tr), np.exp(predictions_trai

                results['acc_test'] = mean_absolute_error(np.exp(y_te), np.exp(predictions_test)
                print "params {}".format(xgb_params)
                print "Trained score = {:.2f} and test score = {:.2f}".format(results['acc_train
                print "Trained time = {:.2f} and pred time = {:.2f}".format(results['train_time'
```

```
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 6, 'min_child_weight': 4}
Trained score = 1109.09 and test score = 1180.59
Trained time = 66.64 and pred time = 0.15
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 6, 'min_child_weight': 6}
Trained score = 1111.30 and test score = 1178.77
Trained time = 67.21 and pred time = 0.14
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 6, 'min_child_weight': 8}
Trained score = 1112.07 and test score = 1179.64
Trained time = 67.33 and pred time = 0.14
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 6, 'min_child_weight': 10}
Trained score = 1113.95 and test score = 1178.14
```

```
Trained time = 65.62 and pred time = 0.15
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 6, 'min_child_weight': 12}
Trained score = 1112.87 and test score = 1179.54
Trained time = 65.12 and pred time = 0.14
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 6, 'min_child_weight': 14}
Trained score = 1115.27 and test score = 1179.16
Trained time = 66.93 and pred time = 0.15
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 6, 'min_child_weight': 16}
Trained score = 1116.16 and test score = 1179.59
Trained time = 73.39 and pred time = 0.18
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 6, 'min_child_weight': 18}
Trained score = 1115.03 and test score = 1177.02
Trained time = 71.91 and pred time = 0.14
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 7, 'min_child_weight': 4}
Trained score = 1074.08 and test score = 1182.92
Trained time = 85.22 and pred time = 0.19
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 7, 'min_child_weight': 6}
Trained score = 1074.80 and test score = 1179.67
Trained time = 84.00 and pred time = 0.18
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 7, 'min_child_weight': 8}
Trained score = 1075.40 and test score = 1182.12
Trained time = 81.37 and pred time = 0.18
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 7, 'min_child_weight': 10}
Trained score = 1077.83 and test score = 1179.44
Trained time = 82.62 and pred time = 0.18
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 7, 'min_child_weight': 12}
Trained score = 1080.10 and test score = 1180.68
Trained time = 79.77 and pred time = 0.19
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 7, 'min_child_weight': 14}
Trained score = 1081.98 and test score = 1180.75
Trained time = 78.15 and pred time = 0.18
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 7, 'min_child_weight': 16}
Trained score = 1085.59 and test score = 1178.27
Trained time = 79.59 and pred time = 0.18
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 7, 'min_child_weight': 18}
Trained score = 1083.79 and test score = 1176.63
Trained time = 83.75 and pred time = 0.18
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 8, 'min_child_weight': 4}
Trained score = 1024.65 and test score = 1182.63
Trained time = 98.49 and pred time = 0.28
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 8, 'min_child_weight': 6}
Trained score = 1028.98 and test score = 1180.55
Trained time = 104.54 and pred time = 0.27
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 8, 'min_child_weight': 8}
Trained score = 1034.02 and test score = 1180.94
Trained time = 96.81 and pred time = 0.20
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 8, 'min_child_weight': 10}
Trained score = 1038.61 and test score = 1181.06
```

```
Trained time = 93.87 and pred time = 0.21
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 8, 'min_child_weight': 12}
Trained score = 1035.71 and test score = 1177.96
Trained time = 90.22 and pred time = 0.22
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 8, 'min_child_weight': 14}
Trained score = 1042.64 and test score = 1181.10
Trained time = 93.02 and pred time = 0.20
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 8, 'min_child_weight': 16}
Trained score = 1045.56 and test score = 1179.87
Trained time = 97.95 and pred time = 0.21
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 8, 'min_child_weight': 18}
Trained score = 1047.58 and test score = 1178.41
Trained time = 96.96 and pred time = 0.20
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 9, 'min_child_weight': 4}
Trained score = 974.02 and test score = 1187.75
Trained time = 104.32 and pred time = 0.26
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 9, 'min_child_weight': 6}
Trained score = 974.67 and test score = 1191.65
Trained time = 104.19 and pred time = 0.26
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 9, 'min_child_weight': 8}
Trained score = 991.17 and test score = 1187.21
Trained time = 103.88 and pred time = 0.23
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 9, 'min_child_weight': 10}
Trained score = 987.48 and test score = 1184.16
Trained time = 102.56 and pred time = 0.24
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 9, 'min_child_weight': 12}
Trained score = 997.49 and test score = 1184.23
Trained time = 104.79 and pred time = 0.31
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 9, 'min_child_weight': 14}
Trained score = 1001.27 and test score = 1184.01
Trained time = 107.58 and pred time = 0.24
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 9, 'min_child_weight': 16}
Trained score = 1003.30 and test score = 1185.08
Trained time = 100.54 and pred time = 0.24
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 9, 'min_child_weight': 18}
Trained score = 1011.20 and test score = 1185.66
Trained time = 100.13 and pred time = 0.23
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 10, 'min_child_weight': 4}
Trained score = 899.27 and test score = 1198.59
Trained time = 113.89 and pred time = 0.28
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 10, 'min_child_weight': 6}
Trained score = 914.06 and test score = 1199.04
Trained time = 113.29 and pred time = 0.29
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 10, 'min_child_weight': 8}
Trained score = 921.31 and test score = 1198.36
Trained time = 112.91 and pred time = 0.29
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 10, 'min_child_weight': 10}
Trained score = 933.02 and test score = 1194.58
```

```
Trained time = 113.12 and pred time = 0.28
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 10, 'min_child_weight': 12}
Trained score = 945.81 and test score = 1193.67
Trained time = 111.72 and pred time = 0.28
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 10, 'min_child_weight': 14}
Trained score = 948.08 and test score = 1190.29
Trained time = 111.49 and pred time = 0.28
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 10, 'min_child_weight': 16}
Trained score = 956.78 and test score = 1191.54
Trained time = 111.50 and pred time = 0.28
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 10, 'min_child_weight': 18}
Trained score = 958.58 and test score = 1189.86
Trained time = 112.42 and pred time = 0.27
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 11, 'min_child_weight': 4}
Trained score = 830.71 and test score = 1206.71
Trained time = 125.45 and pred time = 0.32
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 11, 'min_child_weight': 6}
Trained score = 845.23 and test score = 1207.85
Trained time = 123.29 and pred time = 0.32
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 11, 'min_child_weight': 8}
Trained score = 854.73 and test score = 1206.09
Trained time = 126.71 and pred time = 0.39
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 11, 'min_child_weight': 10}
Trained score = 874.33 and test score = 1202.50
Trained time = 129.33 and pred time = 0.36
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 11, 'min_child_weight': 12}
Trained score = 892.12 and test score = 1195.82
Trained time = 131.62 and pred time = 0.32
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 11, 'min_child_weight': 14}
Trained score = 900.89 and test score = 1199.71
Trained time = 128.31 and pred time = 0.33
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 11, 'min_child_weight': 16}
Trained score = 908.66 and test score = 1202.90
Trained time = 128.39 and pred time = 0.32
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 11, 'min_child_weight': 18}
Trained score = 914.15 and test score = 1201.85
Trained time = 129.41 and pred time = 0.31
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 12, 'min_child_weight': 4}
Trained score = 723.34 and test score = 1214.61
Trained time = 146.10 and pred time = 0.41
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 12, 'min_child_weight': 6}
Trained score = 778.83 and test score = 1211.81
Trained time = 159.12 and pred time = 0.52
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 12, 'min_child_weight': 8}
Trained score = 801.02 and test score = 1208.18
Trained time = 150.85 and pred time = 0.42
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 12, 'min_child_weight': 10}
Trained score = 804.45 and test score = 1213.53
```

```
Trained time = 150.46 and pred time = 0.37
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 12, 'min_child_weight': 12}
Trained score = 837.53 and test score = 1212.79
Trained time = 138.25 and pred time = 0.37
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 12, 'min_child_weight': 14}
Trained score = 844.99 and test score = 1202.53
Trained time = 140.91 and pred time = 0.38
params {'objective': 'reg:linear', 'eta': 0.3, 'max_depth': 12, 'min_child_weight': 16}
Trained score = 847.52 and test score = 1206.24
Trained time = 154.09 and pred time = 0.40
```

In [18]:
```python
from sklearn.metrics import mean_absolute_error
from time import time
from sklearn.metrics import make_scorer


xgb_params = {'seed': 0, 'eta': 0.1, 'colsample_bytree': 0.5, 'silent': 1, 'subsample':
    'objective': 'reg:linear',
    'max_depth': 14,
    'min_child_weight': 16,
}

scorer = make_scorer(mean_absolute_error)
eta_trials = [ 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]

for etas in eta_trials:
    xgb_params['eta'] = etas
    results = {}
    start = time()
    bst_cv1 = xgb.train(params=xgb_params, dtrain=dtrain, num_boost_round=50,
                             feval=scorer, maximize=False)
    end = time()
    results['train_time'] = end - start

    start = time()
    dpred_train = xgb.DMatrix(X_train)
    dpred_test = xgb.DMatrix(X_test)

    predictions_train = bst_cv1.predict(dtrain)
    predictions_test = bst_cv1.predict(dtest)
    end = time()
    results['pred_time'] = end - start

    y_tr = dtrain.get_label()
    y_te = dtest.get_label()
```

```
        results['acc_train'] = mean_absolute_error(np.exp(y_tr), np.exp(predictions_train))

        results['acc_test'] = mean_absolute_error(np.exp(y_te), np.exp(predictions_test))
        print "params {}".format(xgb_params)
        print "Trained score = {:.2f} and test score = {:.2f}".format(results['acc_train'],
        print "Trained time = {:.2f} and pred time = {:.2f}".format(results['train_time'],r
```

```
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.01, 'objecti
Trained score = 3004.95 and test score = 3024.16
Trained time = 48.75 and pred time = 0.97
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.05, 'objecti
Trained score = 1693.96 and test score = 1725.29
Trained time = 61.76 and pred time = 1.05
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.1, 'objectiv
Trained score = 1054.13 and test score = 1189.75
Trained time = 67.63 and pred time = 1.01
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.2, 'objectiv
Trained score = 958.58 and test score = 1195.94
Trained time = 69.29 and pred time = 1.05
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.3, 'objectiv
Trained score = 934.51 and test score = 1254.11
Trained time = 69.23 and pred time = 1.01
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.4, 'objectiv
Trained score = 916.83 and test score = 1308.84
Trained time = 69.87 and pred time = 1.04
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.5, 'objectiv
Trained score = 928.55 and test score = 1396.65
Trained time = 70.16 and pred time = 1.03
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.6, 'objectiv
Trained score = 949.02 and test score = 1495.01
Trained time = 1879.05 and pred time = 1.21
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.7, 'objectiv
Trained score = 994.89 and test score = 1635.51
Trained time = 957.67 and pred time = 1.12
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.8, 'objectiv
Trained score = 1061.31 and test score = 1788.86
Trained time = 1233.69 and pred time = 1.07
```

```
In [19]: from sklearn.metrics import mean_absolute_error
         from time import time
         from sklearn.metrics import make_scorer


         xgb_params = {'seed': 0, 'eta': 0.1, 'colsample_bytree': 0.5, 'silent': 1, 'subsample':
             'objective': 'reg:linear',
             'max_depth': 14,
             'min_child_weight': 16,
```

```
}

scorer = make_scorer(mean_absolute_error)
eta_trials = [ 0.01, 0.06, 0.08, 0.1, 0.2]


for etas in eta_trials:
    xgb_params['eta'] = etas
    results = {}
    start = time()
    bst_cv1 = xgb.train(params=xgb_params, dtrain=dtrain, num_boost_round=100,
                                feval=scorer, maximize=False)
    end = time()
    results['train_time'] = end - start

    start = time()
    dpred_train = xgb.DMatrix(X_train)
    dpred_test = xgb.DMatrix(X_test)

    predictions_train = bst_cv1.predict(dtrain)
    predictions_test = bst_cv1.predict(dtest)
    end = time()
    results['pred_time'] = end - start

    y_tr = dtrain.get_label()
    y_te = dtest.get_label()


    results['acc_train'] = mean_absolute_error(np.exp(y_tr), np.exp(predictions_train))

    results['acc_test'] = mean_absolute_error(np.exp(y_te), np.exp(predictions_test))
    print "params {}".format(xgb_params)
    print "Trained score = {:.2f} and test score = {:.2f}".format(results['acc_train'],
    print "Trained time = {:.2f} and pred time = {:.2f}".format(results['train_time'],r
```

```
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.01, 'objecti
Trained score = 2869.63 and test score = 2888.58
Trained time = 4273.70 and pred time = 1811.18
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.06, 'objecti
Trained score = 1015.01 and test score = 1166.69
Trained time = 1881.21 and pred time = 1.62
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.08, 'objecti
Trained score = 964.60 and test score = 1167.26
Trained time = 3403.50 and pred time = 1.73
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.1, 'objectiv
Trained score = 936.03 and test score = 1171.79
Trained time = 4147.82 and pred time = 1.65
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.2, 'objectiv
```

```
Trained score = 846.00 and test score = 1218.66
Trained time = 3613.07 and pred time = 1.68
```

```
In [20]: from sklearn.metrics import mean_absolute_error
         from time import time
         from sklearn.metrics import make_scorer


         xgb_params = {'seed': 0, 'eta': 0.1, 'colsample_bytree': 0.5, 'silent': 1, 'subsample':
             'objective': 'reg:linear',
             'max_depth': 14,
             'min_child_weight': 16,
         }

         scorer = make_scorer(mean_absolute_error)
         eta_trials = [0.02, 0.04, 0.05, 0.06, 0.7, 0.08 , 0.1]


         for etas in eta_trials:
             xgb_params['eta'] = etas
             results = {}
             start = time()
             bst_cv1 = xgb.train(params=xgb_params, dtrain=dtrain, num_boost_round=200,
                                         feval=scorer, maximize=False)
             end = time()
             results['train_time'] = end - start

             start = time()
             dpred_train = xgb.DMatrix(X_train)
             dpred_test = xgb.DMatrix(X_test)

             predictions_train = bst_cv1.predict(dtrain)
             predictions_test = bst_cv1.predict(dtest)
             end = time()
             results['pred_time'] = end - start

             y_tr = dtrain.get_label()
             y_te = dtest.get_label()


             results['acc_train'] = mean_absolute_error(np.exp(y_tr), np.exp(predictions_train))

             results['acc_test'] = mean_absolute_error(np.exp(y_te), np.exp(predictions_test))
             print "params {}".format(xgb_params)
             print "Trained score = {:.2f} and test score = {:.2f}".format(results['acc_train'],
             print "Trained time = {:.2f} and pred time = {:.2f}".format(results['train_time'],r
```

```
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.02, 'objecti
```

```
Trained score = 1152.18 and test score = 1245.20
Trained time = 4415.08 and pred time = 3.05
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.04, 'objecti
Trained score = 957.17 and test score = 1158.18
Trained time = 7037.68 and pred time = 3.18
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.05, 'objecti
Trained score = 929.89 and test score = 1158.17
Trained time = 1626.54 and pred time = 3.18
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.06, 'objecti
Trained score = 903.49 and test score = 1161.09
Trained time = 886.74 and pred time = 3.21
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.7, 'objectiv
Trained score = 560.38 and test score = 2240.84
Trained time = 1402.13 and pred time = 104.76
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.08, 'objecti
Trained score = 854.39 and test score = 1170.85
Trained time = 1139.51 and pred time = 3.15
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.1, 'objectiv
Trained score = 814.67 and test score = 1183.13
Trained time = 2389.95 and pred time = 3.23
```

In [21]:
```
from sklearn.metrics import mean_absolute_error
from time import time
from sklearn.metrics import make_scorer


xgb_params = {'seed': 0, 'eta': 0.04, 'colsample_bytree': 0.5, 'silent': 1, 'subsample'
    'objective': 'reg:linear',
    'max_depth': 14,
    'min_child_weight': 16,
}

scorer = make_scorer(mean_absolute_error)
gamma_trials = [0.1, 0.2, 0.4, 0.6]


for gamma in gamma_trials:
    xgb_params['gamma'] = gamma
    results = {}
    start = time()
    bst_cv1 = xgb.train(params=xgb_params, dtrain=dtrain, num_boost_round=200,
                                 feval=scorer, maximize=False)
    end = time()
    results['train_time'] = end - start

    start = time()
    dpred_train = xgb.DMatrix(X_train)
```

```
        dpred_test = xgb.DMatrix(X_test)

        predictions_train = bst_cv1.predict(dtrain)
        predictions_test = bst_cv1.predict(dtest)
        end = time()
        results['pred_time'] = end - start

        y_tr = dtrain.get_label()
        y_te = dtest.get_label()


        results['acc_train'] = mean_absolute_error(np.exp(y_tr), np.exp(predictions_train))

        results['acc_test'] = mean_absolute_error(np.exp(y_te), np.exp(predictions_test))
        print "params {}".format(xgb_params)
        print "Trained score = {:.2f} and test score = {:.2f}".format(results['acc_train'],
        print "Trained time = {:.2f} and pred time = {:.2f}".format(results['train_time'],r
```

```
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.04, 'objecti
Trained score = 957.55 and test score = 1158.19
Trained time = 3036.51 and pred time = 3.35
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.04, 'objecti
Trained score = 960.11 and test score = 1156.47
Trained time = 5552.76 and pred time = 3.20
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.04, 'objecti
Trained score = 965.57 and test score = 1155.44
Trained time = 2868.17 and pred time = 3.17
params {'subsample': 0.5, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.04, 'objecti
Trained score = 973.48 and test score = 1156.85
Trained time = 2245.24 and pred time = 3.20
```

0.0.11 Model Tunning

Because XGboost is very time consuming and the un-tuned model has already exceeded the bench mark set for this project we will only be tuning one parameter to see how the model is affected. Because the data set is so large we will only be using a 10% of the data to tune the model.

```
In [22]: from sklearn.metrics import mean_absolute_error
         from time import time
         from sklearn.metrics import make_scorer


         xgb_params = {'seed': 0, 'eta': 0.04, 'colsample_bytree': 0.5, 'silent': 1, 'subsample'
             'objective': 'reg:linear',
             'max_depth': 14,
             'min_child_weight': 16,
         }
```

```
scorer = make_scorer(mean_absolute_error)
subsample_trials = [0.2, 0.4, 0.6, 0.8, 1]


for subsample in subsample_trials:
    xgb_params['subsample'] = subsample
    results = {}
    start = time()
    bst_cv1 = xgb.train(params=xgb_params, dtrain=dtrain, num_boost_round=200,
                                feval=scorer, maximize=False)
    end = time()
    results['train_time'] = end - start

    start = time()
    dpred_train = xgb.DMatrix(X_train)
    dpred_test = xgb.DMatrix(X_test)

    predictions_train = bst_cv1.predict(dtrain)
    predictions_test = bst_cv1.predict(dtest)
    end = time()
    results['pred_time'] = end - start

    y_tr = dtrain.get_label()
    y_te = dtest.get_label()


    results['acc_train'] = mean_absolute_error(np.exp(y_tr), np.exp(predictions_train))

    results['acc_test'] = mean_absolute_error(np.exp(y_te), np.exp(predictions_test))
    print "params {}".format(xgb_params)
    print "Trained score = {:.2f} and test score = {:.2f}".format(results['acc_train'],
    print "Trained time = {:.2f} and pred time = {:.2f}".format(results['train_time'],r
```

```
params {'subsample': 0.2, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.04, 'objecti
Trained score = 1046.38 and test score = 1164.15
Trained time = 1754.89 and pred time = 2.74
params {'subsample': 0.4, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.04, 'objecti
Trained score = 981.12 and test score = 1157.57
Trained time = 449.59 and pred time = 3.18
params {'subsample': 0.6, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.04, 'objecti
Trained score = 932.72 and test score = 1154.18
Trained time = 742.23 and pred time = 3.45
params {'subsample': 0.8, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.04, 'objecti
Trained score = 895.25 and test score = 1153.16
Trained time = 1785.34 and pred time = 3.61
params {'subsample': 1, 'seed': 0, 'colsample_bytree': 0.5, 'silent': 1, 'eta': 0.04, 'objective
Trained score = 867.24 and test score = 1155.36
```

```
Trained time = 4341.03 and pred time = 3.75
```

```
In [23]: from sklearn.metrics import mean_absolute_error
         from time import time
         from sklearn.metrics import make_scorer


         xgb_params = {'seed': 0, 'eta': 0.04, 'colsample_bytree': 0.5, 'silent': 1, 'subsample'
             'objective': 'reg:linear',
             'max_depth': 14,
             'min_child_weight': 16,
             'gamma': 0.6
         }

         scorer = make_scorer(mean_absolute_error)
         colsample_bytree_trials = [0.2, 0.4, 0.6, 0.8, 1]


         for colsample_bytree in colsample_bytree_trials:
             xgb_params['colsample_bytree'] = colsample_bytree
             results = {}
             start = time()
             bst_cv1 = xgb.train(params=xgb_params, dtrain=dtrain, num_boost_round=200,
                                             feval=scorer, maximize=False)
             end = time()
             results['train_time'] = end - start

             start = time()
             dpred_train = xgb.DMatrix(X_train)
             dpred_test = xgb.DMatrix(X_test)

             predictions_train = bst_cv1.predict(dtrain)
             predictions_test = bst_cv1.predict(dtest)
             end = time()
             results['pred_time'] = end - start

             y_tr = dtrain.get_label()
             y_te = dtest.get_label()


             results['acc_train'] = mean_absolute_error(np.exp(y_tr), np.exp(predictions_train))

             results['acc_test'] = mean_absolute_error(np.exp(y_te), np.exp(predictions_test))
             print "params {}".format(xgb_params)
             print "Trained score = {:.2f} and test score = {:.2f}".format(results['acc_train'],
             print "Trained time = {:.2f} and pred time = {:.2f}".format(results['train_time'],r
```

```
params {'subsample': 1, 'seed': 0, 'colsample_bytree': 0.2, 'silent': 1, 'eta': 0.04, 'objective
```

```
Trained score = 953.99 and test score = 1163.07
Trained time = 1496.69 and pred time = 3.74
params {'subsample': 1, 'seed': 0, 'colsample_bytree': 0.4, 'silent': 1, 'eta': 0.04, 'objective
Trained score = 905.96 and test score = 1155.34
Trained time = 819.44 and pred time = 3.64
params {'subsample': 1, 'seed': 0, 'colsample_bytree': 0.6, 'silent': 1, 'eta': 0.04, 'objective
Trained score = 888.54 and test score = 1154.77
Trained time = 1211.21 and pred time = 3.64
params {'subsample': 1, 'seed': 0, 'colsample_bytree': 0.8, 'silent': 1, 'eta': 0.04, 'objective
Trained score = 876.74 and test score = 1156.33
Trained time = 3141.25 and pred time = 3.65
params {'subsample': 1, 'seed': 0, 'colsample_bytree': 1, 'silent': 1, 'eta': 0.04, 'objective':
Trained score = 867.72 and test score = 1157.52
Trained time = 1051.71 and pred time = 5.41
```

In [24]:
```
from sklearn.metrics import mean_absolute_error
from time import time
from sklearn.metrics import make_scorer


xgb_params = {'seed': 0, 'eta': 0.04, 'colsample_bytree': 0.6, 'silent': 1, 'subsample'
    'objective': 'reg:linear',
    'max_depth': 14,
    'min_child_weight': 16,
    'gamma': 0.6
}

scorer = make_scorer(mean_absolute_error)


results = {}
start = time()
bst_cv1 = xgb.train(params=xgb_params, dtrain=dtrain, num_boost_round=400,
                                  feval=scorer, maximize=False)
end = time()
results['train_time'] = end - start

start = time()
dpred_train = xgb.DMatrix(X_train)
dpred_test = xgb.DMatrix(X_test)

predictions_train = bst_cv1.predict(dtrain)
predictions_test = bst_cv1.predict(dtest)
end = time()
results['pred_time'] = end - start

y_tr = dtrain.get_label()
```

```
        y_te = dtest.get_label()


        results['acc_train'] = mean_absolute_error(np.exp(y_tr), np.exp(predictions_train))

        results['acc_test'] = mean_absolute_error(np.exp(y_te), np.exp(predictions_test))
        print "params {}".format(xgb_params)
        print "Trained score = {:.2f} and test score = {:.2f}".format(results['acc_train'],resu
        print "Trained time = {:.2f} and pred time = {:.2f}".format(results['train_time'],resul
```

```
params {'subsample': 1, 'seed': 0, 'colsample_bytree': 0.6, 'silent': 1, 'eta': 0.04, 'objective
Trained score = 796.51 and test score = 1153.28
Trained time = 904.37 and pred time = 9.45
```

0.0.12 Conclusion

After attempting to tune the model using GridSearchCV the un-tuned model mean absolute error 1261 and the tuned model had a mean absolute error of 1312. The goal is to have the lowest mean absolute error so we will have to go with the un-tuned model. There is still many ways of improving the model that can be attempted in the feature. To start off with because there are so many features in the data one can improve the accuracy of the model by finding which features are most important. Along with this correlation between features is a good way to visualize how the different features are interacting with each other.

```
In [25]: y_dummy = range(1,125547)

         d = {'loss': y_dummy}
         df = pd.DataFrame(data=d)
```

```
In [26]: dtest_final = xgb.DMatrix(data=cat_cont_test_data,label=y_dummy)
```

```
In [27]: final_predictions = bst_cv1.predict(dtest_final)
         test_data_id
         print final_predictions.shape
         print test_data_id.shape
```

```
(125546,)
(125546,)
```

```
In [28]: d = {'id': test_data_id, 'loss': final_predictions}
         df = pd.DataFrame(data=d)
         loss_final = np.exp(df['loss'])
         print
         df_final = pd.concat([test_data_id, loss_final], axis=1)



         display(df_final.head())
```

```
   id         loss
0   4  1397.102905
1   6  1865.396729
2   9  5971.495605
3  12  4782.812500
4  15  1503.430176
```

```
In [29]: df_final.to_csv('submission.csv')
```