Kyle Decker
STA561- Probabilistic Machine Learning
Kaggle Competition Writeup

1 Exploratory Analysis

The first step I took was to read the training data in from the csv file into a Pandas data frame in Python. I inspected the contents of the data frame by printing it out which allowed me to look at which variables were continuous and which were categorical. In order to handle the presence of categorical features I translated each categorical feature into a dummy variable using the 'get_dummies' function from the Pandas library.

I then computed some simple statistical measures on the data such as the mean, standard deviation, minimum, maximum, and several percentiles that helped me get a feel for the distribution of each of the continuous features. I noticed that a descent amount of the categorical features were uniform in value across the samples regardless of the corresponding loss. Clearly if the value does not change with respect to the loss that feature will not help in fitting a regression model to predict the loss. In result I dropped all features which took on the same value for more than 10,000 out of the ~100,000 samples.

I then created scatter plots of each of the continuous features vs. the corresponding loss. This allowed me to get a rough feeling for how each of the continuous variables are related to the loss. Also, by plotting each of the continuous variables I was able to notice a few of the distributions looked very similar. In fact, if I plotted the 'cont11' and 'cont12' variables against one another they appear to be highly correlated. Furthermore, I computed the coefficient of determination (R^2) between these two variables and attained a value of ~0.99 signifying that they are highly correlated. The presence of correlated features has a negative impact on training a machine learning model on the data and thus I dropped one of these features from the data. Due to the vast amount of features and difficulty in visualizing categorical features, I performed PCA in order to perform dimensionality reduction and ensure there are not highly correlated variables present in the data.

** Please see outputs in code section **

2 Models

The three models I decided to use were Random Forest regression, Bayesian Ridge regression, and Neural Network (Multi-layer Perceptron) regression. All models were available via the sklearn library in Python.

I chose to use a Random Forest (RF) model because it is highly capable of fitting complex non-linear decision functions in order to predict the output and it is generally easy to account for over fitting by simply adding more trees to the ensemble and limiting the minimum number of samples at each leaf node. Additionally, RF is quite computationally efficient for large sample sets in comparison to other non-linear algorithms, such as SVM with Gaussian or polynomial kernels.

I chose to use a Multi-layer Perceptron (MLP) model due to the ability to highly customize the architecture of the model allowing one to make either very simple or complex decision functions. Also, by using several hidden layers in this approach more complex features can be formed from highly non-linear combinations of the original input features.

In addition to the required two models, I also chose to use a Bayesian Ridge (BR) model mainly due to its simplicity (there are not many parameters to tune) as well as its computational efficiency for large datasets as in our case where our dataset has over 100,000 samples. The computational efficiency of this model allowed me to perform polynomial (2^{nd} degree) feature expansion, and still have training time be less than 1 minute.

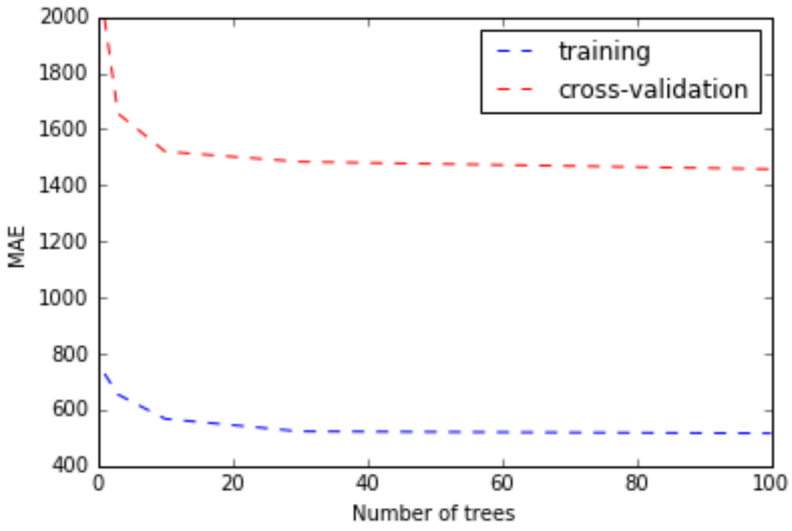
3 Training

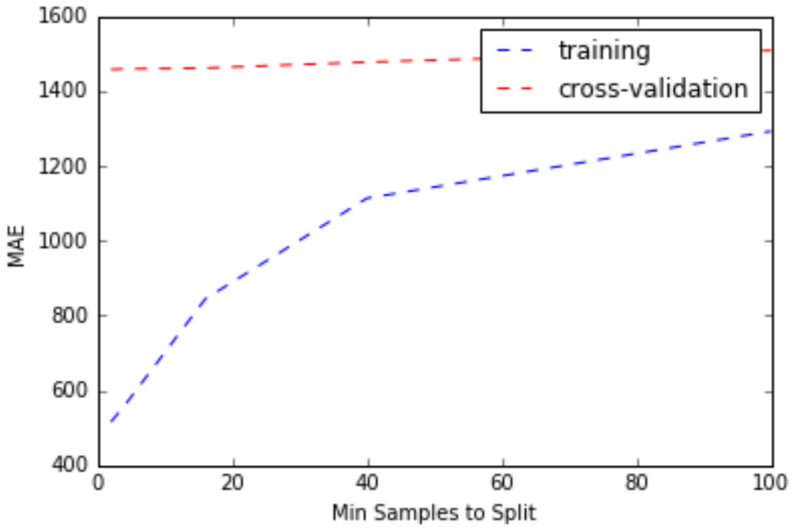
The Random Forest model is trained by fitting a set number of decision trees to the data provided input features and a target variable. The decision trees are trained independently from one another and the data used for training each tree is a bootstrapped sample of the original training set. Strategic splits are made on certain features at each node in a tree based on a mean squared error metric. The number of splits made is dependent on user input and by default splitting is stopped once all leaves are pure (provide same output) or can be changed to require a certain number of samples at each leaf in order to prevent overfitting. Once all of the decision trees are trained, the model can be used on a test set and the output from each tree will be averaged, resulting in superior performance and reduced likelihood of overfitting in comparison to a single tree. Runtime for Random Forest training is highly dependent on the hyperparameters such as the number of trees, number of features considered at each split, etc. A moderate estimate of runtime for training a random forest with 100 trees and limiting the number of features at each split on the provided training data is 135 seconds.

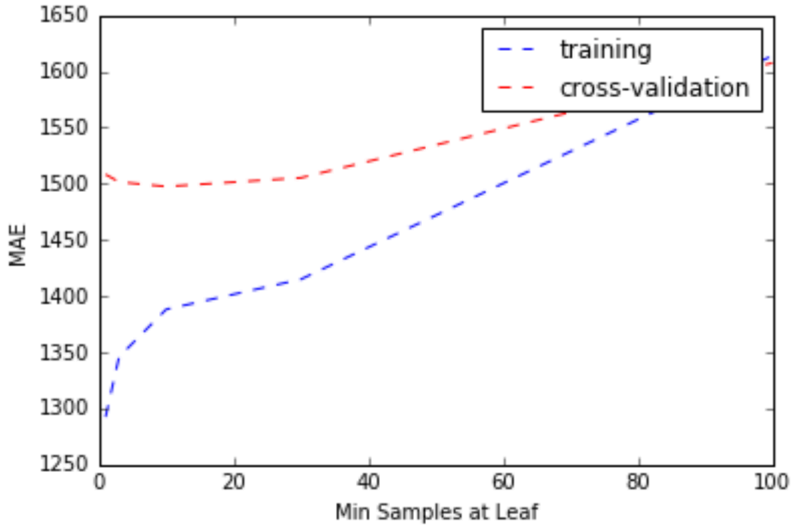
The Multi-layer Perceptron is a neural network that is trained through a process termed back propagation. Back propagation involves taking the predicted output from the network, comparing to the actual ground truth output and computing the error. The error used in this algorithm is the square error. That error is then propagated back to the previous layer and an error at each neuron within that layer is computed. That error is then propagated back to the previous layer, and so on until the input layer is reached. The weights at each neuron are updated in order to reduce the error via gradient descent. The inputs are then forward propagated back through the network until the output is generated. A new error is attained and the process is repeated until convergence. Additionally, L2 regularization is used in order to prevent overfitting. Runtime for training a multi-layer perceptron is highly dependent on the architecture of the hidden layers and the number of input samples. A moderate estimate for training runtime using 10 hidden layers each with 100 neurons is 52 seconds.

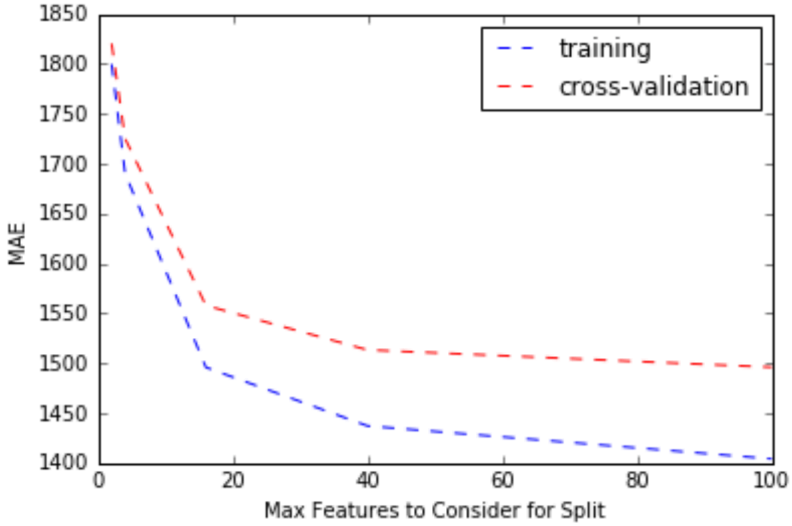
4 Hyperparameter Selection

The main hyperparameters that were tuned for the Random Forest model were the number of trees, the minimum number of samples to split on, the minimum number of samples at a leaf node, and the maximum number of features considered for each split. Initial tuning of parameters was performed using a smaller subset of samples in order to be computationally efficient. I tuned the parameters consecutively starting with the number of trees and ending with the maximum number of features to consider for each split. Plots showing the results are below:



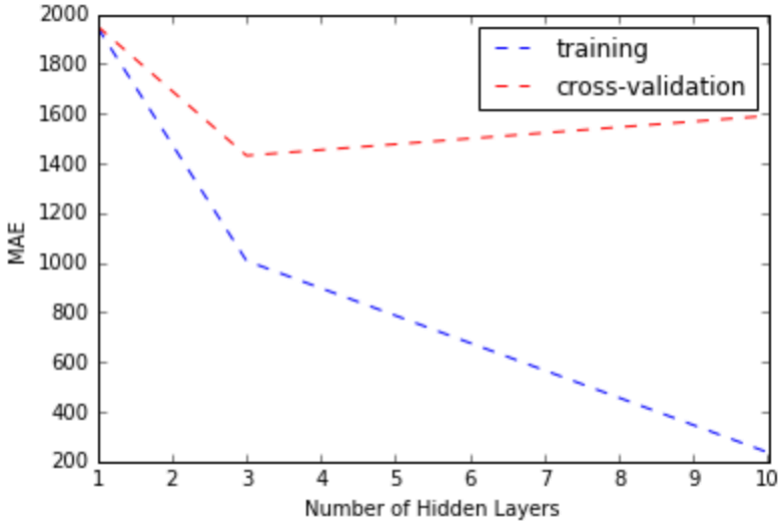


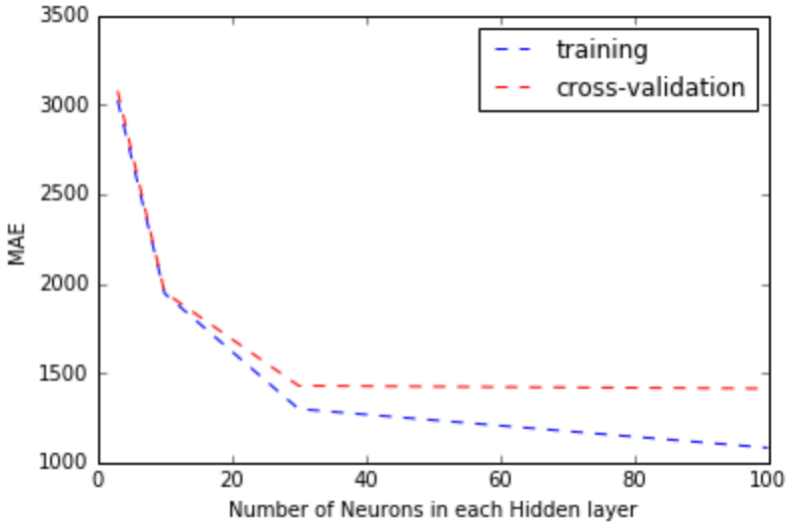




Best Parameters = 100 trees, 40 minimum samples to split, 20 minimum samples at a leaf, and 40 maximum features to consider for splitting.

The main hyperparameters that were tuned for the Multi-layer Perceptron model were the activation function, the number of hidden layers, and the size of the hidden layers. Initial tuning of parameters was performed using a smaller subset of samples in order to be computationally efficient. I tuned the parameters consecutively starting with the activation function and ending with the size of the hidden layers. The activation function with best results was the relu function. Plots showing the results are below:





Best Parameters = relu activation function, 3 hidden layers, 30 neurons in a hidden layer.

5 Data Splits

Due to the large size of the provided training data set, I partitioned the original training data down in to training (80% of the original training data) and cross validation (20% of the original training data) after random permutation of the samples. The model parameters were fit to the training data and the cross-validation was used as an independent data set to judge the algorithms performance with a certain set of parameters. If the performance (mean-absolute-error) was similar in value for both the training and the cross validation set then over fitting is unlikely, whereas if the MAE was much lower for the training set than the cross validation set it is likely that over fitting is an issue.

6 Errors and Mistakes

One of the most time consuming parts of this competition was accounting for the categorical variables. I originally began working in Matlab for the competition and quickly realized the ability to account for categorical variables and assign dummy variables is highly limited and in result switched to Python. Even when switching to python I still found dealing with the categorical variables to be difficult because I could not think of a clever way to visualize their distributions easily, and plot the categorical features that had many different labels within a single feature.

7 Predictive Accuracy

Kaggle Username: Kyle Decker

Below are the mean absolute error performances of each algorithm on the test set:

Multi-layer Perceptron Regression averaged with Random Forest: Below 1200
-Please check actual score on kaggle site, ran out of submissions prior to submission deadline of write up
Multi-layer Perceptron Regression: 1202.99
Random Forest Regression: 1224.42
Bayesian Ridge Regression with polynomial (2^{nd} degree) features: 1234.33
LASSO regression with PCA features: 1289.27

I believe the MLP was able to have the best performance because it was able to create a highly non-linear decision function and create unique combination of features by using several hidden layers. LASSO performed worst due to the limitation on the simplicity of the model and I was not able to expand the features without it becoming very computationally expensive.

8 Code

Kyle Decker

STA561 - Probabilistic Machine Learning
Kaggle Competition Writeup

Import libraries

In [2]:

```
import pandas as pd
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn import decomposition
from sklearn.linear_model import Lasso
from sklearn.svm import SVR
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
%matplotlib inline
from sklearn.linear_model import BayesianRidge
from sklearn.preprocessing import PolynomialFeatures
import numpy as np
from sklearn.neural_network import MLPRegressor
import time
import warnings
import pylab
warnings.filterwarnings('ignore')
```

Read in data and perform minor preprocessing

In [3]:

```
# Read in training and test data
train_tmp = pd.read_csv('pml_train.csv')
test_tmp = pd.read_csv('pml_test_features.csv')

# Drop the loss col temporarily in order to merge data frames
loss = train_tmp['loss']
train_tmp.drop(labels=['loss'], axis=1,inplace = True)

# Cat the training and test in order to convert categorical vars to dummies
frames = [train_tmp,test_tmp]
data_tmp = pd.concat(frames)

# Convert categorical features to dummy vars
data_dummy = pd.get_dummies(data_tmp)

# Split back in to train and test
train_size = len(train_tmp['id'])
train = data_dummy.iloc[0:train_size,:]
test = data_dummy.iloc[train_size:,:]
```

Exploratory analysis and Feature Reduction

In [4]:

```
# Get some summary stats
train.describe()
```

Out[4]:

	id	cont1	cont2	cont3	cont4	c
count	131822.00000	131822.000000	131822.000000	131822.000000	131822.000000	1
mean	65910.50000	0.493342	0.507205	0.499348	0.491129	0
std	38053.87793	0.187592	0.207240	0.202093	0.211094	0
min	0.00000	0.000016	0.001149	0.002634	0.176921	0
25%	32955.25000	0.344779	0.358319	0.336963	0.327354	0
50%	65910.50000	0.475784	0.555782	0.527991	0.452887	0
75%	98865.75000	0.623912	0.681761	0.634224	0.652072	0
max	131821.00000	0.984975	0.862654	0.944251	0.952482	0

8 rows × 1154 columns

In [5]:

```
# Drop the categorical features that have very few instances (<10000)
cols_to_drop = []
for i in range(len(train.iloc[1])):
    if (np.count_nonzero(train.iloc[:,i]) < 10000):
        cols_to_drop = np.append(cols_to_drop,i)
train.drop(train.columns[cols_to_drop.astype(int)],axis=1,inplace = True)
test.drop(test.columns[cols_to_drop.astype(int)],axis=1,inplace = True)

# Move the loss to column 0
train.insert(0, 'loss', loss)

# randomly partition data into train_sub (80%) and cross_val (20%) sets
train_sub,cross_val = train_test_split(train, test_size=.2)
```

In [ ]:

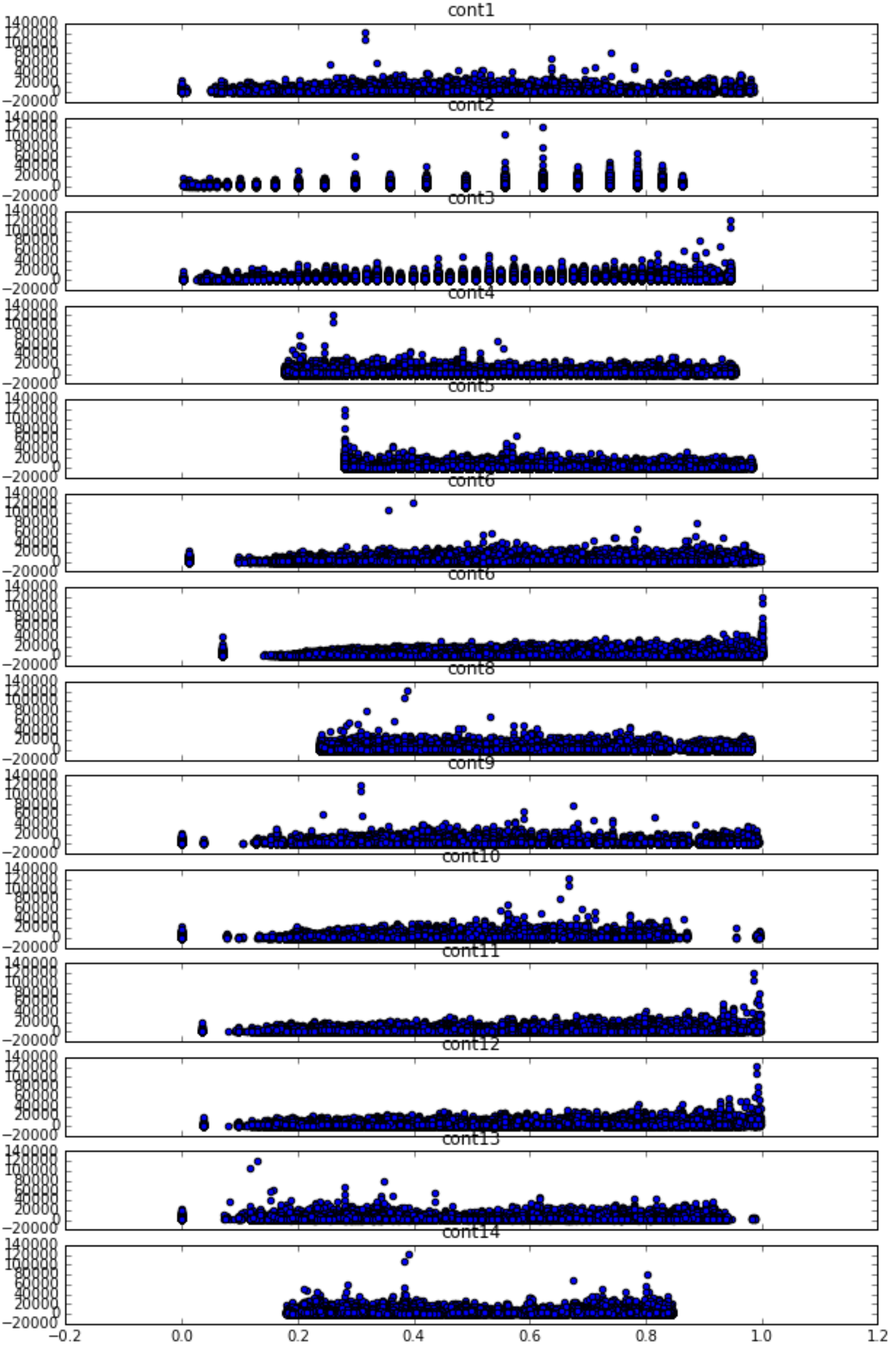
```
len(train.iloc[1])
```

In [6]:

```
# Scatter plot of continuous vars
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 10
fig_size[1] = 16
plt.rcParams["figure.figsize"] = fig_size

f, (ax1, ax2, ax3, ax4, ax5, ax6, ax7, ax8, ax9, ax10, ax11, ax12, ax13, ax14) = p
lt.subplots(14, sharex=True, sharey=True)
ax1.scatter(train['cont1'],train['loss'])
f, (ax1).title.set_text('cont1')
ax2.scatter(train['cont2'],train['loss'])
f, (ax2).title.set_text('cont2')
ax3.scatter(train['cont3'],train['loss'])
f, (ax3).title.set_text('cont3')
ax4.scatter(train['cont4'],train['loss'])
f, (ax4).title.set_text('cont4')
ax5.scatter(train['cont5'],train['loss'])
f, (ax5).title.set_text('cont5')
ax6.scatter(train['cont6'],train['loss'])
f, (ax6).title.set_text('cont6')
ax7.scatter(train['cont7'],train['loss'])
f, (ax7).title.set_text('cont6')
ax8.scatter(train['cont8'],train['loss'])
f, (ax8).title.set_text('cont8')
ax9.scatter(train['cont9'],train['loss'])
f, (ax9).title.set_text('cont9')
ax10.scatter(train['cont10'],train['loss'])
f, (ax10).title.set_text('cont10')
ax11.scatter(train['cont11'],train['loss'])
f, (ax11).title.set_text('cont11')
ax12.scatter(train['cont12'],train['loss'])
f, (ax12).title.set_text('cont12')
ax13.scatter(train['cont13'],train['loss'])
f, (ax13).title.set_text('cont13')
ax14.scatter(train['cont14'],train['loss'])
f, (ax14).title.set_text('cont14')
```

Out[6]:

```
(<matplotlib.figure.Figure at 0x11633b3c8>, None)
```

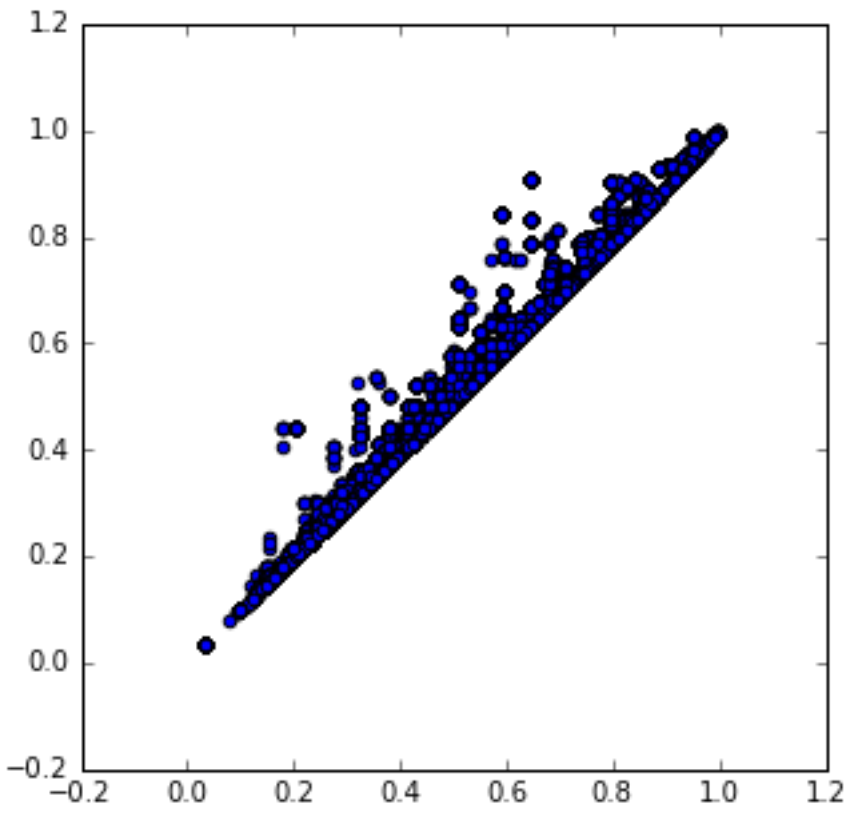

cont1
cont2
cont3
cont4
cont5
cont6
cont7
cont8
cont9
cont10
cont11
cont12
cont13
cont14
140000
120000
100000
80000
60000
40000
20000
0
−20000
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2

Some features seem highly correlated

In [7]:

```
# Plot two variables against eachother that seem correlated
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 5
fig_size[1] = 5
plt.rcParams["figure.figsize"] = fig_size
plt.scatter(train['cont11'],train['cont12'])
# Determine correlation r2 between the two variables
r2 = r2_score(train['cont11'],train['cont12'])
print('cont11 and cont12 are highly correlated with r2 score of %f' %r2)
# PCA should take care of these but lets drop one of them just in case
train.drop(labels=['cont11'], axis=1,inplace = True)
train_sub.drop(labels=['cont11'], axis=1,inplace = True)
cross_val.drop(labels=['cont11'], axis=1,inplace = True)
test.drop(labels=['cont11'], axis=1,inplace = True)
```

```
cont11 and cont12 are highly correlated with r2 score of 0.988691
```



PCA to perform dimensionality reduction and take care of correlated features

In [8]:

```
# PCA
pca= decomposition.PCA(n_components=75)
train_pca = pca.fit_transform(train.iloc[:,2:])
train_sub_pca = pca.transform(train_sub.iloc[:,2:])
cross_val_pca = pca.transform(cross_val.iloc[:,2:])
test_pca = pca.transform(test.iloc[:,1:])
```

Random Forest Model

In [9]:

```
# Random Forest Regression
#rf = RandomForestRegressor(n_estimators = 100, criterion='mse',min_samples_leaf=1
0,max_features = 100, random_state = 3)
rf = RandomForestRegressor(n_estimators = 100, criterion='mse',min_samples_split=4
0,min_samples_leaf=20,max_features = 40, random_state = 3)
#rf = RandomForestRegressor(n_estimators = 5, criterion='mse',min_samples_leaf=15,
max_features = 100)

start_time = time.time()
# train the model
rf.fit(train_sub.iloc[:,2:],train_sub.iloc[:,0])
#rf.fit(train_sub_pca,train_sub.iloc[:,0])
elapsed_time = time.time() - start_time
print('Model trained in  %f seconds' % elapsed_time)

# predict the loss for training and cross validation
loss_pred_train = rf.predict(train_sub.iloc[:,2:])
#loss_pred_train = rf.predict(train_sub_pca)
MAE_train = mean_absolute_error(train_sub.iloc[:,0],loss_pred_train)
print('MAE on training set = %f' % MAE_train)

loss_pred_cv = rf.predict(cross_val.iloc[:,2:])
#loss_pred_cv = rf.predict(cross_val_pca)
MAE_cv = mean_absolute_error(cross_val.iloc[:,0],loss_pred_cv)
print('MAE on cross validation set = %f' % MAE_cv)
```

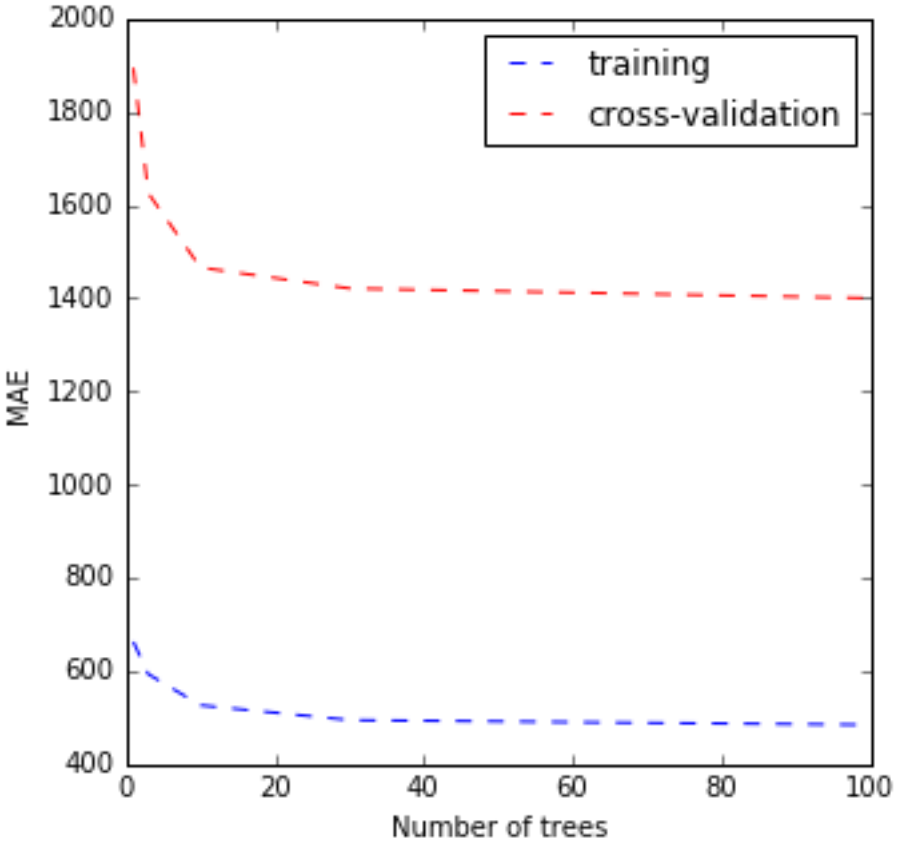
```
Model trained in  49.237059 seconds
MAE on training set = 1146.780570
MAE on cross validation set = 1249.461602
```

In [10]:

```
# Choose Number of Trees
trees = [1,3,10,30,100]
train_error_all = []
cv_error_all = []
size = 1000;
for i in range(len(trees)):
    rf = RandomForestRegressor(n_estimators = trees[i], random_state = 3)
    rf.fit(train_sub.iloc[:size,2:],train_sub.iloc[:size,0])
    loss_pred_train = rf.predict(train_sub.iloc[:size,2:])
    MAE_train = mean_absolute_error(train_sub.iloc[:size,0],loss_pred_train)
    loss_pred_cv = rf.predict(cross_val.iloc[:,2:])
    MAE_cv = mean_absolute_error(cross_val.iloc[:,0],loss_pred_cv)

    train_error_all = np.append(train_error_all,MAE_train)
    cv_error_all = np.append(cv_error_all,MAE_cv)

pylab.plot(trees,train_error_all,'b--',label='training')
pylab.plot(trees,cv_error_all,'r--',label='cross-validation')
pylab.legend(loc='upper right')
pylab.xlabel('Number of trees')
pylab.ylabel('MAE')
```

Out[10]:

```
<matplotlib.text.Text at 0x118fdd0f0>
```

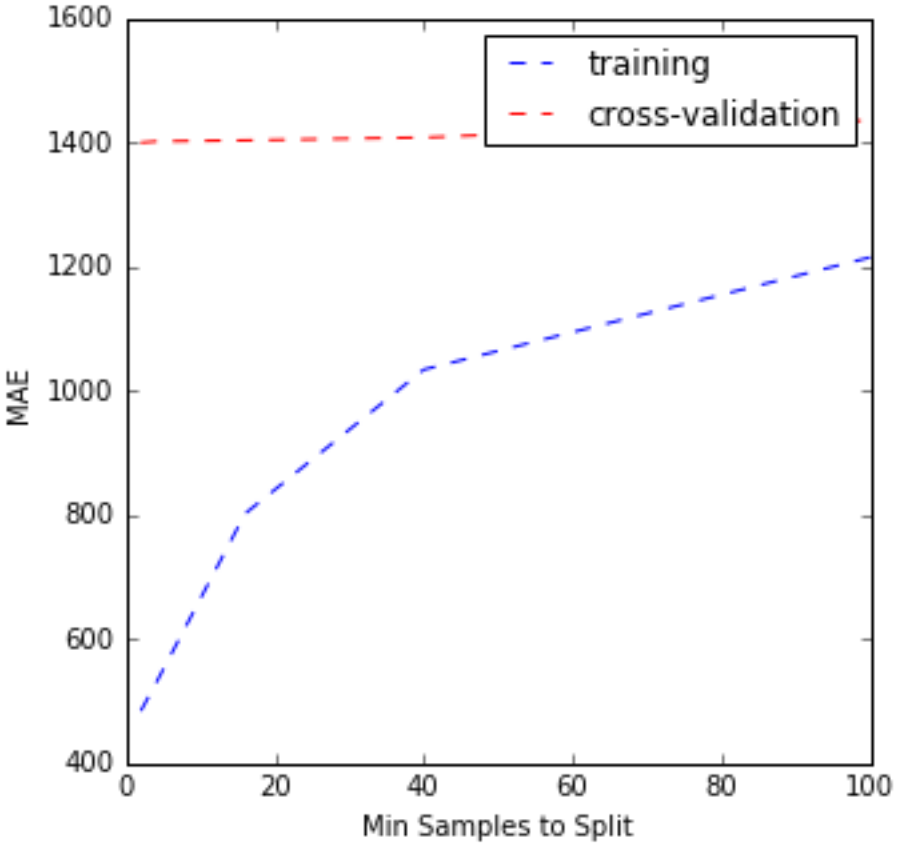


In [11]:

```
# Choose Min Samples to Split
splits = [2,4,16,40,100]
train_error_all = []
cv_error_all = []
size = 1000;
for i in range(len(trees)):
    rf = RandomForestRegressor(n_estimators = 100, min_samples_split=splits[i], ra
ndom_state = 3)
    rf.fit(train_sub.iloc[:size,2:],train_sub.iloc[:size,0])
    loss_pred_train = rf.predict(train_sub.iloc[:size,2:])
    MAE_train = mean_absolute_error(train_sub.iloc[:size,0],loss_pred_train)
    loss_pred_cv = rf.predict(cross_val.iloc[:,2:])
    MAE_cv = mean_absolute_error(cross_val.iloc[:,0],loss_pred_cv)

    train_error_all = np.append(train_error_all,MAE_train)
    cv_error_all = np.append(cv_error_all,MAE_cv)

pylab.plot(splits,train_error_all,'b--',label='training')
pylab.plot(splits,cv_error_all,'r--',label='cross-validation')
pylab.legend(loc='upper right')
pylab.xlabel('Min Samples to Split')
pylab.ylabel('MAE')
```

Out[11]:

```
<matplotlib.text.Text at 0x1162bdfd0>
```

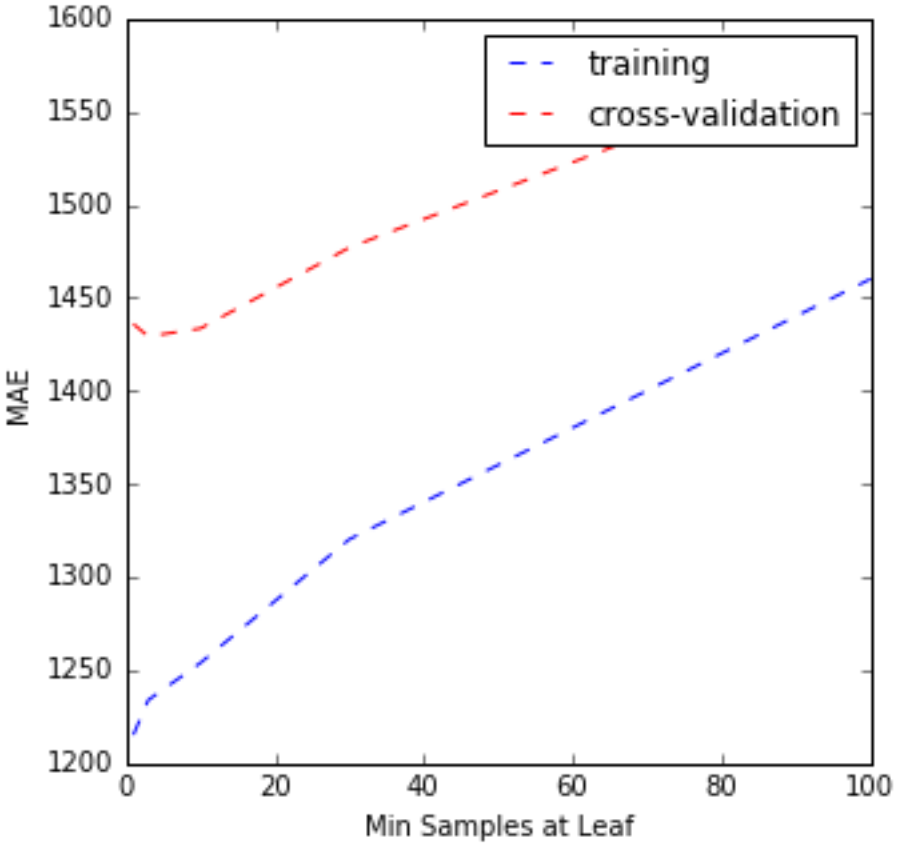


In [12]:

```
# Choose Min Samples at Leaf
leaf = [1,3,10,30,100]
train_error_all = []
cv_error_all = []
size = 1000;
for i in range(len(trees)):
    rf = RandomForestRegressor(n_estimators = 100, min_samples_split=100,min_sampl
es_leaf=leaf[i], random_state = 3)
    rf.fit(train_sub.iloc[:size,2:],train_sub.iloc[:size,0])
    loss_pred_train = rf.predict(train_sub.iloc[:size,2:])
    MAE_train = mean_absolute_error(train_sub.iloc[:size,0],loss_pred_train)
    loss_pred_cv = rf.predict(cross_val.iloc[:,2:])
    MAE_cv = mean_absolute_error(cross_val.iloc[:,0],loss_pred_cv)

    train_error_all = np.append(train_error_all,MAE_train)
    cv_error_all = np.append(cv_error_all,MAE_cv)

pylab.plot(leaf,train_error_all,'b--',label='training')
pylab.plot(leaf,cv_error_all,'r--',label='cross-validation')
pylab.legend(loc='upper right')
pylab.xlabel('Min Samples at Leaf')
pylab.ylabel('MAE')
```

Out[12]:

```
<matplotlib.text.Text at 0x116774080>
```

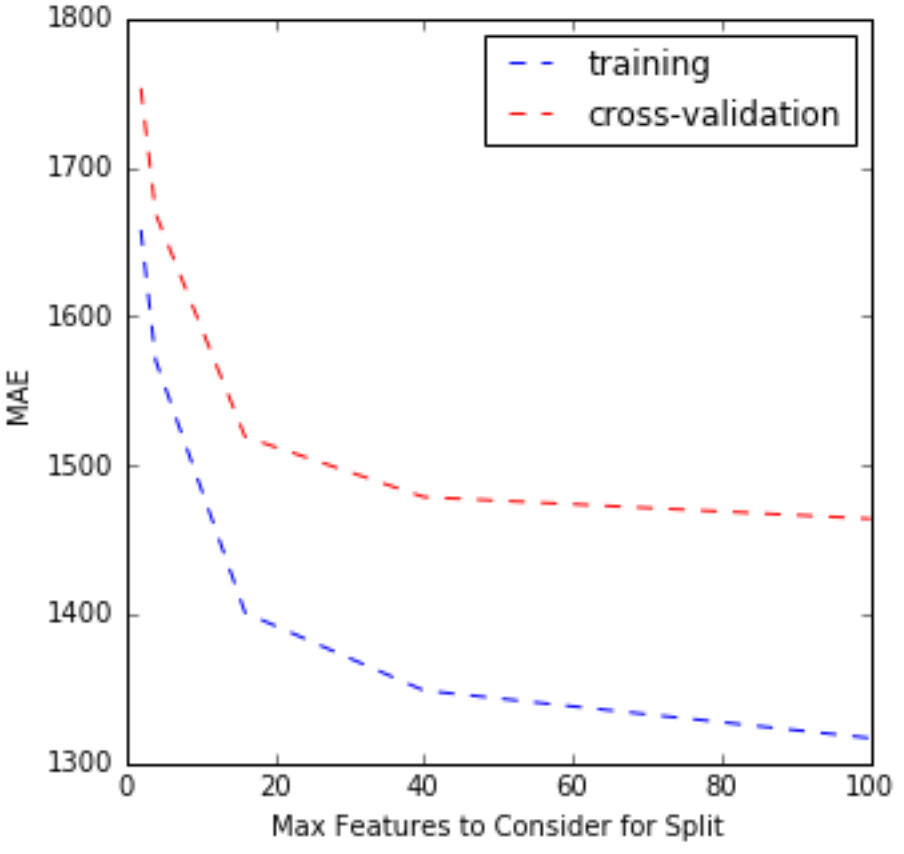


In [13]:

```
# Choose Max Features to consider
feat = [2,4,16,40,100]
train_error_all = []
cv_error_all = []
size = 1000;
for i in range(len(trees)):
    rf = RandomForestRegressor(n_estimators = 100, min_samples_split=100,min_sampl
es_leaf=20,max_features = feat[i], random_state = 3)
    rf.fit(train_sub.iloc[:size,2:],train_sub.iloc[:size,0])
    loss_pred_train = rf.predict(train_sub.iloc[:size,2:])
    MAE_train = mean_absolute_error(train_sub.iloc[:size,0],loss_pred_train)
    loss_pred_cv = rf.predict(cross_val.iloc[:,2:])
    MAE_cv = mean_absolute_error(cross_val.iloc[:,0],loss_pred_cv)

    train_error_all = np.append(train_error_all,MAE_train)
    cv_error_all = np.append(cv_error_all,MAE_cv)

pylab.plot(feat,train_error_all,'b--',label='training')
pylab.plot(feat,cv_error_all,'r--',label='cross-validation')
pylab.legend(loc='upper right')
pylab.xlabel('Max Features to Consider for Split')
pylab.ylabel('MAE')
```

Out[13]:

```
<matplotlib.text.Text at 0x1129a5cf8>
```



In [ ]:

```
# Try on the test set and make kaggle output
loss_pred_test = rf.predict(test.iloc[:,1:])
#loss_pred_test = rf.predict(test_pca)

test_results_id = test['id']
test_results_loss = loss_pred_test

# Create results data frame
test_df = pd.DataFrame(data=test_results_id)
test_df.insert(1, 'loss', test_results_loss)

# Save output to csv format
test_df.to_csv('RF_results.csv',index = False)
```

Lasso Model

In [14]:

```
# Lasso Regression
lasso = Lasso(alpha=0.1);

pca_bool = True;

x_train = train_sub_pca
x_cv = cross_val_pca

if (pca_bool == True):
    lasso.fit(x_train,train_sub.iloc[:,0])
    loss_pred_train = lasso.predict(x_train)
    loss_pred_cv = lasso.predict(x_cv)
else:
    lasso.fit(train_sub.iloc[:,2:],train_sub.iloc[:,0])
    loss_pred_train = lasso.predict(train_sub.iloc[:,2:])
    loss_pred_cv = lasso.predict(cross_val.iloc[:,2:])

# predict the loss for training and cross validation
MAE_train = mean_absolute_error(train_sub.iloc[:,0],loss_pred_train)
print('MAE on training set = %f' % MAE_train)

MAE_cv = mean_absolute_error(cross_val.iloc[:,0],loss_pred_cv)
print('MAE on cross validation set = %f' % MAE_cv)
```

```
MAE on training set = 1359.122693
MAE on cross validation set = 1364.807023
```

In [ ]:

```
# Try on the test set and make kaggle output
loss_pred_test = lasso.predict(test_pca)

test_results_id = test['id']
test_results_loss = loss_pred_test

# Create results data frame
test_df = pd.DataFrame(data=test_results_id)
test_df.insert(1, 'loss', test_results_loss)

# Save output to csv format
test_df.to_csv('LASSO_results.csv',index = False)
```

Support Vector Model

In [ ]:

```
# Support Vector Regression
svr = SVR(kernel='rbf', epsilon = 0.1);
#svr = SVR(kernel='poly', degree=2);
size = 100;
svr.fit(train_sub_pca[0:size,:],train_sub.iloc[0:size,0])

# predict the loss for training and cross validation
#loss_pred_train = rf.predict(train_sub.iloc[:,2:])
loss_pred_train = svr.predict(train_sub_pca)
MAE_train = mean_absolute_error(train_sub.iloc[:,0],loss_pred_train)
print('MAE on training set = %f' % MAE_train)

#loss_pred_cv = rf.predict(cross_val.iloc[:,2:])
loss_pred_cv = svr.predict(cross_val_pca)
MAE_cv = mean_absolute_error(cross_val.iloc[:,0],loss_pred_cv)
print('MAE on cross validation set = %f' % MAE_cv)
```

Bayesian Model

In [ ]:

```
poly = PolynomialFeatures(degree=2,interaction_only=True)
x = poly.fit_transform(train_sub_pca)
x.shape
```

In [ ]:

```
# Bayesian regression w/ poly features
br = BayesianRidge();

x_poly = poly.transform(train_sub_pca)
cv_poly = poly.transform(cross_val_pca)

br.fit(x_poly,train_sub.iloc[:,0])

# predict the loss for training and cross validation
#loss_pred_train = br.predict(train_sub_pca)
loss_pred_train = br.predict(x_poly)
MAE_train = mean_absolute_error(train_sub.iloc[:,0],loss_pred_train)
print('MAE on training set = %f' % MAE_train)

#loss_pred_cv = br.predict(cross_val_pca)
loss_pred_cv = br.predict(cv_poly)
MAE_cv = mean_absolute_error(cross_val.iloc[:,0],loss_pred_cv)
print('MAE on cross validation set = %f' % MAE_cv)
```

In [ ]:

```
# Try on the test set and make kaggle output
loss_pred_test = br.predict(poly.transform(test_pca))

test_results_id = test['id']
test_results_loss = loss_pred_test

# Create results data frame
test_df = pd.DataFrame(data=test_results_id)
test_df.insert(1, 'loss', test_results_loss)

# Save output to csv format
test_df.to_csv('Poly_Bayesian_results.csv',index = False)
```

Multi-layer Perceptron

In [15]:

```
# MLP Regressor
#nn = MLPRegressor(hidden_layer_sizes=(100, 100, 100, 100, 100, 100, 100, 100, 100
, 100), activation = 'relu',random_state = 13)
nn = MLPRegressor(hidden_layer_sizes=(30, 30, 30,), activation = 'relu',random_sta
te = 2)

start_time = time.time()
# train the model
nn.fit(train_sub.iloc[0:,2:],train_sub.iloc[0:,0])

elapsed_time = time.time() - start_time
print('Model trained in  %f seconds' % elapsed_time)

# predict the loss for training and cross validation
loss_pred_train = nn.predict(train_sub.iloc[:,2:])

MAE_train = mean_absolute_error(train_sub.iloc[:,0],loss_pred_train)
print('MAE on training set = %f' % MAE_train)

loss_pred_cv = nn.predict(cross_val.iloc[:,2:])


MAE_cv = mean_absolute_error(cross_val.iloc[:,0],loss_pred_cv)
print('MAE on cross validation set = %f' % MAE_cv)
```

```
Model trained in  54.765320 seconds
MAE on training set = 1242.430098
MAE on cross validation set = 1257.990008
```

In [16]:

```
# Choose Activation Function
act = ['identity', 'logistic', 'tanh', 'relu']
train_error_all = []
cv_error_all = []
size = 1000;
for i in range(len(act)):
    nn = MLPRegressor(activation = act[i])
    nn.fit(train_sub.iloc[:size,2:],train_sub.iloc[:size,0])
    loss_pred_train = nn.predict(train_sub.iloc[:size,2:])
    MAE_train = mean_absolute_error(train_sub.iloc[:size,0],loss_pred_train)
    loss_pred_cv = nn.predict(cross_val.iloc[:,2:])
    MAE_cv = mean_absolute_error(cross_val.iloc[:,0],loss_pred_cv)

    train_error_all = np.append(train_error_all,MAE_train)
    cv_error_all = np.append(cv_error_all,MAE_cv)

print(cv_error_all)
print(train_error_all)
```

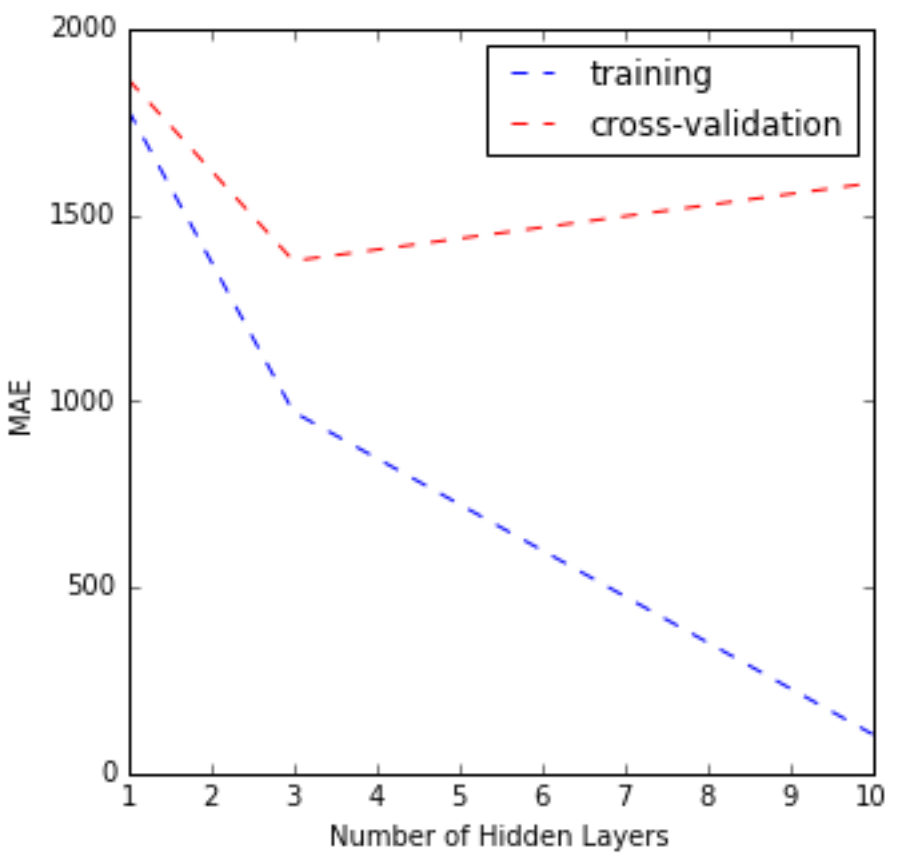
```
[ 1824.86250227  2940.84139112  2909.51261542  1868.47162348]
[ 1741.52089315  2802.98512511  2771.7013301   1786.74260806]
```

In [18]:

```
# Choose Number of Hidden Layers
num_layers = [1,3,10]

train_error_all = []
cv_error_all = []
size = 1000;
for i in range(len(num_layers)):
    if (i ==0):
        nn = MLPRegressor(activation = 'relu', hidden_layer_sizes=(100,))
    elif (i == 1):
        nn = MLPRegressor(activation = 'relu', hidden_layer_sizes=(100,100,100,))
    elif (i == 2):
        nn = MLPRegressor(activation = 'relu', hidden_layer_sizes=(100,100,100,100
,100,100,100,100,100,100,))
    nn.fit(train_sub.iloc[:size,2:],train_sub.iloc[:size,0])
    loss_pred_train = nn.predict(train_sub.iloc[:size,2:])
    MAE_train = mean_absolute_error(train_sub.iloc[:size,0],loss_pred_train)
    loss_pred_cv = nn.predict(cross_val.iloc[:,2:])
    MAE_cv = mean_absolute_error(cross_val.iloc[:,0],loss_pred_cv)

    train_error_all = np.append(train_error_all,MAE_train)
    cv_error_all = np.append(cv_error_all,MAE_cv)

pylab.plot(num_layers,train_error_all,'b--',label='training')
pylab.plot(num_layers,cv_error_all,'r--',label='cross-validation')
pylab.legend(loc='upper right')
pylab.xlabel('Number of Hidden Layers')
pylab.ylabel('MAE')
```

Out[18]:

```
<matplotlib.text.Text at 0x122b77ac8>
```

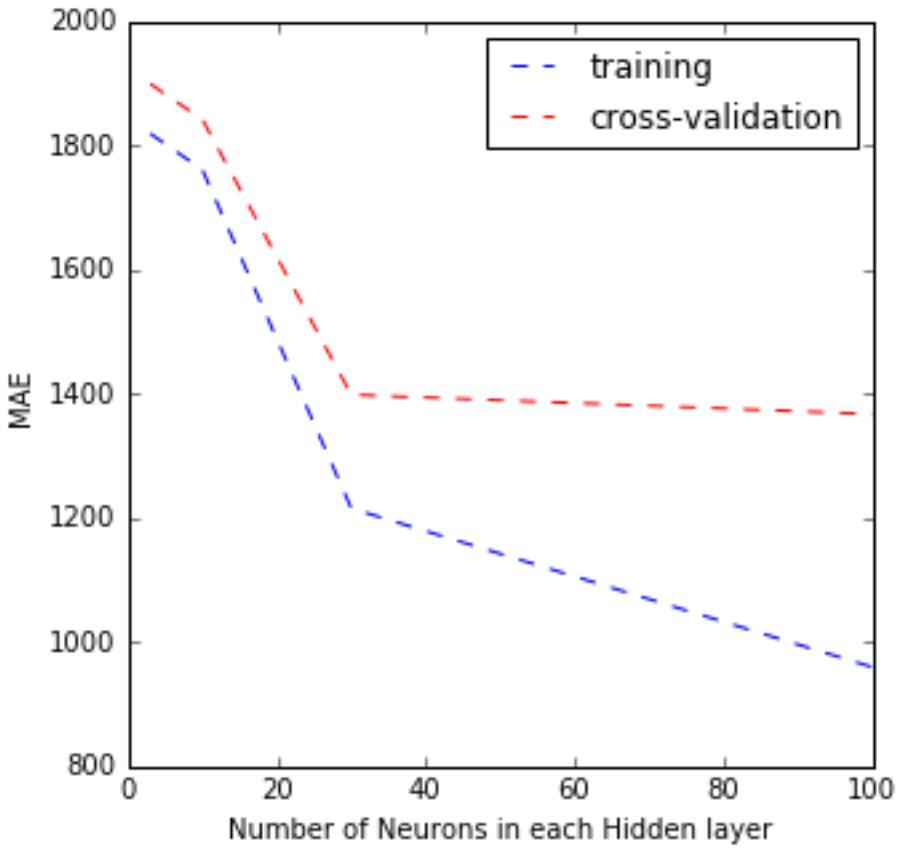


In [19]:

```
# Choose Hidden Layer Size
layer_size = [3,10,30,100]

train_error_all = []
cv_error_all = []
size = 1000;
for i in range(len(layer_size)):
    nn = MLPRegressor(activation = 'relu', hidden_layer_sizes=(layer_size[i],layer
_size[i],layer_size[i],))
    nn.fit(train_sub.iloc[:size,2:],train_sub.iloc[:size,0])
    loss_pred_train = nn.predict(train_sub.iloc[:size,2:])
    MAE_train = mean_absolute_error(train_sub.iloc[:size,0],loss_pred_train)
    loss_pred_cv = nn.predict(cross_val.iloc[:,2:])
    MAE_cv = mean_absolute_error(cross_val.iloc[:,0],loss_pred_cv)

    train_error_all = np.append(train_error_all,MAE_train)
    cv_error_all = np.append(cv_error_all,MAE_cv)

pylab.plot(layer_size,train_error_all,'b--',label='training')
pylab.plot(layer_size,cv_error_all,'r--',label='cross-validation')
pylab.legend(loc='upper right')
pylab.xlabel('Number of Neurons in each Hidden layer')
pylab.ylabel('MAE')
```

Out[19]:

```
<matplotlib.text.Text at 0x117782ba8>
```



In [ ]:

```
# Try on the test set and make kaggle output
loss_pred_test = nn.predict(test.iloc[:,1:])
#loss_pred_test = rf.predict(test_pca)

test_results_id = test['id']
test_results_loss = loss_pred_test

# Create results data frame
test_df = pd.DataFrame(data=test_results_id)
test_df.insert(1, 'loss', test_results_loss)

# Save output to csv format
test_df.to_csv('NN_results.csv',index = False)
```

Combine MLP and RF

In [ ]:

```
# Combine RF and MLP
loss_pred_cv_rf = rf.predict(cross_val.iloc[:,2:])
loss_pred_cv_nn = nn.predict(cross_val.iloc[:,2:])
loss_pred_cv = (loss_pred_cv_rf + loss_pred_cv_nn)/2
#loss_pred_cv = rf.predict(cross_val_pca)
MAE_cv = mean_absolute_error(cross_val.iloc[:,0],loss_pred_cv)
print('MAE on cross validation set = %f' % MAE_cv)
```

In [ ]:

```
# Try on the test set and make kaggle output
loss_pred_test = (nn.predict(test.iloc[:,1:]) + rf.predict(test.iloc[:,1:]))/2
#loss_pred_test = rf.predict(test_pca)

test_results_id = test['id']
test_results_loss = loss_pred_test

# Create results data frame
test_df = pd.DataFrame(data=test_results_id)
test_df.insert(1, 'loss', test_results_loss)

# Save output to csv format
test_df.to_csv('NN_RF_combo_results.csv',index = False)
```