In-Class Kaggle Competition Writeup

Darien Zhang

1 Exploratory Analysis

There are much more categorical variables than continuous variables. Since this will be a regression challenge, it may be better if I first transform the categorical variables to numbers and dig into the correlations and distribution of the continuous variables.

Here is the correlation plot of all variables and continuous variables.

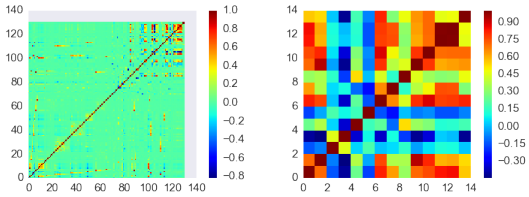


Figure 1: The left shows correlation among all variables. The right shows correlation among all continuous variables

Here is the boxplots of the continuous variables.

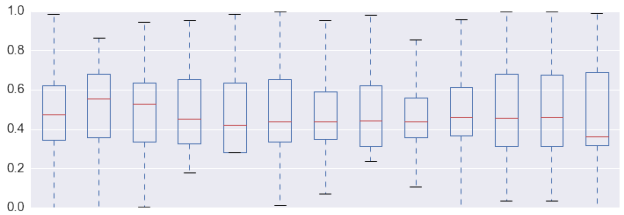


Figure 2: Boxplots of continuous variables

Since there are some strong correlations detected and considering we have quite a lot of variables, one thing we can do is to conduct PCA for possible future usage. To do this, I also need to scale the data features.

From the EDA we can get some basic idea of the data and some data preprocessing can be conducted. The detailed steps of data preprocessing can be found in the code. The most important thing I have done is turning categorical variables to numerical and scaled the data for future usage (like training neural networks). Since from the box plot we can see that there is not much outliers, we do not need robust scaler when scaling data. Furthermore, We could see some strong correlations among continous variable. This is why I try Ridge regression and use principle components as inputs in some models.

2 Models

Since this is a regression problem, I first tried on the multivariate linear regression as the baseline model. Based on this. I also tried Principle Component regression and Ridge regression. However, the predictive accuracy is not very ideal. Thus, to depict such complicated non-linear realtionship and make predictions, I would like to use Random Forest and Neural Network as two of the algorithms to try on for this competition. We can import RandomForestRegressor from sklearn.ensemble, which will provide a good random forest regressor (not classifier) suitable for this problem. One big reason for choosing Random forest is that there are fewer hyperparameters to tune (Basically number of trees and max depth) and it is a ensemble method which can be powerful. Theoretically speaking, though I believe that some boosted version of Random Forest such as Gradient Boosting Random Forest may be more powerful in making nice predictions, it runs extremely slow on my old computer (even slower than the neural nets I tried later on), I have to choose to move on with another model.

The motivation for choosing neural network mainly comes from the theorem that neural nets with ReLU as activation function can approximate any function. Since this problem seems to be a fairly complicated nonlinear regression problem, it is worth trying to spend some time on training neural networks. Different from Computer Vision or NLP problems, we need not refer to such complicated models as CNN, RNN, LSTM, etc, so the models I use here are quite simple: there will not be any convolution, maxpooling or softmax layer. All I have used are just the fully connected layers with some dropouts (this may empirically mitigate overfitting issues). In python, Keras is a powerful wrapper for deep learning and it is fairly easy to build neural nets with it. The backend I choose is Theano.

Comparing to Random Forest, one big issue of neural networks is that there are so many parameters to tune and it is not fast training process, especially when I am not working on GPU. Unlike in other cases when we can take advantage of some previously constructed network structures by other scholars like LeNet and VGGnet, I need to build a network from scratch. A more decent way is to do a lot of literature researches for constructing the network structure, but it may be too much for this competition. Thus, the network I am building is actually not very deep. I would be satisfied to see if it can beat the Random Forest model and my main focus is whether the number of layers will matter. Thus, though I also tuned the number of neurons and the dropout proportions, etc, I would like to see if the predictions can be better as the model becomes deeper.

3 Training

Training a Random Forest regressor is a little different from what we have learned in class. In regression problems, since the target values are continuous, we will fit a regression model with each of the features. With a bunch of candidate splitting values, we can split the data at these points and calculate Sum of Squared Error(SSE) between the predicted values and the true values. We will pick the variable that minimize the SSE and treat it as the splitting feature for the node.

In terms of the neural network model, we should first have a basic idea of the structure of the network. For every record, we are to use all the features as inputs and the output will be a single number. Thus the dimensionality of the input layer will be equal to the number of features we have. The activation function I chose was ReLU and mean absolute error was the loss metric. AdaDelta is picked for a better performance consideration. It does some trick on the learning rate that would be different from the Stochastic Gradient Descent we studied in class. AdaDelta uses exponentially decaying average of the second moment of gradient. To train the neural network models, weights are gained through backpropagation. I also added some dropout so that some neurons of some layer will be randomly picked out. This may be good to prevent overfitting. The input feature should empirically best to be scaled. This is where my scaled set of data is used to train for neural networks. The output values "loss" is not scaled.

The running time varies. In Random Forest, it would take about 10 minutes to over an hour as we increase the number of trees. As for the Neural Networks, since the model is not very deep, it typically takes about

five to ten minutes to train.

4 Hyperparameter Selection

For the Random Forest models, the hyperparameters I tuned are number of trees and max depth, while the former is the primary hyperparameter I focus on. There seems to be obvious prediction advantage when using bigger depth (by comparing between the first two "300" -tree models, with the first one having smaller max depth). To see a pattern, I have tried to gradually increase the number of the trees. Since it is quite time consuming to train and tune one model, I have picked five sets of hyperparameters to find the pattern. A model with top 20 features is also tried to make comparison.

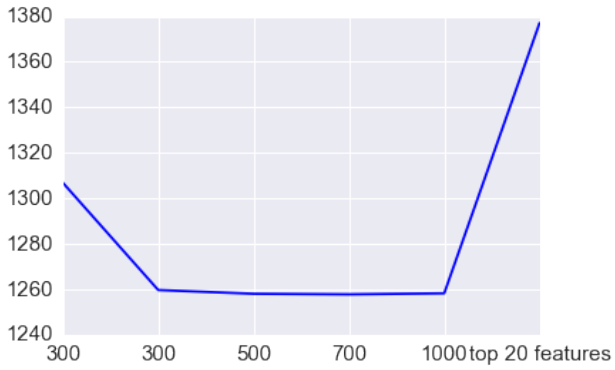


Figure 3: Number of trees and Mean absolute error plot for Random Forest Regressors

We could see that too many trees or not enough trees are both bad for prediction. Overall, the MAEs are above 1200, which is acceptable but not ideal.

As for the Neural Network models, as I have explained in the "Models" Section, I could only try my best to focus on one aspect of the hyperparameters here in this task, I would like to see if a deeper network can be better than shallower networks or even Random Forest. Most of the time of this task is spent on this part and I am not showing the process of tuning neuron because it would be a little verbose and in the end I am not sure if this is the best set of number of neurons in every layer. Briefly speaking, my idea of such "rough" tuning is whether to decrease the number of neurons in every layer or first increase then decrease the number of neurons in every layer. I have also tried a couple of optimization metrics and I stopped trying once I get a relatively satisfying result (by comparing to the leaderboard). After setting these things up, I spend a lot of time tuning the depth of the network. In the Figure 4, I have mainly shown five models with more and more layers.

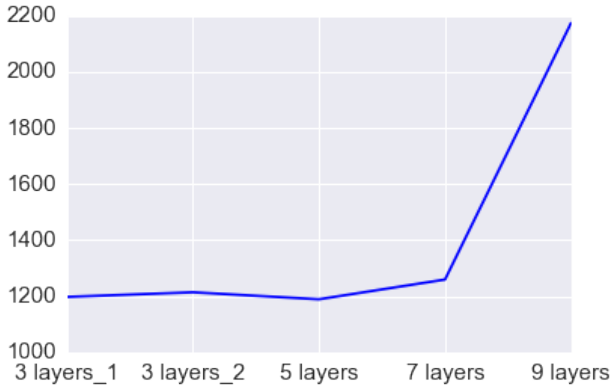


Figure 4: Number of Layers and Mean Absolute Error plot for Neural Networks

From the graph we can see that a deeper network with more layers is not necessarily better than a shallower network. While the numbers of neurons in the first two 3-layer cases show that there may not be significant difference in terms of small change in number of neurons in layers. The best model with the smallest MAE is the five-layer Neural Network, which is used for final prediction.

5 Data Splits

I have made up three data sets for split. One is the original data set with numerized categorical variables. It is split into $X_{train}, X_{test}, y_{train}, y_{test}$ with test size being one-third of the origin data rows. I have done the same thing for the scaled dataset and top principle component data set.

Since the data is split in this way, we can do cross validation and make comparison between models. I would train my model on the same subset of the data so that the predictive accuracy on my test data sets can be used to compare among different models. Since I using mean-absolute-error as the metric, it would also be convenient to pick the best prediction to upload and compare with other teams.

6 Errors and Mistakes

The hardest part of the task is tuning the hyperparameters as the models I choose require a lot of computation. One model I have tried on but is not included in the report is the Gradient Boosting Random Forest. It took so long time to train a single model that it would be impossible to tune hyperparameters with my computer. The two models I choose here took much less time than GBRF but still can be very time-consuming. To tune the hyperparameter of Neural Network, I could only focus on one aspect (number of layers in this case) as the main hyperparameter to tune. Though I have tried various number of neurons in different layers and different optimization methods and some other hyperparameters to get the relatively best predicitons, I could not thoroughly go through each and every aspect of the hyperparameters of neural networks.

7 Predictive Accuracy

Username: `yz273_Darien Zhang`

From the mean-absolute-error values of the models I have tried, we can conclude that Neural Networks perform better than Random Forest Regressors. Within the Neural Networks, it is not necessarily true that deeper structure works better. A comparision of effectiveness among the models via mean-absolute-error can be seen in Figure 3 and 4. The best network I have tried is a five-layer network. Here I have attached the

graphs of the true values and predictions of the best model for displaying the accuracy.

Figure 5: First 100 predictions and true values (Red points are predictions)

Figure 6: All predictions and true values (Red points are predictions)

From the graphs we can see that the prediction catches the pattern pretty nicely, while the exact predictions on some big values are not ideal, which causes an increase in the MAE.

8 Code
(attached with output from ipython notebook)

HW5_Kaggle_copy

November 29, 2016

In [2]:
```
import pandas as pd
import time
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.cross_validation import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt
```

In [3]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn import preprocessing
from keras.layers.advanced_activations import PReLU
from keras.wrappers.scikit_learn import KerasRegressor
```

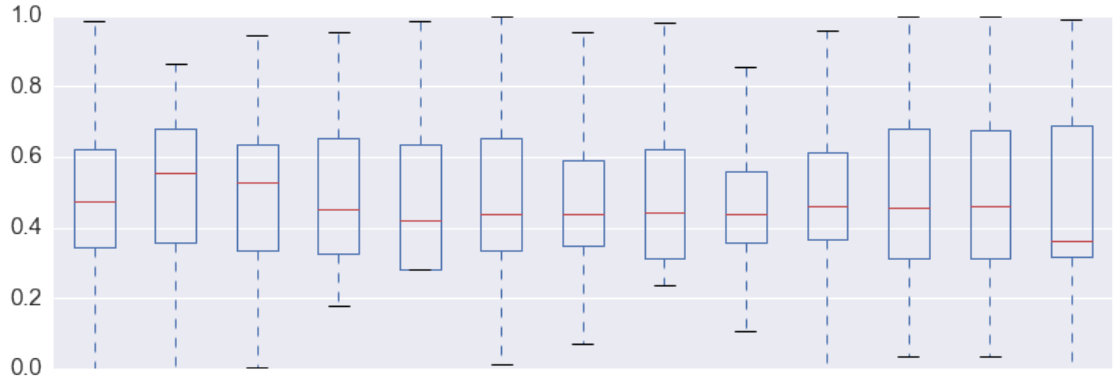
In [4]:
```
pml_example_submission = pd.read_csv('pml_example_submission.csv', sep = ',
pml_test_features = pd.read_csv('pml_test_features.csv', sep = ',')
pml_train = pd.read_csv('pml_train.csv', sep = ',')
```

1. Exploratory Analysis

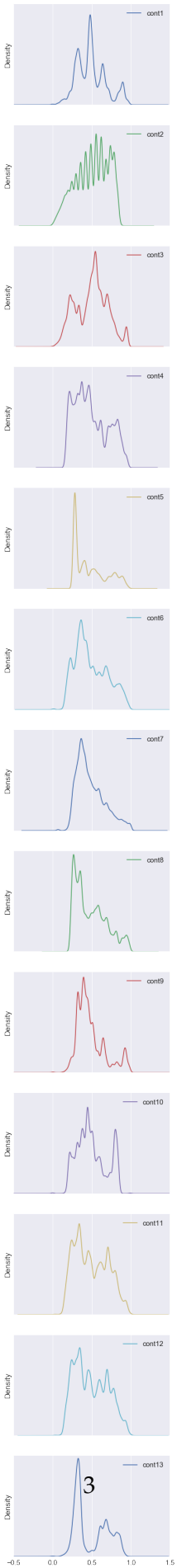
In [5]:
```
#store the target variable to "loss"
data, loss = pml_train.ix[:, :-1], pml_train.ix[:, -1]
```

In [6]:
```
#Transform categorical variables to numerical
np.random.seed(123)
label_encoder = preprocessing.LabelEncoder()
for i in range(1,117):
    data.ix[:,i] = label_encoder.fit_transform(data.ix[:,i])
```

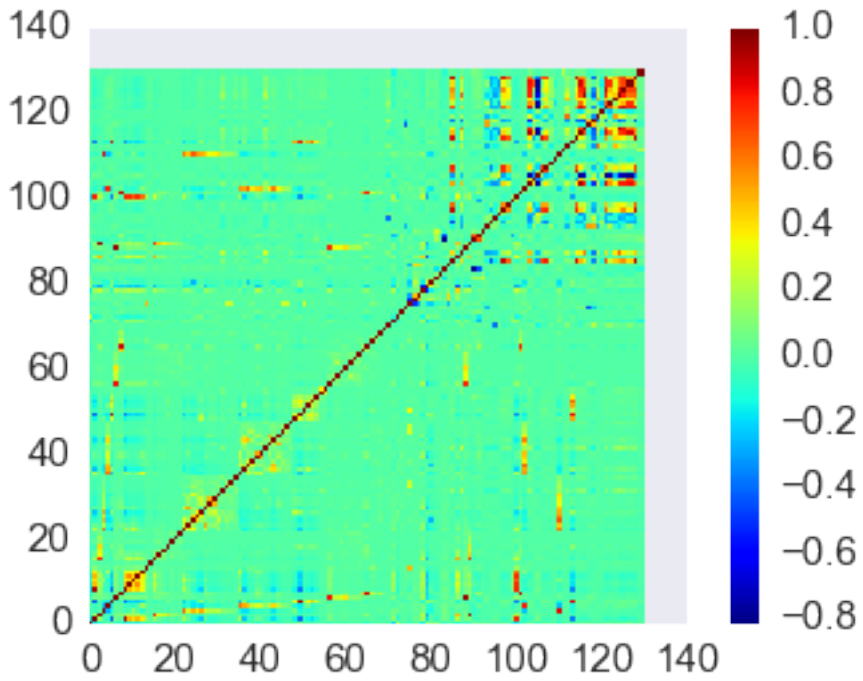
In [33]:
```
#Box plot of the continuous variables
data.ix[:,-14:-1].plot.box(figsize=(12,4), xticks=[])
pass
```



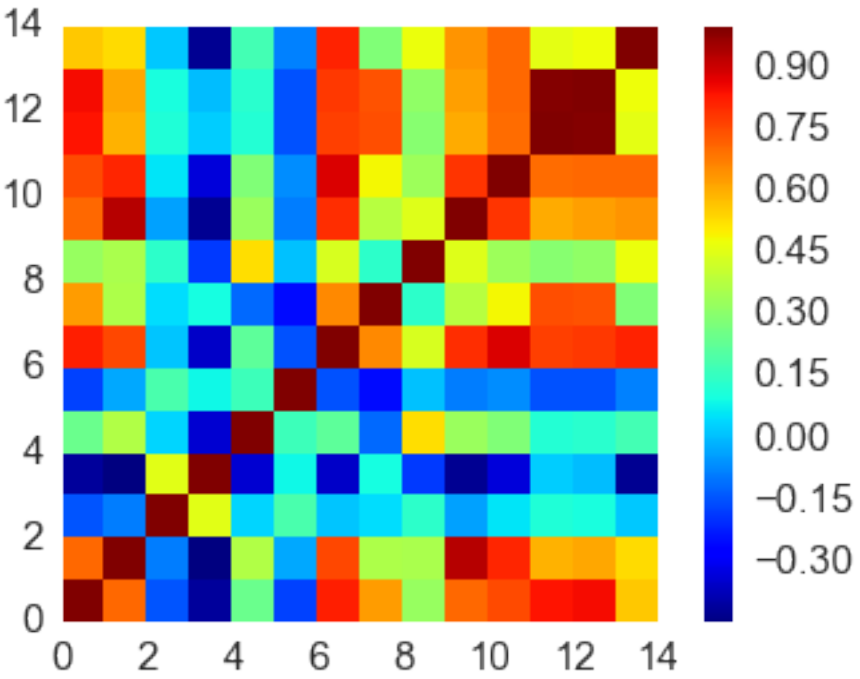
In [34]:
```
#Density plot of the continuous variables
data.ix[:,-14:-1].plot.density(figsize=(6, 60), subplots=True, yticks=[])
pass
```


cont1
Density
cont2
Density
cont3
Density
cont4
Density
cont5
Density
cont6
Density
cont7
Density
cont8
Density
cont9
Density
cont10
Density
cont11
Density
cont12
Density
cont13
Density
−0.5
0.0
0.5
1.0
1.5

In [10]:
```
#Correlation plot of all the variables
heatmap = plt.pcolor(data.ix[:,1:].corr(), cmap='jet')
plt.colorbar(heatmap)
plt.gca().set_aspect('equal')
```



In [13]:
```
#Correlation plot of all the continuous variables (Top right corner of the
heatmap = plt.pcolor(data.ix[:,-15:-1].corr(), cmap='jet')
plt.colorbar(heatmap)
plt.gca().set_aspect('equal')
```



In [8]:
```
#Scale the data
data_scaled = DataFrame(StandardScaler().fit_transform(data.ix[:,1:]), colu
data_scaled.head(10)
```

Out[8]:

	cat1	cat2	cat3	cat4	cat5	cat6	cat7	\
0	-0.574436	-0.875693	-0.240963	-0.681466	1.381206	-0.654966	-0.157428	
1	1.740837	-0.875693	-0.240963	-0.681466	-0.724005	1.526797	-0.157428	
2	-0.574436	-0.875693	-0.240963	1.467424	-0.724005	-0.654966	-0.157428	
3	1.740837	-0.875693	-0.240963	-0.681466	-0.724005	1.526797	-0.157428	
4	-0.574436	1.141953	-0.240963	-0.681466	-0.724005	-0.654966	-0.157428	
5	-0.574436	-0.875693	-0.240963	-0.681466	-0.724005	1.526797	-0.157428	
6	1.740837	1.141953	-0.240963	-0.681466	-0.724005	-0.654966	-0.157428	
7	-0.574436	-0.875693	-0.240963	1.467424	-0.724005	-0.654966	-0.157428	
8	-0.574436	-0.875693	-0.240963	-0.681466	1.381206	-0.654966	-0.157428	
9	1.740837	-0.875693	-0.240963	-0.681466	1.381206	-0.654966	-0.157428	

	cat8	cat9	cat10	...	cont5	cont6	cont7	\
0	-0.250047	-0.816525	-0.419393	...	0.434548	0.874233	0.171934	
1	-0.250047	-0.816525	-0.419393	...	-0.984678	-0.027890	1.191833	
2	-0.250047	-0.816525	-0.419393	...	0.063010	-0.680533	-0.679232	
3	-0.250047	-0.816525	-0.419393	...	-0.984678	1.257057	0.299208	
4	-0.250047	1.224702	-0.419393	...	0.021400	-0.826347	-0.888080	
5	-0.250047	-0.816525	-0.419393	...	-0.308476	-0.842553	-0.686890	
6	-0.250047	1.224702	-0.419393	...	-0.586125	-0.299445	1.885153	
7	3.999243	-0.816525	-0.419393	...	-0.103174	-1.190716	-1.330394	

```
8 -0.250047 -0.816525 -0.419393    ...     0.021400 -0.743993  0.188480
9 -0.250047 -0.816525 -0.419393    ...    -0.308476 -1.317836 -1.017267

      cont8     cont9    cont10    cont11    cont12    cont13    cont14
0  0.341392  0.289982 -0.137154 -0.006243 -0.056590  1.240723 -0.680046
1 -0.656860 -0.432880 -0.490427 -0.117691 -0.165583 -0.885405 -0.113837
2  0.488610 -0.229253 -0.547897 -0.723694 -0.673171 -0.721308 -0.952167
3  0.311677  0.105147  0.072149  1.042080  0.982149  1.020278  1.260224
4 -0.870334 -0.488485 -0.633158 -1.183604 -1.201461 -1.088564 -0.690925
5 -1.092842 -0.388021 -0.315218 -0.886780 -0.913631 -0.831756 -0.905298
6 -0.137356  0.024137 -0.519243  0.198153  0.143880 -0.885405 -0.953174
7 -1.020915 -0.892818 -1.433961 -1.113331 -1.133307 -0.790814 -0.105372
8 -0.629253 -1.440158 -0.047530  0.142526  0.089243 -1.672169 -1.083086
9 -1.045158 -0.565582 -1.010884 -1.183604 -1.201461 -0.911816 -0.864821

[10 rows x 130 columns]
```

Conduct PCA

```
In [9]: from sklearn.decomposition import PCA
        pca = PCA()
        data_scaled_pca = DataFrame(pca.fit_transform(data_scaled))

In [10]: #Get the cumulative explained variance ratio
         v = pca.explained_variance_ratio_
         vc = v.cumsum()

In [11]: #Pick those with cumulative explained variance ratio over 80%
         n_comps = 1 + np.argmax(vc > 0.8)
         n_comps

Out[11]: 60

In [12]: #Pick the principle components
         data_scaled_pca = data_scaled_pca.ix[:, 1:n_comps]
         data_scaled_pca.shape

Out[12]: (131822, 60)
```

2 Models

3 Training

4 Hyperparameter Selection

5 Data Splits

```
In [14]: # Split into train and test sets
         X_train_pc, X_test_pc, y_train_pc, y_test_pc = \
           train_test_split(data_scaled_pca, loss, test_size=0.33, random_state=42)

In [15]: X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled = \
           train_test_split(data_scaled, loss, test_size=0.33, random_state=42)

In [16]: X_train, X_test, y_train, y_test = \
           train_test_split(data.ix[:,1:], loss, test_size=0.33, random_state=42)

In [71]:
```

Linear Regression (Baseline Model)

```
In [17]: from sklearn.linear_model import LinearRegression
         # Initialize the model class.
         model = LinearRegression()
         # Fit the model to the training data.
         model.fit(X_train, y_train)
         # Generate our predictions for the test set.
         predictions = model.predict(X_test)
         # Compute error between our test predictions and the actual values.
         mean_absolute_error(predictions, y_test)

Out[17]: 1336.4525302990494

In [18]: #Try on Principle Component regression
         model.fit(X_train_pc, y_train_pc)
         predictions_pc = model.predict(X_test_pc)
         mean_absolute_error(predictions_pc, y_test)

Out[18]: 1363.9549670492449
```

Ridge Regression with 6 hyperparameters

```
In [102]: from sklearn.linear_model import Ridge
          alpha_ridge = [1e-7,1e-2, 1, 5, 10, 20]
          for i in range(6):
              ridgereg = Ridge(alpha=alpha_ridge[i],normalize=True)
              ridgereg.fit(X_train,y_train)
              predictions = ridgereg.predict(X_test)
              print(mean_absolute_error(predictions, y_test))

1336.45228243
1334.91979909
1365.32963806
1562.27302763
1683.09304301
1789.88330235
```

Random Forest

```
In [105]: start_time = time.time()
          from sklearn.ensemble import RandomForestRegressor
          params = {'n_estimators': 300, 'max_depth': 10, 'min_samples_split': 2}
          rf_model1 = RandomForestRegressor(**params)
          rf_model1.fit(X = X_train ,  y = y_train)
          rf_predictions1 = rf_model1.predict(X_test)
          print(mean_absolute_error(y_test, rf_predictions1))
          # print("OOB accuracy: ")
          # print(rf_model.oob_score_)
          print("--- %s seconds ---" % (time.time() - start_time))

1306.81986675
--- 444.14832305908203 seconds ---


In [106]: start_time = time.time()
          params = {'n_estimators': 300, 'max_depth': 50, 'min_samples_split': 2}
          rf_model2 = RandomForestRegressor(**params)
          rf_model2.fit(X = X_train ,  y = y_train)
          rf_predictions2 = rf_model2.predict(X_test)
          print(mean_absolute_error(y_test, rf_predictions2))
          # print("OOB accuracy: ")
          # print(rf_model.oob_score_)
          print("--- %s seconds ---" % (time.time() - start_time))

1259.6701512
--- 1176.3717801570892 seconds ---


In [107]: start_time = time.time()
          params = {'n_estimators': 500, 'max_depth': 50, 'min_samples_split': 2}
          rf_model3 = RandomForestRegressor(**params)
          rf_model3.fit(X = X_train ,  y = y_train)
          rf_predictions3 = rf_model3.predict(X_test)
          print(mean_absolute_error(y_test, rf_predictions3))
          # print("OOB accuracy: ")
          # print(rf_model.oob_score_)
          print("--- %s seconds ---" % (time.time() - start_time))

1258.02643895
--- 1979.8514709472656 seconds ---


In [108]: start_time = time.time()
          params = {'n_estimators': 700, 'max_depth': 50, 'min_samples_split': 2}
          rf_model4 = RandomForestRegressor(**params)
          rf_model4.fit(X = X_train ,  y = y_train)
```

```
          rf_predictions4 = rf_model4.predict(X_test)
          print(mean_absolute_error(y_test, rf_predictions4))
          # print("OOB accuracy: ")
          # print(rf_model.oob_score_)
          print("--- %s seconds ---" % (time.time() - start_time))
```

```
1257.76338724
--- 2785.47393989563 seconds ---
```

```
In [109]: start_time = time.time()
          params = {'n_estimators': 1000, 'max_depth': 50, 'min_samples_split': 2}
          rf_model5 = RandomForestRegressor(**params)
          rf_model5.fit(X = X_train ,  y = y_train)
          rf_predictions5 = rf_model5.predict(X_test)
          print(mean_absolute_error(y_test, rf_predictions5))
          # print("OOB accuracy: ")
          # print(rf_model.oob_score_)
          print("--- %s seconds ---" % (time.time() - start_time))
```

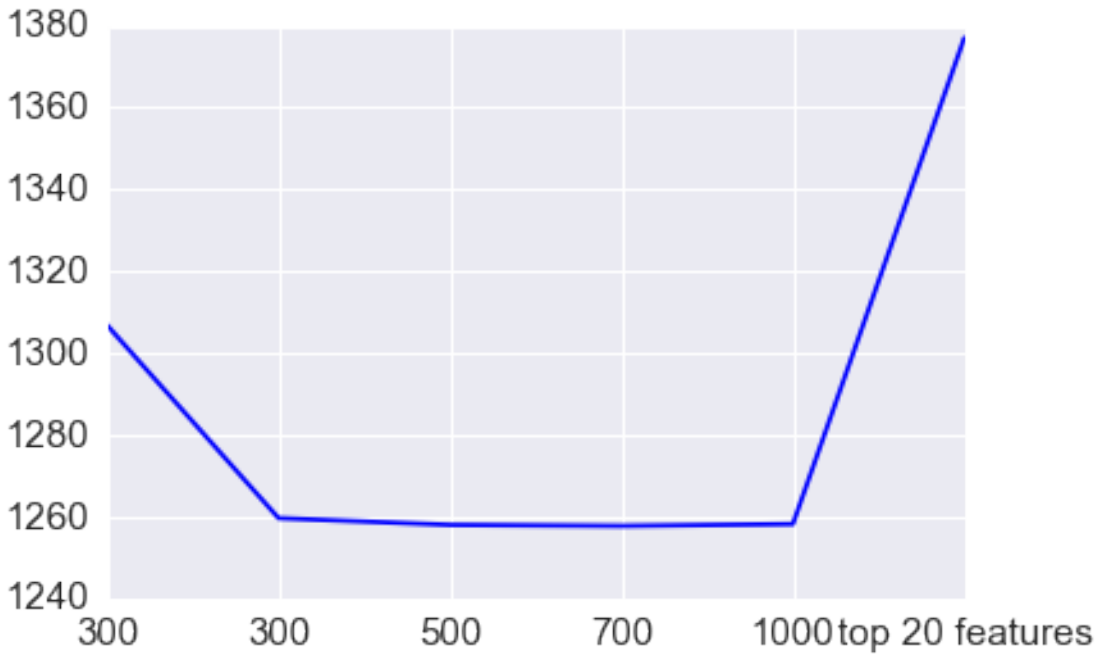
```
1258.22464009
--- 4028.0248260498047 seconds ---
```

```
In [61]: #The top 20 features
         sorted(zip(rf_model4.feature_importances_, X_train.columns), key=lambda x:
```

```
Out[61]: [(0.22650257745084235, 'cat80'),
          (0.10518972935805672, 'cont7'),
          (0.052579525370638362, 'cat57'),
          (0.045198070624659491, 'cont2'),
          (0.039310413707315368, 'cont14'),
          (0.03826421335568822, 'cat79'),
          (0.025616866252263587, 'cat12'),
          (0.022083141546236255, 'cat101'),
          (0.019038160053481325, 'cat81'),
          (0.017097463937983396, 'cont8'),
          (0.016472526987514436, 'cont3'),
          (0.016393838435648778, 'cat112'),
          (0.016053379444843443, 'cat100'),
          (0.015457549287409214, 'cont12'),
          (0.015358720237642585, 'cont1'),
          (0.014969670717810944, 'cont5'),
          (0.014378737624914444, 'cont11'),
          (0.01428304602923549, 'cont6'),
          (0.013901605240016135, 'cont4'),
          (0.012806271398941664, 'cont13')]
```

```
In [150]: start_time = time.time()
```

```
            from sklearn.ensemble import RandomForestRegressor
            params = {'n_estimators': 700, 'max_depth': 50, 'min_samples_split': 2}
            rfm = RandomForestRegressor(**params)
            var_idx = rf_model4.feature_importances_.argsort()[-20:]
            rfm.fit(X = X_train[var_idx] ,  y = y_train)
            rf_predictions_rfm = rfm.predict(X_test[var_idx])
            print(mean_absolute_error(y_test, rf_predictions_rfm))
            print("--- %s seconds ---" % (time.time() - start_time))

1377.0601899
--- 1091.7604010105133 seconds ---


In [151]: y = [mean_absolute_error(rf_predictions1, y_test),
               mean_absolute_error(rf_predictions2, y_test),
               mean_absolute_error(rf_predictions3, y_test),
               mean_absolute_error(rf_predictions4, y_test),
               mean_absolute_error(rf_predictions5, y_test),
               mean_absolute_error(rf_predictions_rfm, y_test)]
           x = range(len(y))
           LABELS = ['300','300','500', '700', '1000', 'top 20 features']

           plt.plot(x, y, color="blue")
           plt.xticks(x, LABELS)

Out[151]: ([<matplotlib.axis.XTick at 0x67faa29e8>,
            <matplotlib.axis.XTick at 0x67e3785c0>,
            <matplotlib.axis.XTick at 0x67e3f8320>,
            <matplotlib.axis.XTick at 0x677bae278>,
            <matplotlib.axis.XTick at 0x67df17048>,
            <matplotlib.axis.XTick at 0x67faf6e48>],
           <a list of 6 Text xticklabel objects>)
```



Neural Networks

```
In [147]: #3 layers
          def base_model1():
              nn_model = Sequential()
              nn_model.add(Dense(400, input_dim=130, init='he_normal', activation='
              nn_model.add(Dense(50, init='he_normal', activation='relu'))
              nn_model.add(Dense(1, init='he_normal'))
              nn_model.compile(loss="mae", optimizer='adadelta', metrics=['accuracy
              return nn_model
          #3 layers, more neurons
          def base_model2():
              nn_model = Sequential()
              nn_model.add(Dense(50, input_dim=130, init='he_normal', activation='r
              nn_model.add(Dense(30, init='he_normal', activation='relu'))
              nn_model.add(Dense(1, init='he_normal'))
              nn_model.compile(loss="mae", optimizer='adadelta', metrics=['accuracy
              return nn_model
          #5 layers
          def base_model3():
              nn_model = Sequential()
              nn_model.add(Dense(400, input_dim=130, init='he_normal'))
              nn_model.add(PReLU())
              nn_model.add(Dense(200, init='he_normal'))
              nn_model.add(PReLU())
              nn_model.add(Dropout(0.4))
```

```
    nn_model.add(Dense(70, init='he_normal'))
    nn_model.add(PReLU())
    nn_model.add(Dropout(0.4))
    nn_model.add(Dense(50, init='he_normal'))
    nn_model.add(PReLU())
    nn_model.add(Dropout(0.2))
    nn_model.add(Dense(1, init='he_normal'))
    nn_model.compile(loss="mae", optimizer='adadelta', metrics=['accuracy
    return nn_model
#7 layers
def base_model4():
    nn_model = Sequential()
    nn_model.add(Dense(400, input_dim=130, init='he_normal'))
    nn_model.add(PReLU())
    nn_model.add(Dropout(0.4))
    nn_model.add(Dense(200, init='he_normal'))
    nn_model.add(PReLU())
    nn_model.add(Dropout(0.4))
    nn_model.add(Dense(100, init='he_normal'))
    nn_model.add(PReLU())
    nn_model.add(Dropout(0.4))
    nn_model.add(Dense(70, init='he_normal'))
    nn_model.add(PReLU())
    nn_model.add(Dropout(0.4))
    nn_model.add(Dense(50, init='he_normal'))
    nn_model.add(PReLU())
    nn_model.add(Dropout(0.2))
    nn_model.add(Dense(15, init='he_normal'))
    nn_model.add(PReLU())
    nn_model.add(Dropout(0.2))
    nn_model.add(Dense(1, init='he_normal'))
    nn_model.compile(loss="mae", optimizer='adadelta', metrics=['accuracy
    return nn_model

#9 layers
def base_model5():
    nn_model = Sequential()
    nn_model.add(Dense(400, input_dim=130, init='he_normal'))
    nn_model.add(PReLU())
    nn_model.add(Dropout(0.4))
    nn_model.add(Dense(300, init='he_normal'))
    nn_model.add(PReLU())
    nn_model.add(Dropout(0.4))
    nn_model.add(Dense(200, init='he_normal'))
    nn_model.add(PReLU())
    nn_model.add(Dropout(0.4))
    nn_model.add(Dense(100, init='he_normal'))
    nn_model.add(PReLU())
```

```
            nn_model.add(Dropout(0.4))
            nn_model.add(Dense(70, init='he_normal'))
            nn_model.add(PReLU())
            nn_model.add(Dropout(0.4))
            nn_model.add(Dense(50, init='he_normal'))
            nn_model.add(PReLU())
            nn_model.add(Dropout(0.4))
            nn_model.add(Dense(25, init='he_normal'))
            nn_model.add(PReLU())
            nn_model.add(Dropout(0.2))
            nn_model.add(Dense(15, init='he_normal'))
            nn_model.add(PReLU())
            nn_model.add(Dropout(0.2))
            nn_model.add(Dense(1, init='he_normal'))
            nn_model.compile(loss="mae", optimizer='adadelta', metrics=['accuracy
            return nn_model
```

In [ ]:

In [110]:
```
# base_model1
np.random.seed(123)
start_time = time.time()
# evaluate model
nn_estimator1 = KerasRegressor(build_fn=base_model1, nb_epoch=15, batch_s
nn_estimator1.fit(np.array(X_train_scaled),np.array(y_train))
nn_predictions1 = nn_estimator1.predict(np.array(X_test_scaled))
print(mean_absolute_error(nn_predictions1, y_test))
print("--- %s seconds ---" % (time.time() - start_time))
```

```
1197.81184538
--- 97.84829592704773 seconds ---
```

In [111]:
```
# base_model2
np.random.seed(123)
start_time = time.time()
# evaluate model
nn_estimator2 = KerasRegressor(build_fn=base_model2, nb_epoch=15, batch_s
nn_estimator2.fit(np.array(X_train_scaled),np.array(y_train))
nn_predictions2 = nn_estimator2.predict(np.array(X_test_scaled))
print(mean_absolute_error(nn_predictions2, y_test))
print("--- %s seconds ---" % (time.time() - start_time))
```

```
1214.22942562
--- 32.45439791679382 seconds ---
```

In [112]:
```
# base_model3
np.random.seed(123)
```

```
start_time = time.time()
# evaluate model
nn_estimator3 = KerasRegressor(build_fn=base_model3, nb_epoch=15, batch_s
nn_estimator3.fit(np.array(X_train_scaled),np.array(y_train))
nn_predictions3 = nn_estimator3.predict(np.array(X_test_scaled))
print(mean_absolute_error(nn_predictions3, y_test))
print("--- %s seconds ---" % (time.time() - start_time))
```

```
1189.3229291
--- 239.7537989616394 seconds ---
```

```
In [116]: # base_model4
          np.random.seed(123)
          start_time = time.time()
          # evaluate model
          nn_estimator4 = KerasRegressor(build_fn=base_model4, nb_epoch=15, batch_s
          nn_estimator4.fit(np.array(X_train_scaled),np.array(y_train))
          nn_predictions4 = nn_estimator4.predict(np.array(X_test_scaled))
          print(mean_absolute_error(nn_predictions4, y_test))
          print("--- %s seconds ---" % (time.time() - start_time))
```

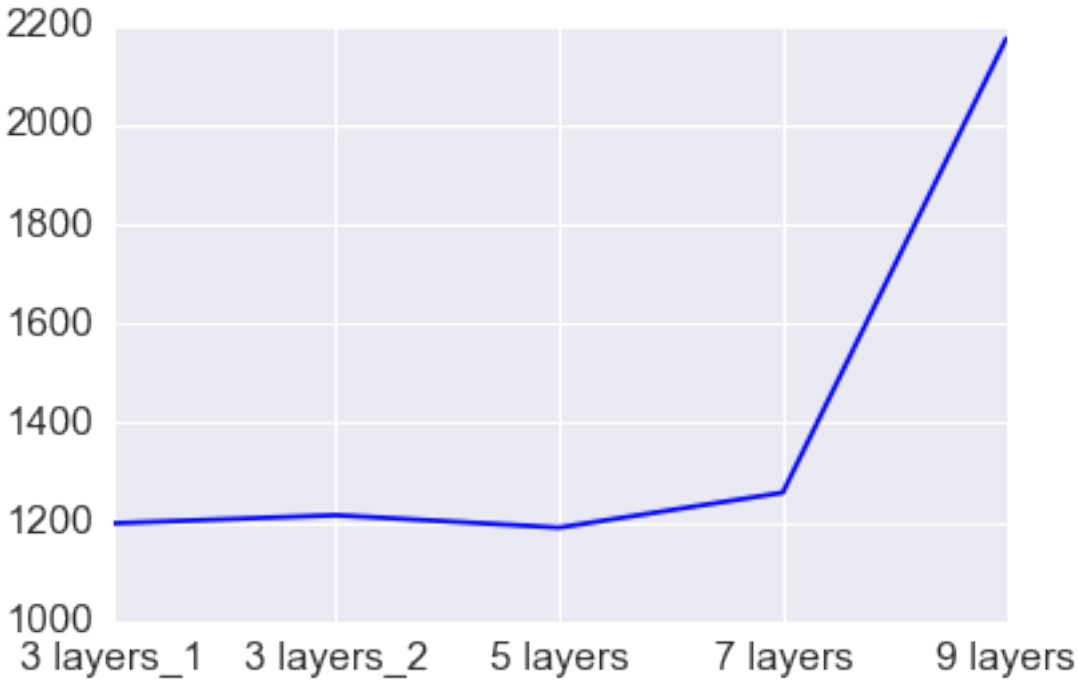
```
1259.61819061
--- 339.99370312690735 seconds ---
```

```
In [145]: # base_model5
          np.random.seed(123)
          start_time = time.time()
          # evaluate model
          nn_estimator5 = KerasRegressor(build_fn=base_model5, nb_epoch=15, batch_s
          nn_estimator5.fit(np.array(X_train_scaled),np.array(y_train))
          nn_predictions5 = nn_estimator5.predict(np.array(X_test_scaled))
          print(mean_absolute_error(nn_predictions5, y_test))
          print("--- %s seconds ---" % (time.time() - start_time))
```

```
2173.22319008
--- 416.2498002052307 seconds ---
```

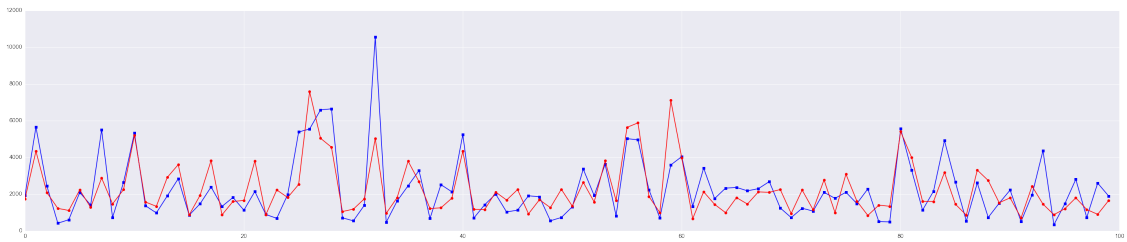
```
In [146]: y = [mean_absolute_error(nn_predictions1, y_test),
               mean_absolute_error(nn_predictions2, y_test),
               mean_absolute_error(nn_predictions3, y_test),
               mean_absolute_error(nn_predictions4, y_test),
               mean_absolute_error(nn_predictions5, y_test)]
          x = range(5)
          LABELS = ['3 layers_1','3 layers_2','5 layers', '7 layers', '9 layers']

          plt.plot(x, y, color="blue")
          plt.xticks(x, LABELS)
```

```
Out[146]: ([<matplotlib.axis.XTick at 0x67ecfdac8>,
            <matplotlib.axis.XTick at 0x67f68b320>,
            <matplotlib.axis.XTick at 0x6776ef828>,
            <matplotlib.axis.XTick at 0x170ab3160>,
            <matplotlib.axis.XTick at 0x170abbb38>],
           <a list of 5 Text xticklabel objects>)
```

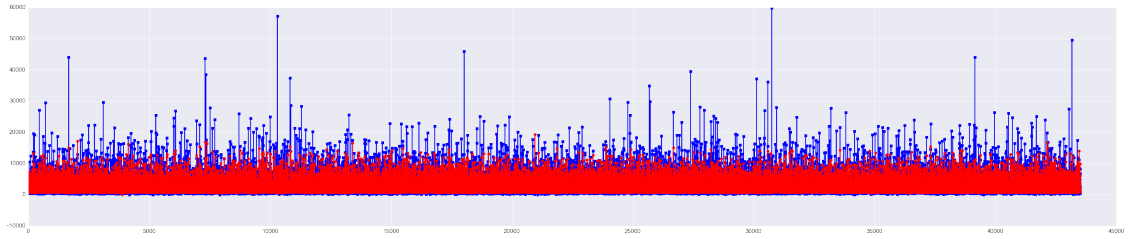


Predictive Accuracy

```
In [180]: import matplotlib.pyplot as plt
          plt.figure(figsize=(50,10))
          # First 100 predictions
          x =range(len(y_test[:101]))
          plt.plot(x,y_test[:101], c='b', marker="s")
          plt.plot(x,nn_predictions_3[:101], c='r', marker="o")
          plt.show()
```



```
In [182]: # all predictions
          plt.figure(figsize=(50,10))
          x =range(len(y_test))
          plt.plot(x,y_test, c='b', marker="s")
          plt.plot(x,nn_predictions_3, c='r', marker="o")
          plt.show()
```



Output submission file

```
In [27]: #same data preprocessing on pml_test_features
         np.random.seed(123)
         label_encoder = preprocessing.LabelEncoder()
         for i in range(1,117):
             pml_test_features.ix[:,i] = label_encoder.fit_transform(pml_test_featu

         pml_test_features_scaled = DataFrame(StandardScaler().fit_transform(pml_te
         pml_test_features_scaled.head(10)
```

Out[27]:

	cat1	cat2	cat3	cat4	cat5	cat6	cat7	\
0	1.737134	-0.871402	-0.240579	1.455353	-0.718798	1.520908	-0.158587	
1	-0.575661	-0.871402	-0.240579	1.455353	-0.718798	-0.657502	-0.158587	
2	-0.575661	1.147575	-0.240579	-0.687118	1.391211	-0.657502	-0.158587	
3	-0.575661	-0.871402	-0.240579	-0.687118	-0.718798	1.520908	-0.158587	
4	1.737134	1.147575	4.156631	-0.687118	-0.718798	-0.657502	-0.158587	
5	-0.575661	-0.871402	-0.240579	-0.687118	-0.718798	1.520908	-0.158587	
6	1.737134	-0.871402	-0.240579	-0.687118	-0.718798	1.520908	-0.158587	
7	-0.575661	-0.871402	-0.240579	-0.687118	-0.718798	1.520908	-0.158587	
8	-0.575661	-0.871402	-0.240579	-0.687118	1.391211	-0.657502	-0.158587	
9	-0.575661	-0.871402	4.156631	-0.687118	-0.718798	-0.657502	-0.158587	

	cat8	cat9	cat10	...	cont5	cont6	cont7	\
0	-0.248546	-0.812486	-0.417532	...	0.008897	-0.361438	-0.772029	
1	-0.248546	-0.812486	-0.417532	...	-0.518132	1.707849	0.102381	
2	-0.248546	1.230791	-0.417532	...	1.456589	1.540112	-0.035039	
3	-0.248546	-0.812486	-0.417532	...	1.997295	-1.224113	-1.129818	
4	-0.248546	1.230791	2.395024	...	0.883266	-0.846735	-0.686828	
5	-0.248546	-0.812486	-0.417532	...	-0.992069	0.912598	1.702978	
6	-0.248546	-0.812486	-0.417532	...	1.907119	-0.575773	-0.722635	
7	-0.248546	-0.812486	-0.417532	...	-0.992069	-0.468958	0.596701	

```
8 -0.248546 -0.812486 -0.417532    ...     0.050295 -0.772742 -0.603327
9 -0.248546 -0.812486 -0.417532    ...    -0.992069 -0.085006  1.501036

      cont8     cont9    cont10    cont11    cont12    cont13    cont14
0  0.483416  0.109226 -0.580600 -0.772076 -0.638527 -0.851536  0.729658
1  2.288586  2.456704  1.804031  1.616299  1.619002  1.493362 -0.519435
2  1.181718  2.514632  1.804031  2.093380  2.067071  1.660332  1.406645
3 -1.208633 -0.976359 -0.935917 -1.287854 -1.302544 -0.755767 -1.277959
4 -1.094311 -0.393676 -0.320309 -0.885088 -0.911905 -0.838053 -0.898454
5 -0.519895 -0.393676  0.477948  1.070277  1.055756  0.549522  1.253986
6 -0.631814 -0.235486 -0.261393 -0.962984 -0.942410 -0.727835 -0.521731
7 -1.140831 -0.348761 -0.692958  0.308159  0.252274 -0.917946 -1.232837
8  0.483416 -0.235486 -0.552108 -0.722320 -0.671898 -0.727835 -1.258302
9 -0.462960 -0.166961 -0.142918 -0.246419 -0.291120 -0.824500 -0.619774

[10 rows x 130 columns]
```

In [28]:
```
pml_submission = pml_example_submission.copy()
#Make prediction from the best model
nn_predictions_final = nn_estimator3.predict(np.array(pml_test_features_sc
```

In [30]:
```
# Create submission file
pml_submission.loss = nn_predictions_final
pml_submission.head()
```

Out[30]:

	id	loss
0	131822	1297.544556
1	131823	1654.102295
2	131824	3400.237305
3	131825	1063.073730
4	131826	3276.271973

In [31]:
```
# Store the file
pml_submission.to_csv('pml_submission.csv', index=False)
```

In [ ]: