




KAGGLE COMPETITION REPORT

ALL STATE INSURANCE CLAIMS

NIRAL SHAH

PROFESSOR RUDIN- MACHINE LEARNING COURSE

Duke University



I. Exploratory Analysis

After loading the data, it became evident that there were 116 categorical features and 14 continuous features. This makes developing a model more complicated. After a quick check, fortunately there was no missing data. So their first goal was to figure out if the continuous data held any useful information, could be formatted in a nice way or could be simply thrown out.

So first the continuous data was analyzed:

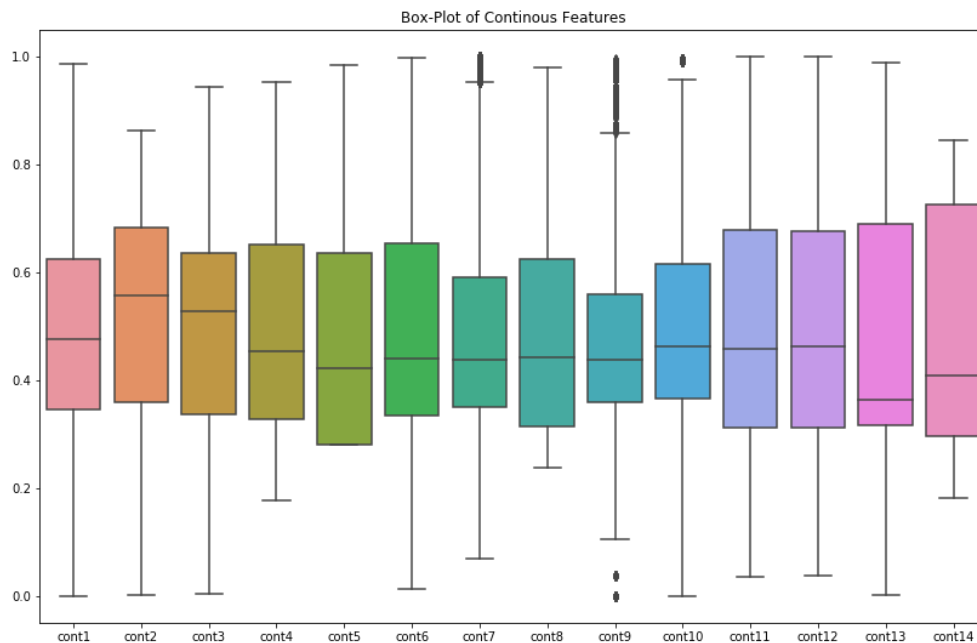


Figure 1

Looking at this, we can see that most of the features are not skewed and thus normalization is not very necessary. However, as a step we could try normalizing the variables that are very skewed like cont13 and cont14.

(See next page for continued exploratory analysis):

Let's take a look at the relationships between variables deeper to understand the relationship between the continuous variables, a correlation matrix was made to make it easier to visualize.

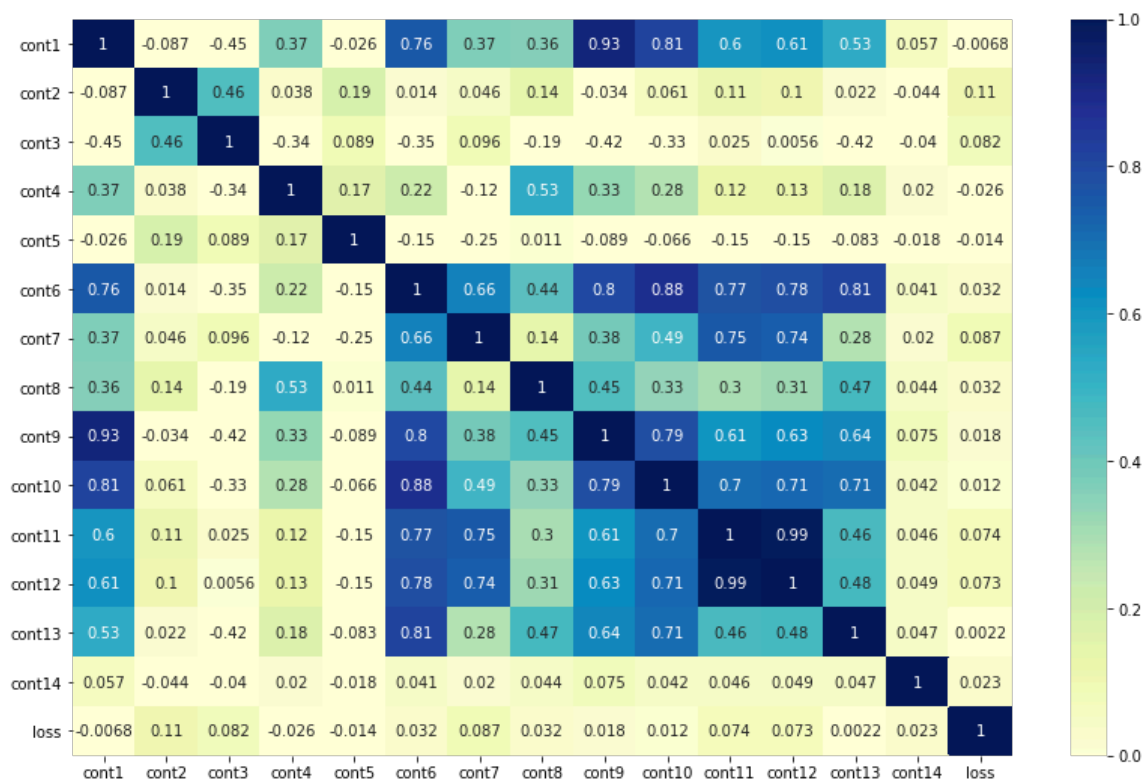


Figure 2

From this correlation matrix it is evident that some variables are highly correlated like cont1 and cont9 as well as cont11 and cont12, similarly cont12 and cont11. Thus, of these pairs, one of each can likely be thrown out. Thereby reducing the dimension, without losing much information.

Next let's take a look at the feature we are trying to predict, Loss. Plotting a distribution plot as well as a violin plot, it appears the loss column is extremely right skewed:

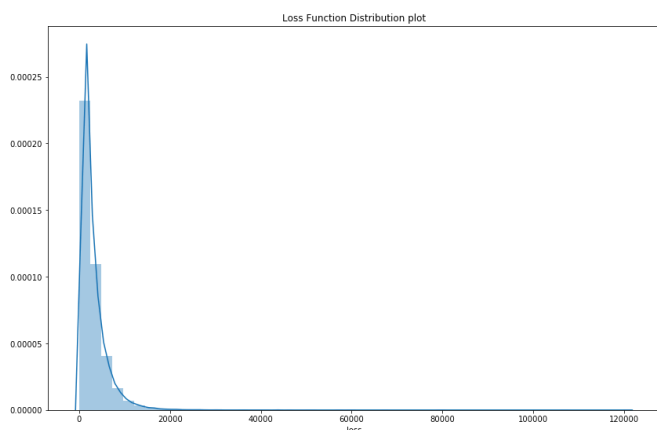


Fig 3. Skewed Loss Distribution

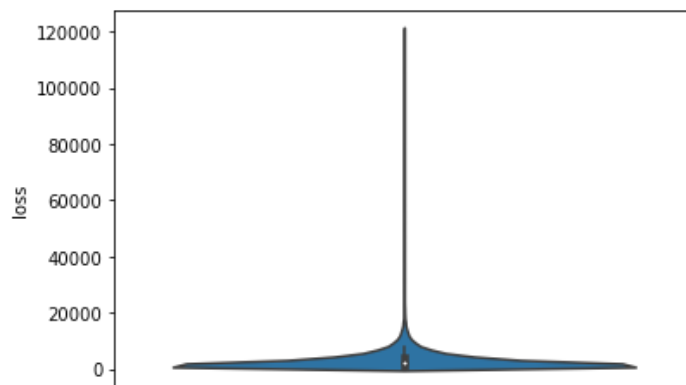


Fig 4. Violin Plot of Loss

This extreme right skew is due to the apparently many outliers. We could normalize loss by taking the log of it. Upon some research it was found that taking the $\log(1+p)$ works better for loss values that are very small. This resulted in a better loss distribution:

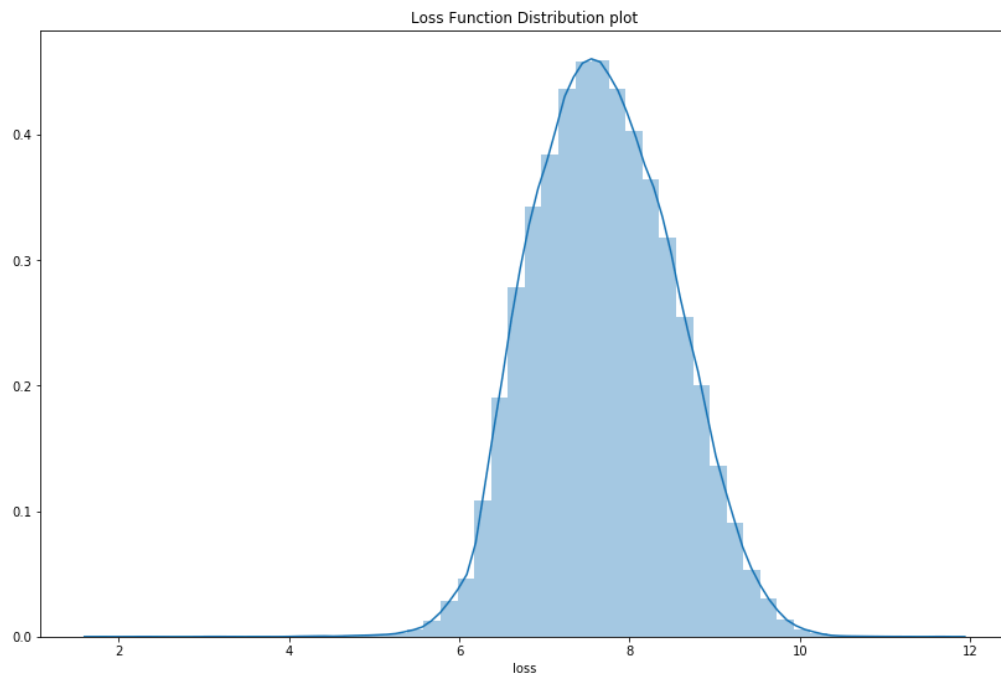


Figure 5

Categorical Variables:

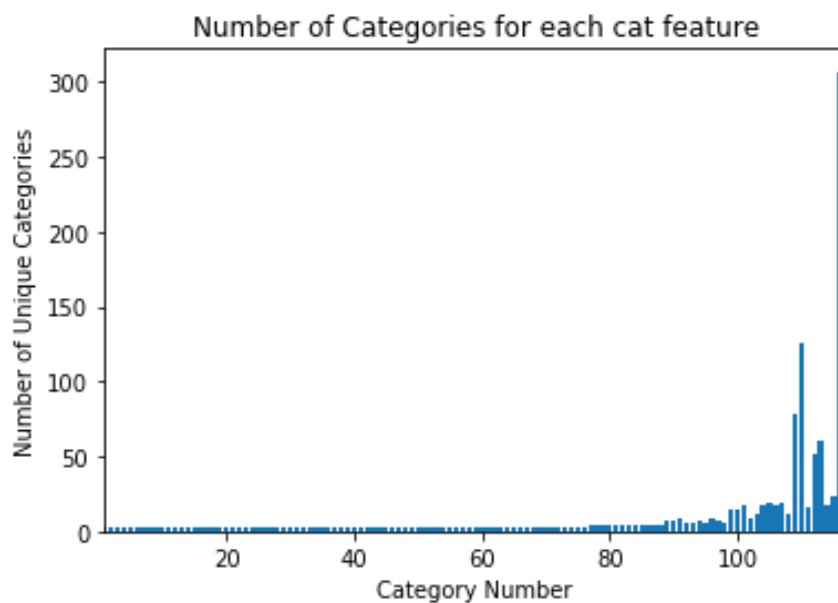


Figure 6

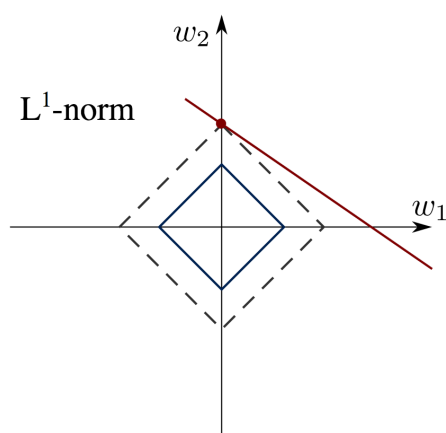
The above plot highlights how many unique categories exist for each “cat” feature. Out of the 116 categorical features the majority of them have only 2 categories, while a select few have 10

or more categories. Due to the varying number of categories for each feature, a one hot encoding scheme was attempted to make the variables numerical.

II. Models

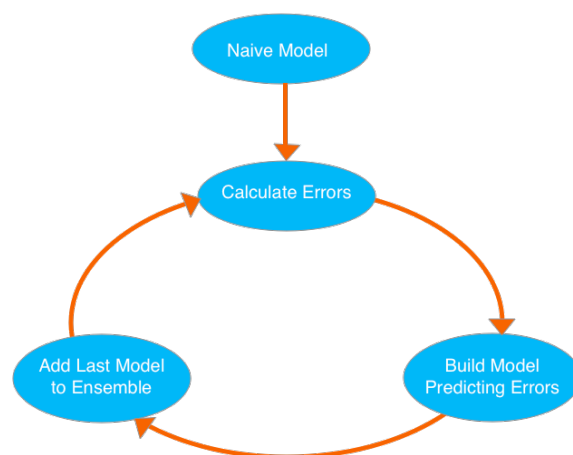
The Mean Absolute Error is given by:

$$\text{MAE} = \frac{\sum_{i=1}^n |y_i - x_i|}{n} = \frac{\sum_{i=1}^n |e_i|}{n}.$$



(Figure 8)

Lasso Regression – L1 Norm (Fig 9)



XGBoost Process (Fig 10)

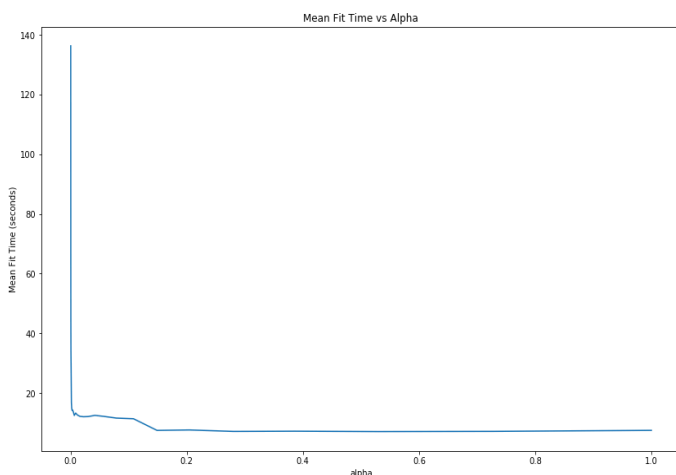
In this section, two algorithms were developed and compared to complete a regression over the loss, LASSO and XGBoost (Boosted Decision Trees). The goal was to minimize the mean absolute error (8). I wanted to compare these two algorithms since one algorithm is a linear model while the other is a nonlinear (ensemble method). Collectively I felt this would produce interesting results and provide for a nice comparison. Going into the challenge, due to the decision-making process behind an insurance claim, some type of decision tree seemed to be an intuitive choice.

As a linear regressor, LASSO was chosen due to its tendency to sparsity, thereby completing feature selection. Due to the size of the data and to the computational power available, using a recursive feature selection would be too time consuming. Due to the sharp corners of the L1 Norm (9), some of the less useful variables would be zeroed out, reducing dimensionality as well as minimizing the chance of over-fitting on the training data. Though in practice, LASSO performed reasonable, this was primarily used to set a baseline for expectations of performance for a better suited (non-linear) regressor. This would help provide a baseline for expectations of performance. Due to the number of features (>100) and a mix of continuous and discrete features, I did not expect the data to be linearly separable.

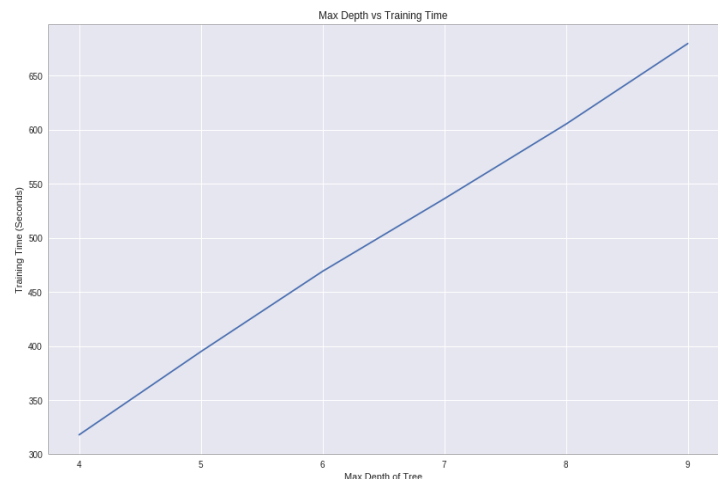
Subsequently, following intuition that a decision-making process like insurance claims would follow a decision tree like structure, I initially decided to focus on a Random-Forest-Regressors using CART Decision Tree's as stumps. However soon it became evident that this would be computationally difficult and too time consuming. CART Decision trees are built to minimize the Gini index, however even using Decision Stumps the Random-Forest-Regressor model took an intractable amount of time to minimize the objective function: *mean absolute error*. As a result, a similar but more computational efficient model was used, XGBoost or Extreme Gradient Boosting. XGBoost builds several decision trees like a random forest and then measures how well each tree is structured (similar to an impurity measure). Then it attempts to fit another weak learner to the error, attempting to minimize the error residual. Once this is completed its added to the model, making the original model slightly more complex and better fitted to the data. This iterative process in which during each cycle the model becomes more complex is represented by figure 10.

- LASSO Algorithm:
 - Implemented in python using sci-kit learn
 - http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html
- XGBoost Algorithm:
 - Python Library by dmlc
 - <http://xgboost.readthedocs.io/en/latest/model.html>
 - Figure 9: <https://www.kaggle.com/dansbecker/learning-to-use-xgboost>
- Mean Absolute Error:
 - https://en.wikipedia.org/wiki/Mean_absolute_error

III. Training:



Lasso Parameter Tuning (alpha) – Mean Fit Time (Fig 11)



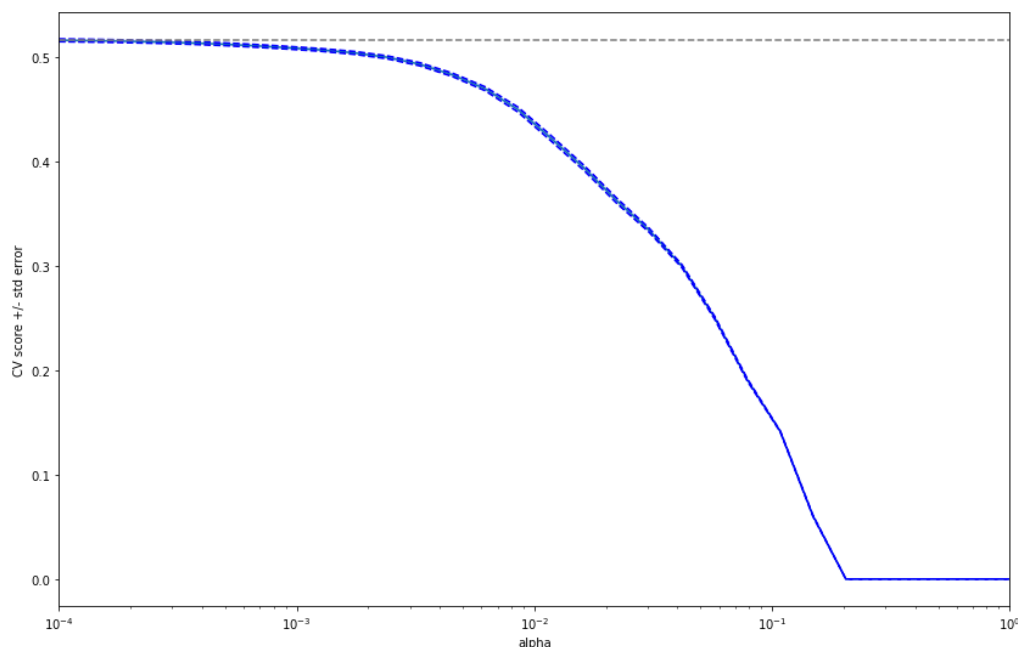
XGBoost Parameter Tuning (Max Depth)- Mean Fit Time (Fig 12)

Both algorithms do not have a closed form solution and the objective/loss function, in this case Mean Absolute Error, needs to be minimized using some type of descent algorithm. The sci-kit learn implementation of lasso utilizes coordinate descent. Coordinate descent is similar to gradient descent, except we move along one coordinate at a time. Each time choosing a coordinate with the highest gradient. This can often perform faster than gradient descent at minimizing the error. In contrast XGBoost uses gradient descent to minimize the objective function. Though similar to coordinate descent it performs in an iterative process, designing one decision tree at a time and each time taking a step of gradient descent to minimize the loss function.

Though both perform relatively quickly computations; as evident from the above graphs XGBoost takes a considerably longer time to run in general and this grows linearly as the max depth of the tree increases. This trend is similar in LASSO as the regularized term increases from zero enabling more error in the fitted model. Though overall Lasso at its worst takes only 130 seconds to train while XGBoost takes 300 seconds.

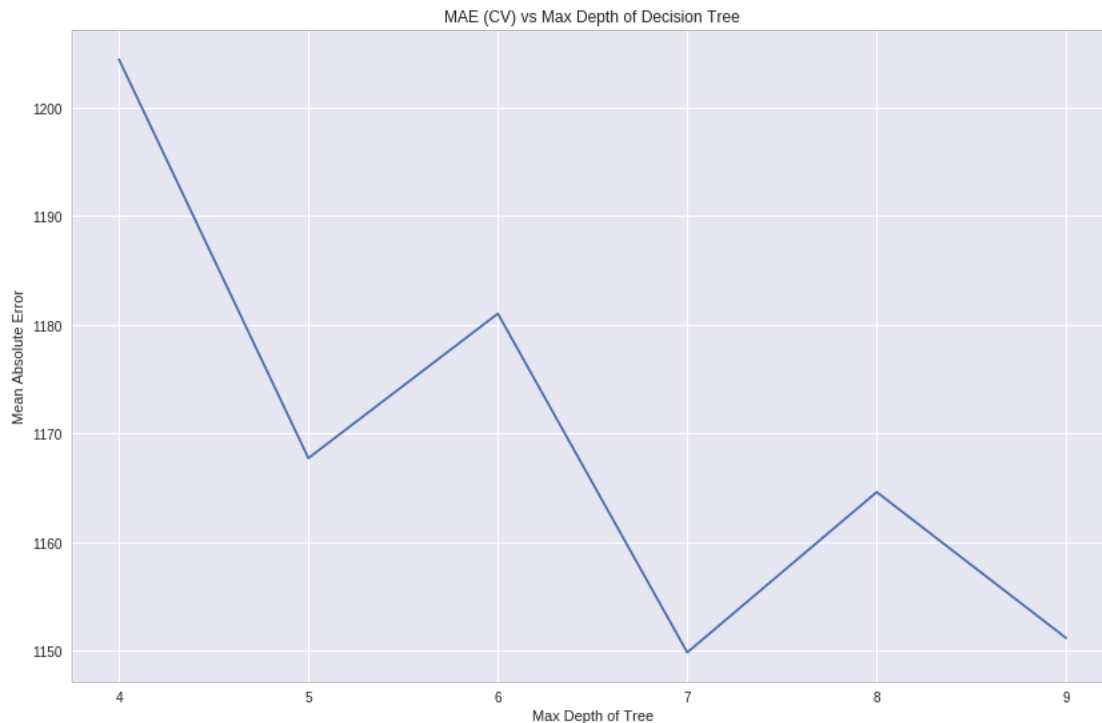
IV. [Hyperparameter Selection:](#)

Lasso: Mean Validation Error vs Alpha Parameter Tuning



To tune the alpha parameter three-fold nested cross validation was used. This means for each iteration of alpha (thirty alphas were tried), 9 fits occurred where at any given time two folds belonged to the train and validation set respectively and a third belonged to the test set. Nested Cross Validation is useful for hyper-parameter tuning like this. As is evident from the graph above after an alpha of 0.9, it achieved perfect classification on the validation set. Though this is partially indicative of potential overfitting.

XGBoost: Mean Absolute Error vs Max Depth of Decision Tree



For XGBoost ten-fold cross validation was used to do hyper parameter tuning. The max depth of a decision tree was measured from a max depth of four to a max depth of nine. Increasing the depth of the decision tree each time increases the chance of overfitting and thus this was kept in mind when choosing a parameter. Though this chart showed an interesting result that the mean absolute error significantly dropped at a max depth of seven and subsequently increased for a max depth of eight and nine.

V. Data Splits:

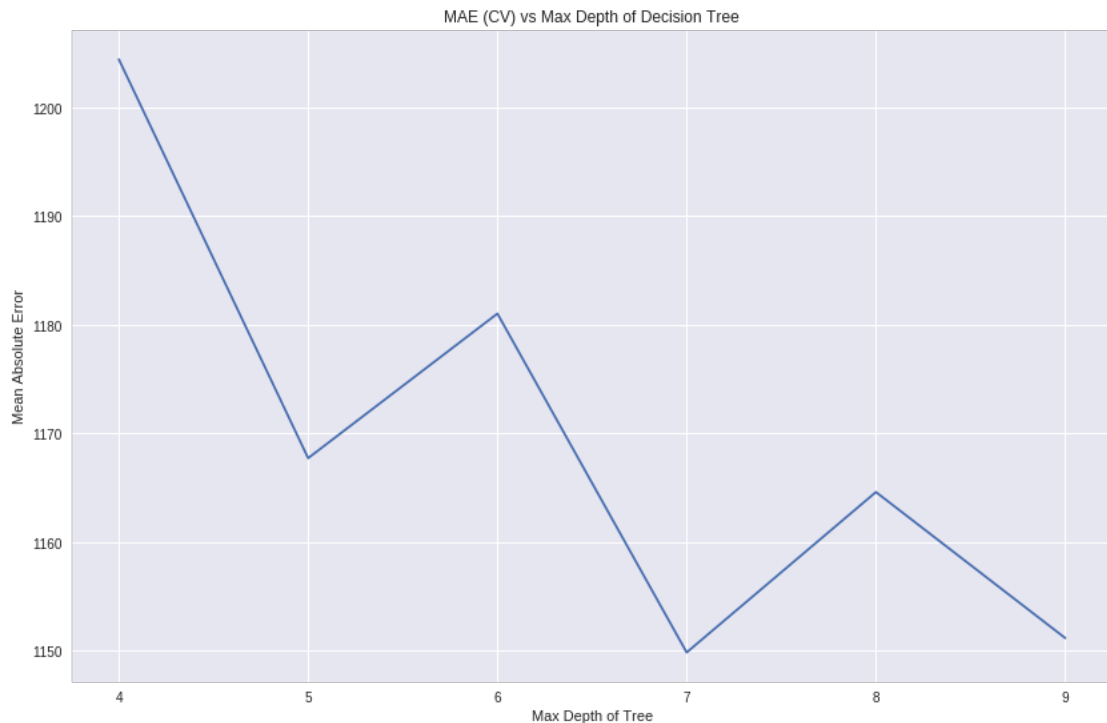
As discussed in the previous section for each choice of hyper parameter tuning, nested cross validation was completed. This means that for each run of k-folds cross validation another subset of the training split was used as a test fold. This ensured the average error measured would be more realistic.

VI. Errors & Mistakes:

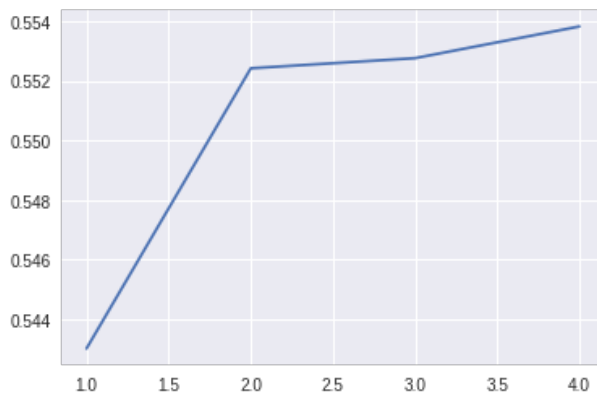
A major error I kept making at the beginning of the competition was forgetting the fact that to normalize the loss I applied the log function to it. This means that upon measuring my mean absolute error on the validation set I would need to convert it back to normal loss predictions to get a proper idea of how the model would perform and prepare it for submission. I found the feature engineering aspect of the competition to be the most difficult part. As an ECE major, some of the thought prices in looking at the statistics of the data is unintuitive to me. For example I didn't realize that taking the log loss of a function would normalize the

loss or that log would help reduce skew in certain features. I eventually realized this by attempting to learn from the mistakes of previous submissions.

VII. Predictive Accuracy:



In the end a XGBoost model with a max depth of seven was found to be the optimal model. In the competition submission this resulted in a score of 1148.91696. As you can see this nicely matched up with the results from cross validation, indicating there's not much if any overfitting occurring.



This graph on the right show's cross validated accuracy on the four folds.

VIII. Code: (see attached)

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

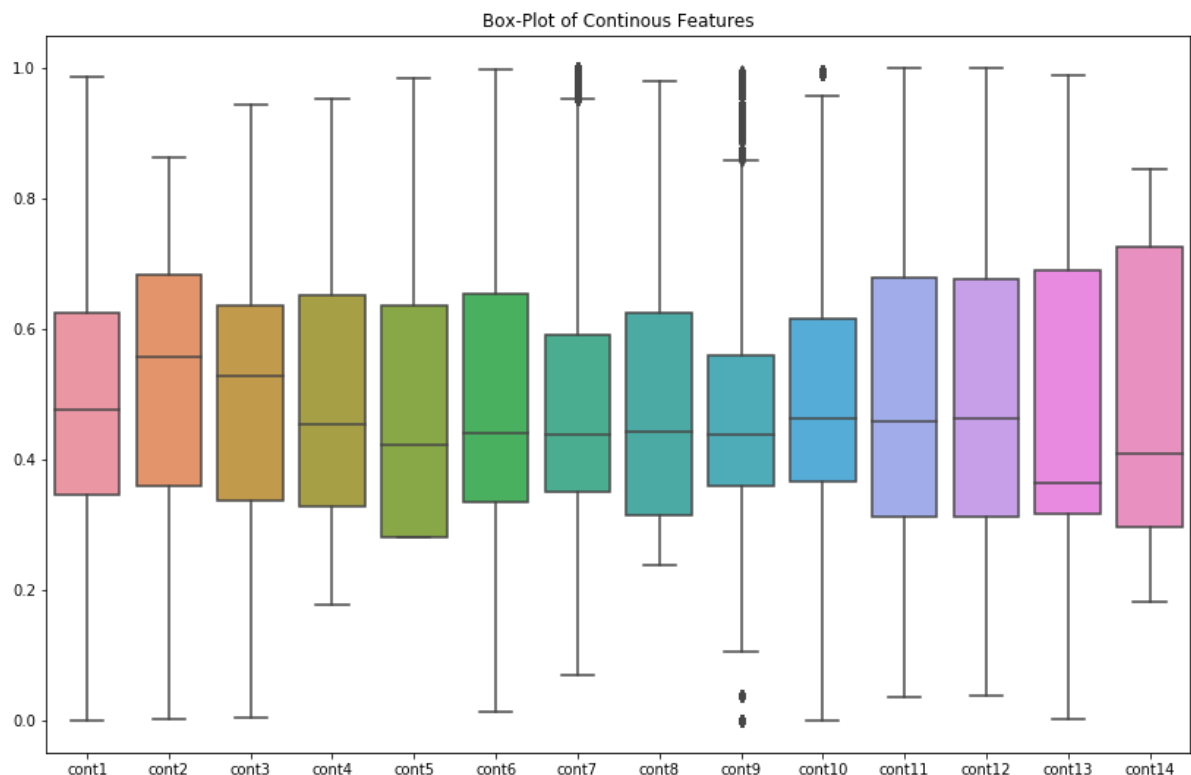
```
In [2]: df_train = pd.read_csv('pml_train.csv')
df_test = pd.read_csv('pml_test_features.csv')
```

```
In [3]: cont_features = []
cat_features = []
for c in df_train.columns:
    if 'cont' in c:
        cont_features.append(c)
    elif 'cat' in c:
        cat_features.append(c)
```

Exploratory Analysis:

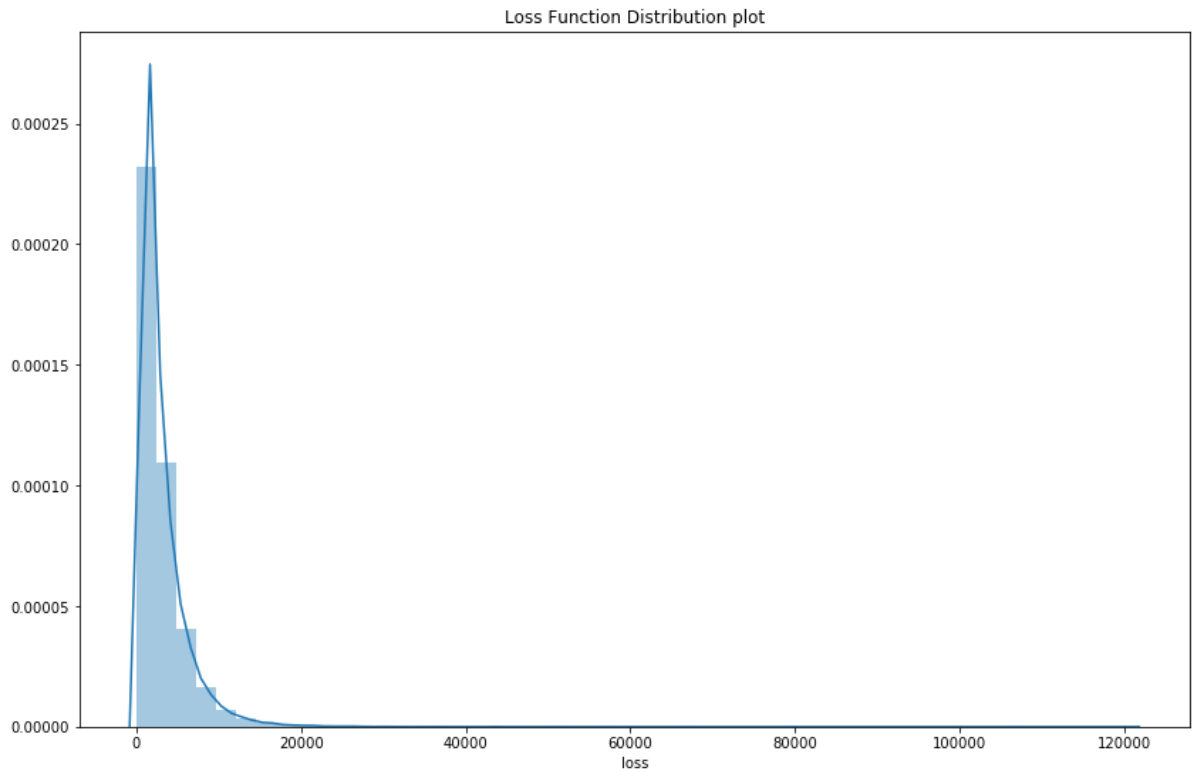
```
In [5]: plt.figure(figsize=(14,9))
sns.boxplot(data=df_train[cont_features])
plt.title('Box-Plot of Continous Features')
```

Out[5]: <matplotlib.text.Text at 0x110983c90>

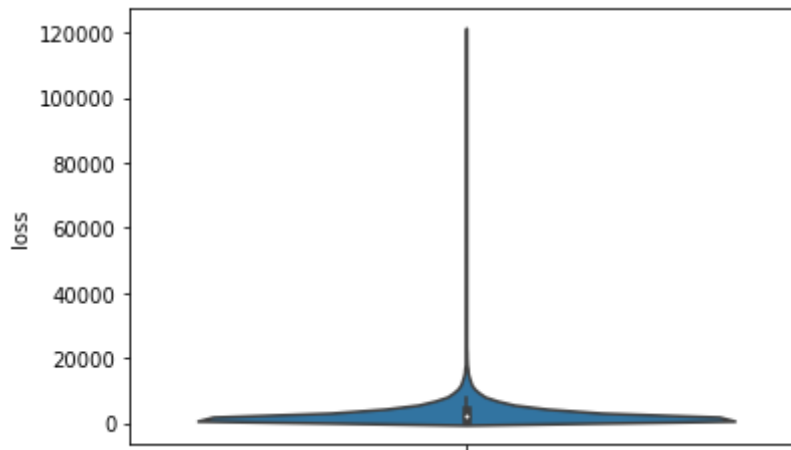


```
In [6]: plt.figure(figsize=(14,9))
sns.distplot(df_train['loss'])
plt.title('Loss Function Distribution plot')
```

Out[6]: <matplotlib.text.Text at 0x108fe7b90>

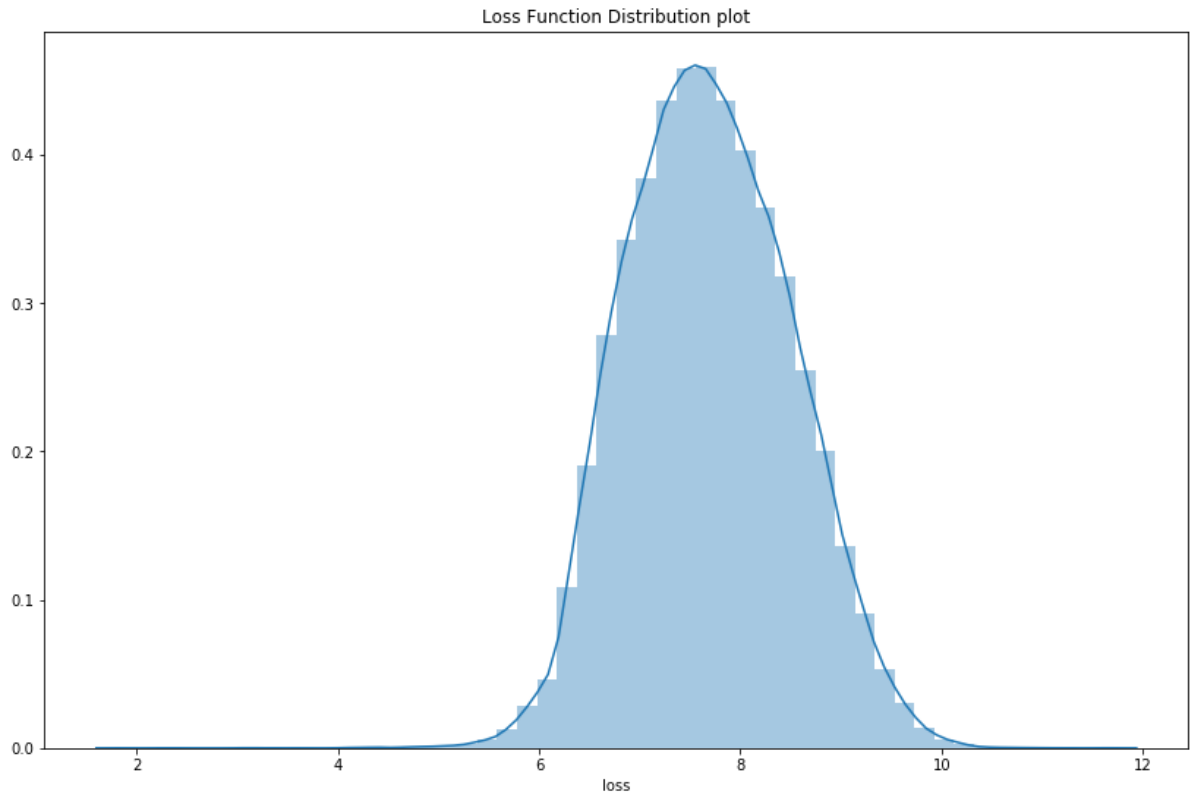


```
In [4]: sns.violinplot(data=df_train,y='loss')
plt.show()
```

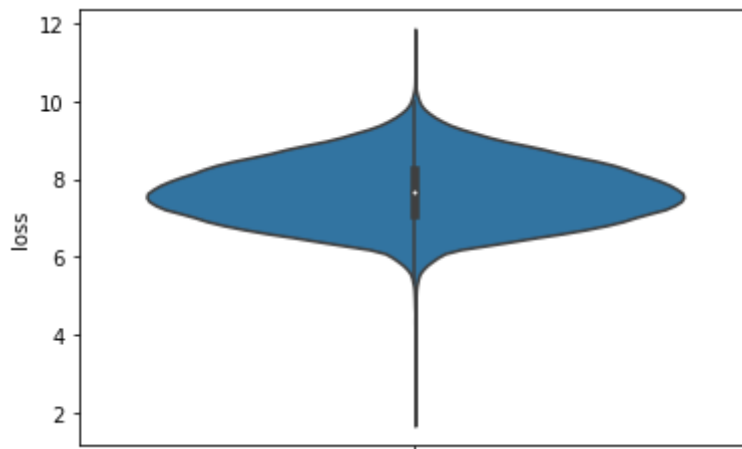


```
In [5]: # since all values of loss are so small, use loglp to  
# normalize distribution of loss  
plt.figure(figsize=(14,9))  
sns.distplot(np.loglp(df_train['loss']))  
plt.title('Loss Function Distribution plot')
```

Out[5]: <matplotlib.text.Text at 0x11969e790>

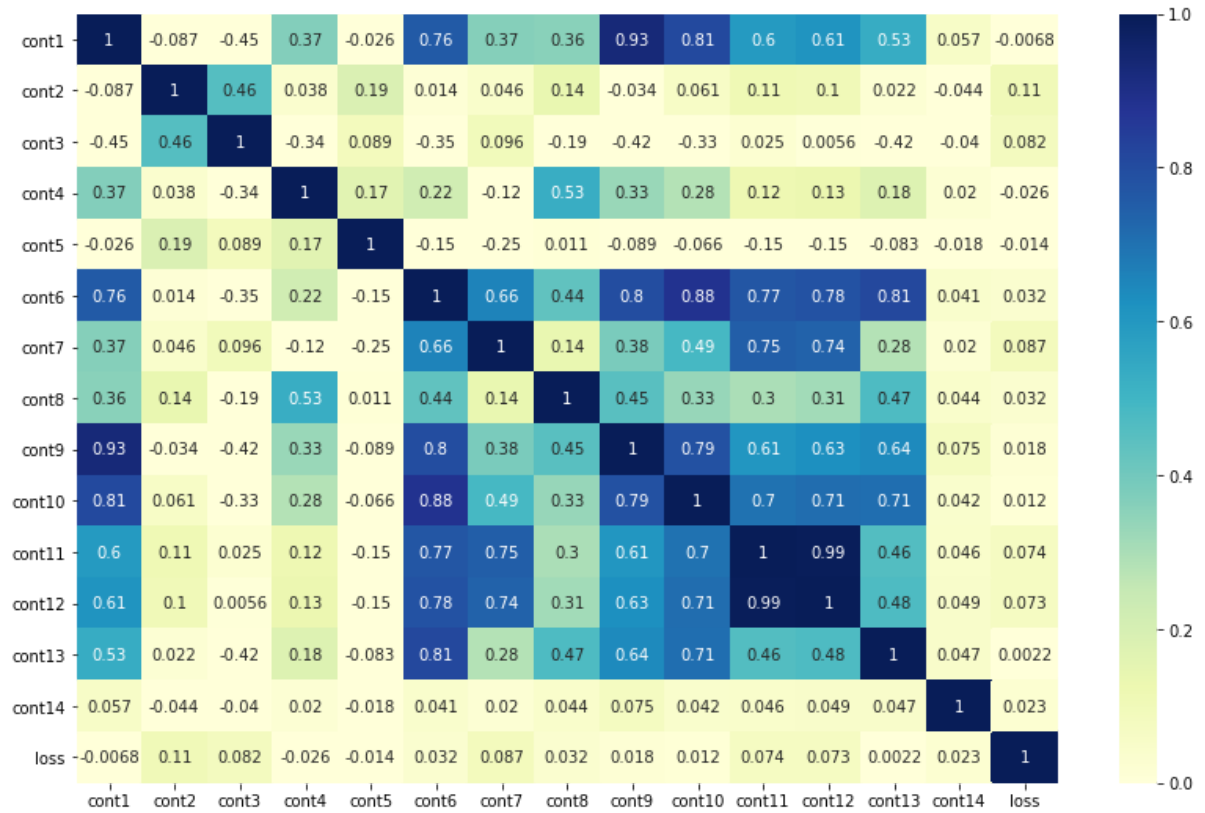


```
In [6]: df_train['loss'] = np.loglp(df_train['loss'])  
sns.violinplot(data=df_train, y='loss')  
plt.show()  
# improved skew, but still some exists in the bottom quartile
```



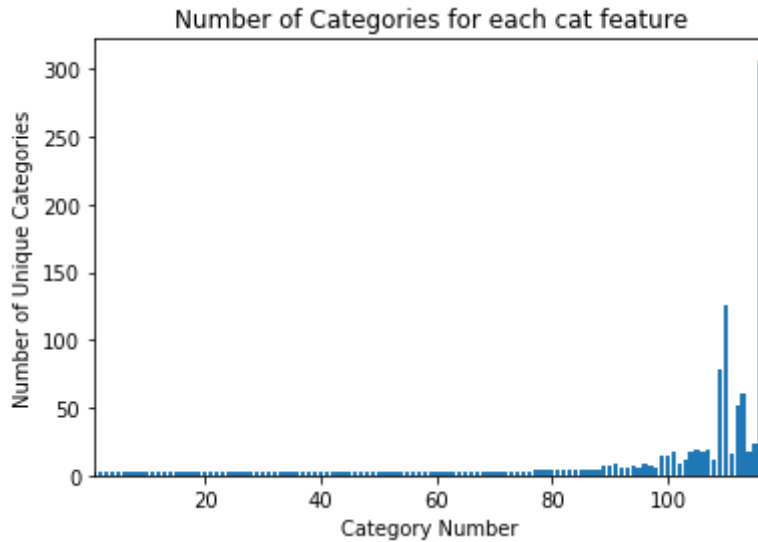
Correlation Matrix:

```
In [7]: cont_features.append('loss')
corr_cont = df_train[cont_features].corr()
corrMat = corr_cont.as_matrix()
corr_list = []
plt.figure(figsize=(14,9))
sns.heatmap(corr_cont,annot=True,vmin=0,vmax=1,cmap='YlGnBu')
plt.show()
```



```
In [8]: plt.bar(np.arange(1,117),height=np.array(df_train[cat_features].nunique
        ()))
        plt.xlim([1,117])
        plt.xlabel('Category Number')
        plt.ylabel('Number of Unique Categories')
        plt.title('Number of Categories for each cat feature')
```

Out[8]: <matplotlib.text.Text at 0x1196481d0>



Encode Categorical Features

Since many of the categorical features are non-numerical, I decided to use a one-hot encoding scheme to convert may of

```
In [9]: from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
labels =[]
for c in cat_features:
    train = df_train[c].unique()
    test = df_test[c].unique()
    labels.append(list(set(train) | set(test)))
cats = []
cats_test =[]
for i,v in enumerate(cat_features):
    #Label encode
    label_encoder = LabelEncoder()
    label_encoder.fit(labels[i])
    feature = label_encoder.transform(df_train[v])
    f_test = label_encoder.transform(df_test[v])
    feature = feature.reshape(df_train.shape[0], 1)
    f_test = f_test.reshape(df_test.shape[0],1)
    #One hot encode
    onehot_encoder = OneHotEncoder(sparse=False,n_values=len(labels[i]))
    feature = onehot_encoder.fit_transform(feature)
    f_test = onehot_encoder.fit_transform(f_test)
    cats.append(feature)
    cats_test.append(f_test)
encoded_cats = np.column_stack(cats)
encoded_catsTest=np.column_stack(cats_test)
```

```
In [10]: dataset_encoded = np.concatenate((encoded_cats,df_train[cont_features].values),axis=1)
cont_features.remove('loss')
dataTestE = np.concatenate((encoded_catsTest,df_test[cont_features].values),axis=1)
```

Machine Learning:

```
In [11]: from sklearn.metrics import mean_absolute_error
import time
```

```
In [12]: rows, columns = dataset_encoded.shape
data = dataset_encoded[:,0:(columns-1)] # remove loss column
loss = dataset_encoded[:,(columns-1)] # loss column
```

Lasso Algorithm

```
In [13]: from sklearn.linear_model import LassoCV
from sklearn.linear_model import Lasso
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
```

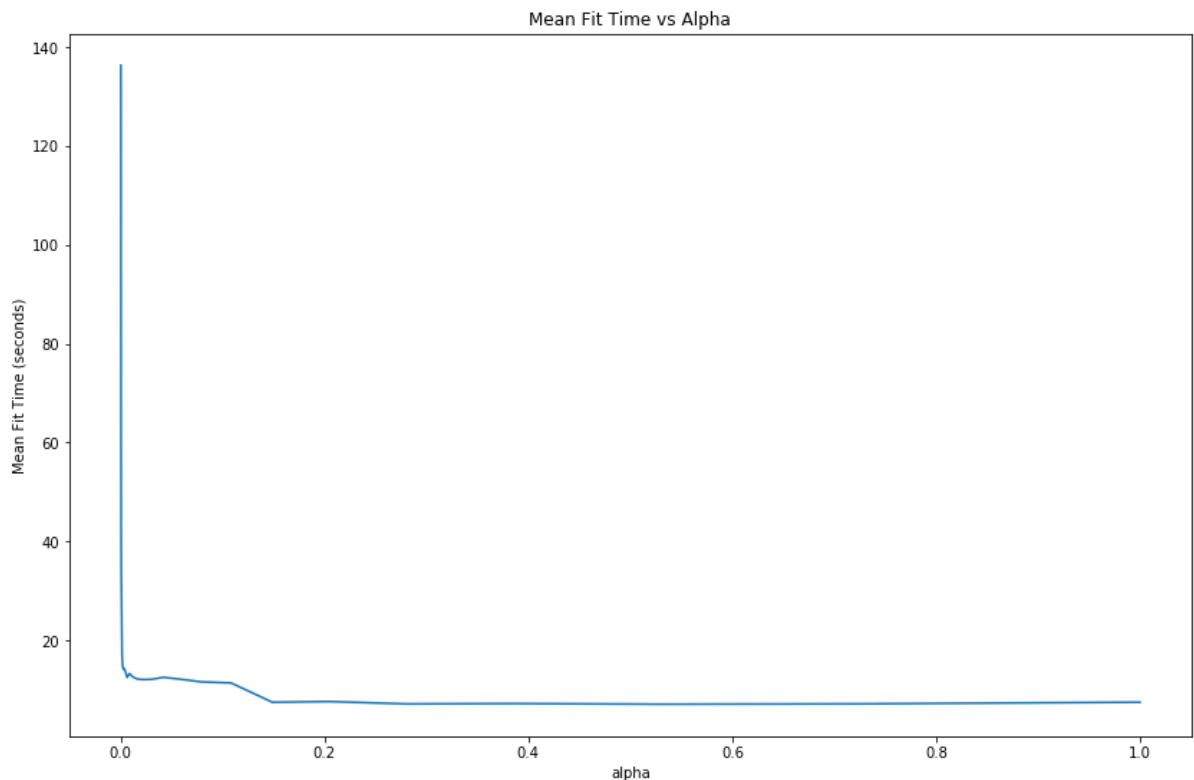
```
In [16]: seed = 2016
lasso = Lasso(random_state=seed)
alphas = np.logspace(-4,0,30)

tuned_parameters = [{'alpha':alphas}]
n_folds = 3

# Hyper-Parameter Tuning with Nested CV:
clf = GridSearchCV(lasso,tuned_parameters, cv = n_folds,scoring='neg_mean_absolute_error')
clf.fit(data,loss)
scores = clf.cv_results_['mean_test_score']
scores_std = clf.cv_results_['std_test_score']
```

```
In [18]: times = clf.cv_results_['mean_fit_time']
```

```
In [23]: plt.figure(figsize=(14,9))
plt.plot(alphas,times)
plt.title('Mean Fit Time vs Alpha')
plt.xlabel('alpha')
plt.ylabel('Mean Fit Time (seconds)')
plt.show()
```




```

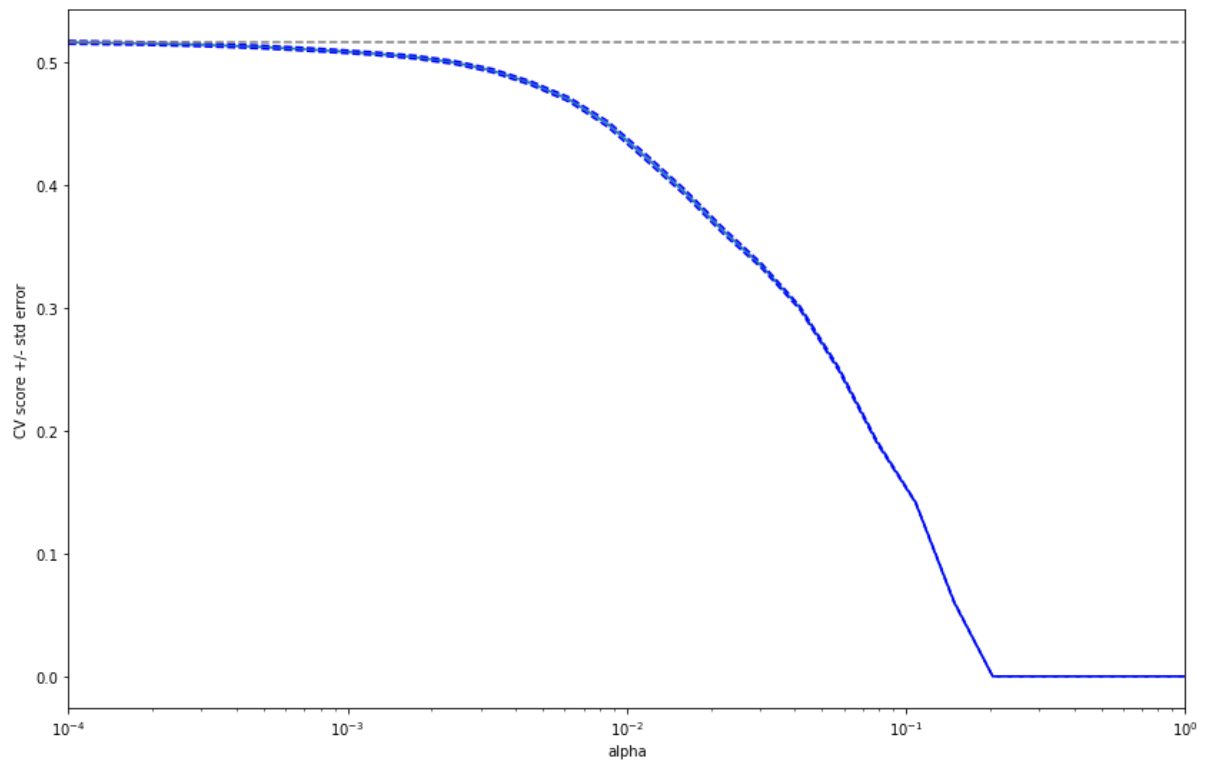
In [15]: plt.figure(figsize=(14,9))
plt.semilogx(alphas,scores)
# plot error lines showing +/- std. errors of the scores
std_error = scores_std / np.sqrt(n_folds)

plt.semilogx(alphas, scores + std_error, 'b--')
plt.semilogx(alphas, scores - std_error, 'b--')

# alpha=0.2 controls the translucency of the fill color
plt.fill_between(alphas, scores + std_error, scores - std_error, alpha=
0.2)

plt.ylabel('CV score +/- std error')
plt.xlabel('alpha')
plt.axhline(np.max(scores), linestyle='--', color='.5')
plt.xlim([alphas[0], alphas[-1]])
plt.show()

```



```
In [53]: # http://scikit-learn.org/stable/auto\_examples/model\_selection/plot\_nested\_cross\_validation\_iris.html
scores
```

```
Out[53]: array([[ 5.16069009e-01,  5.15660067e-01,  5.15155616e-01,  5.14455964e-01,
                  5.13528053e-01,  5.12331654e-01,  5.10733259e-01,  5.08882388e-01,
                  5.06782679e-01,  5.04016227e-01,  4.99832798e-01,  4.93169141e-01,
                  4.83001252e-01,  4.69620319e-01,  4.49721771e-01,  4.22483511e-01,
                  3.94223621e-01,  3.62746409e-01,  3.34176916e-01,  2.99737583e-01,
                  2.50275995e-01,  1.90533838e-01,  1.41410142e-01,  6.03400856e-02,
                  -9.19934907e-06, -9.19934907e-06, -9.19934907e-06, -9.19934907e-06,
                  -9.19934907e-06, -9.19934907e-06])
```

```
In [55]: predictions = np.expml(clf.predict(dataTestE))
```

```
In [56]: predLoss = pd.Series(predictions,name='loss')
submission = pd.concat([df_test['id'],predLoss],axis=1)
```

```
In [58]: submission.to_csv('submission-LASSO.csv',index=False) #Scored: 1239.0284
```

XGBoost Algorithm

```
In [27]: from sklearn.model_selection import KFold
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error
import time
import scipy.stats as stats
```

```
In [28]: rows, columns = dataset_encoded.shape
data = dataset_encoded[:,0:(columns-1)] # remove loss column
loss = dataset_encoded[:,(columns-1)] # loss column
```

```
In [ ]: seed = 2016
model = XGBRegressor(seed=2016,learning_rate=0.3,n_jobs=-1)
```

```
In [23]: estimators= stats.randint(110,200)
max_depth = range(3,6)
learning_rate = stats.uniform(0.01,0.07)
colsample_bytree = stats.uniform(0.5,0.45)
min_child_weight= range(1,4)

tuned_parameters={'n_estimators':estimators,
                  'max_depth': max_depth,}
# learning_rate': learning_rate',
#                  'colsample_bytree': colsample_bytree,
#                  'min_child_weight': min_child_weight
n_folds = 3
clf = RandomizedSearchCV(model,param_distributions=tuned_parameters, cv
= n_folds,
                        scoring='neg_mean_absolute_error',
                        n_jobs=-1,verbose=1)
```

```
In [19]: model.fit(data,loss,eval_metric='mae')
```

```
Out[19]: XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                    colsample_bytree=1, gamma=0, learning_rate=0.3, max_delta_step=
                    0,
                    max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
                    n_jobs=-1, nthread=None, objective='reg:linear', random_state=0,
                    reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=2016,
                    silent=True, subsample=1)
```

```
In [20]: predictions = np.expml(model.predict(dataTestE))
```

```
In [21]: predictions
```

```
Out[21]: array([1401.5009, 1431.4539, 3974.6248, ..., 1353.4996, 3306.5022,
                3045.555 ], dtype=float32)
```

```
In [22]: predLoss = pd.Series(predictions,name='loss')
submission = pd.concat([df_test['id'],predLoss],axis=1)
submission.to_csv('submission-XGBOOST2.csv',index=False) #Scored: 1239.0
284
```

```
In [ ]: clf.fit(data,loss)
```

Fitting 3 folds for each of 10 candidates, totalling 30 fits

```
In [ ]: scores = clf.cv_results_['neg_mean_absolute_error']
scores_mt = clf.cv_results_['mean_test_score']
scores_std = clf.cv_results_['std_test_score']
```

```
In [ ]:
```

In [29]: **from sklearn.model_selection import cross_val_score**

```
model = XGBRegressor(seed=2016,max_depth=7)
scores = cross_val_score(model, data, loss, cv=4)
```

In [30]: `scores`

Out[30]: `array([0.5430061 , 0.55242186, 0.55276236, 0.55382748])`

In []:

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from google.colab import files

from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error
import time
import scipy.stats as stats

# project done in Google Co-lab to take advantage of the GPU Computational power
!pip install -U -q PyDrive

from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# 1. Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# PyDrive reference:
# https://googledrive.github.io/PyDrive/docs/build/html/index.html

# 2. Create & upload a file text file.
# uploaded = drive.CreateFile({'title': 'Sample upload.txt'})
# uploaded.SetContentString('Sample upload file content')
# uploaded.Upload()
# print('Uploaded file with ID {}'.format(uploaded.get('id')))

# # 3. Load a file by ID and print its contents.
# downloaded = drive.CreateFile({'id': uploaded.get('id')})
# print('Downloaded content "{}"'.format(downloaded.GetContentString()))

import os
fList = drive.ListFile({'q': "'1kPNjhSKgaQQY-AYX3EufYTO8DdVW3D65' in parents"}).GetList()
local_download_path=os.path.expanduser('~')
for f in fList:
    # 3. Create & download by id.
    print('title: %s, id: %s' % (f['title'], f['id']))
    fname = os.path.join(local_download_path, f['title'])
    print('downloading to {}'.format(fname))
    f_ = drive.CreateFile({'id': f['id']})
    f_.GetContentFile(fname)

with open(fname, 'r') as f:
    print(f.read())

```



title: dataset_encoded, id: 1e4o8l-Cw2EWS-VpCDoo4uscyEG48ZrL4
downloading to /content/dataset_encoded
title: dataTestE, id: 1fhdVakV9zwEBhgXH3hU6fJQugI0eQRve
downloading to /content/dataTestE
title: pml_test_features.csv, id: 1_DdNm0SCrGgaUDEc9QaRD0bKKjsz-bHO
downloading to /content/pml_test_features.csv
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

```
dataset_encoded = np.fromfile('/content/dataset_encoded')  
  
NotebookApp.iopub_data_rate_limit=10000000.0 (bytes/sec)  
dataTestE = np.fromfile('/content/dataTestE').
```

```
dataset_encoded=dataset_encoded.reshape(131822,1154)  
dataTestE = dataTestE.reshape(56496,1153)
```

```
rows, columns = dataset_encoded.shape  
data = dataset_encoded[:,0:(columns-1)] # remove loss column  
loss = dataset_encoded[:,(columns-1)] # loss column
```

```
seed = 2016  
model = XGBRegressor(seed=2016)  
  
max_depth = range(3,10)  
  
tuned_parameters=[{'max_depth': max_depth}]  
n_folds = 3  
clf = GridSearchCV(model,tuned_parameters, cv = n_folds,  
                    scoring='neg_mean_absolute_error',  
                    n_jobs=-1,verbose=1)
```

```
model = XGBRegressor(seed=2016,max_depth=5)
```

```
model.fit(data,loss,eval_metric='mae')
```

```
↳ XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,  
               colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,  
               max_depth=5, min_child_weight=1, missing=None, n_estimators=100,  
               n_jobs=1, nthread=None, objective='reg:linear', random_state=0,  
               reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=2016,  
               silent=True, subsample=1)
```

```
predictions = np.expml(model.predict(dataTestE))
```

```
predLoss = pd.Series(predictions,name='loss')  
submission = pd.concat([df_test['id'],predLoss],axis=1)
```

```
df_test = pd.read_csv('/content/pml_test_features.csv')
```

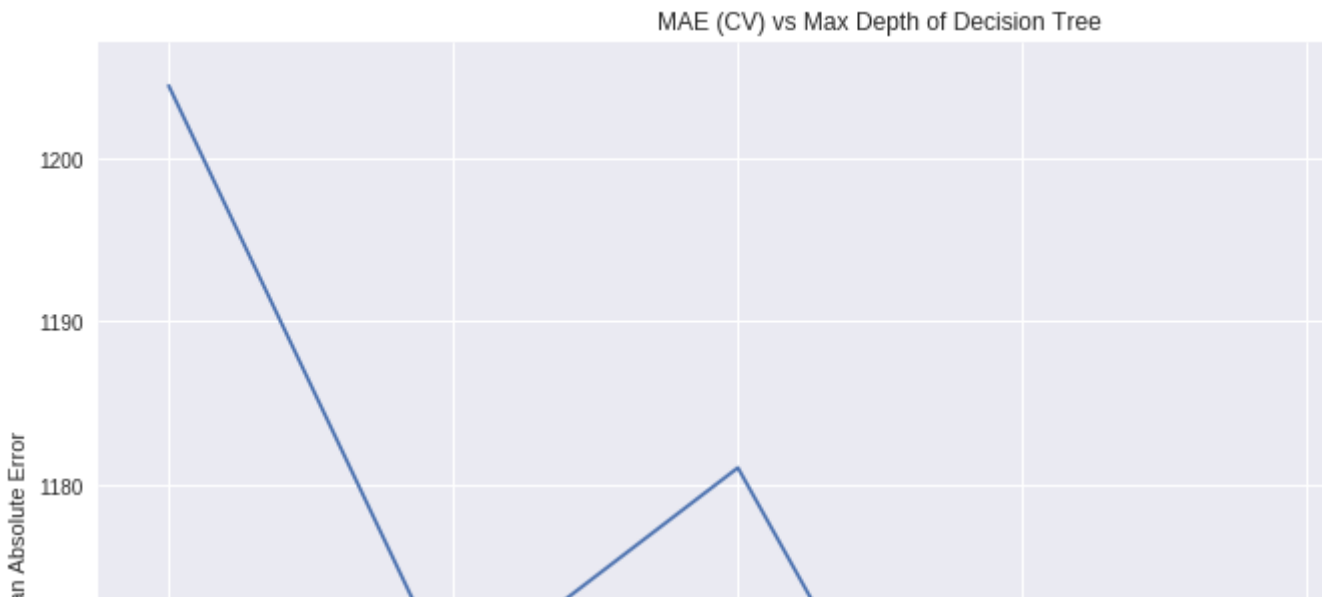
```
submission.to_csv('submission-XGBOOST3-5Depth.csv',index=False)
files.download('submission-XGBOOST3-5Depth.csv')
```

```
elapsed = []
results = []
# Nested Cross Validation :
from sklearn import cross_validation
for i in range(4,10):
    start = time.time()
    val_size = 0.1
    X_train, X_val, Y_train, Y_val = cross_validation.train_test_split(data, loss, test_size=val_size)
    model = XGBRegressor(seed=2016,max_depth=i)
    model.fit(X_train,Y_train,eval_metric='mae')
    end = time.time()
    elapsed.append(end-start)
    results.append(mean_absolute_error(np.expml(Y_val),np.expml(model.predict(X_val))))
print "Performance:" +str(results[-1])
print "Time:" +str(elapsed[-1])
```

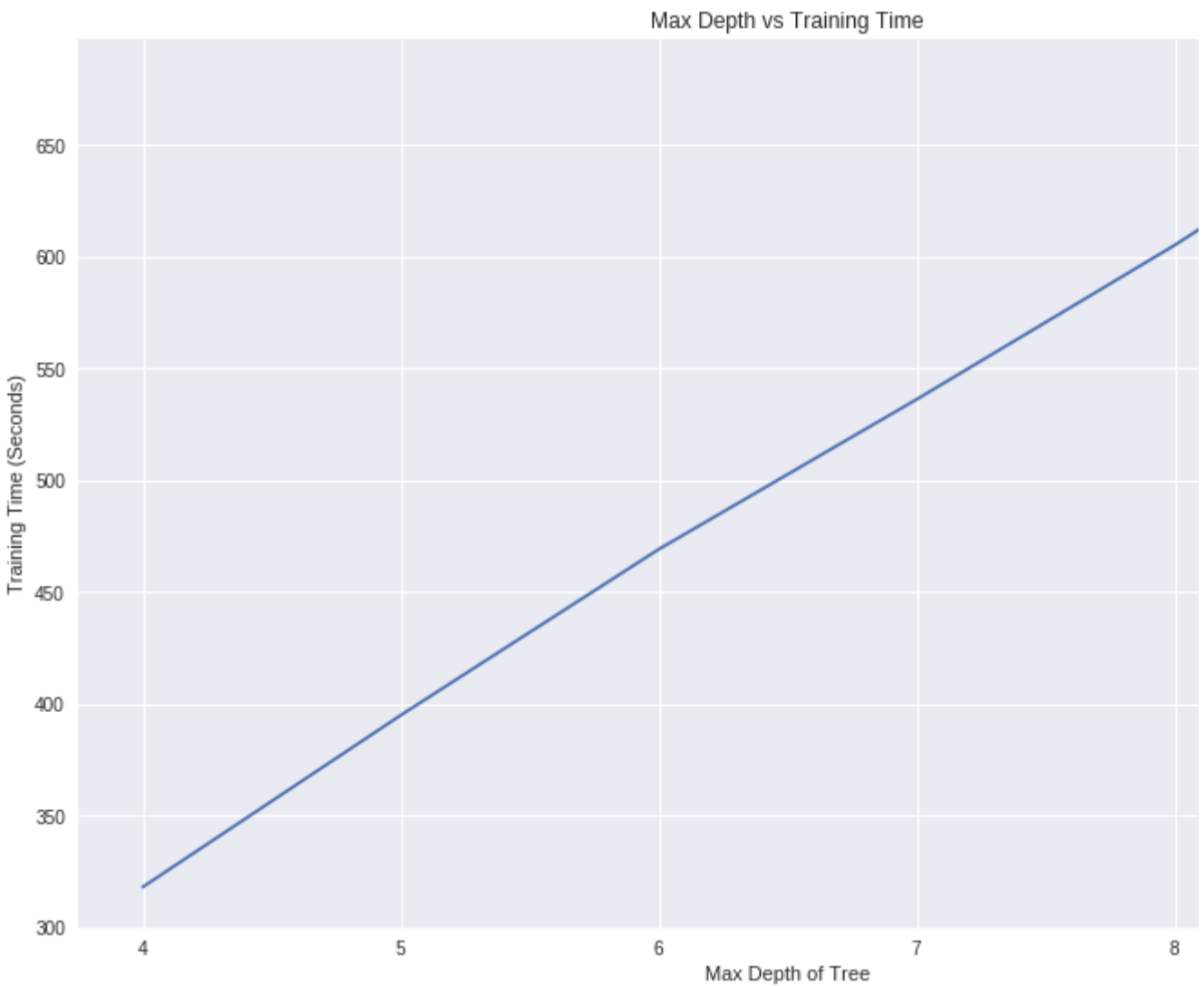
```
➤ /usr/local/lib/python2.7/dist-packages/sklearn/cross_validation.py:41: DeprecationWarning:
    "This module will be removed in 0.20.", DeprecationWarning)
Performance:1204.4408884114928
Time:317.990732908
Performance:1167.7376902300489
Time:394.815212011
Performance:1181.042126941717
Time:469.015637159
Performance:1149.8536614067507
Time:536.06773591
Performance:1164.6145181493175
Time:605.0114851
Performance:1151.1771897726023
Time:679.411887884
```

```
results = [1204.4408884114928,1167.7376902300489,1181.042126941717,1149.8536614067507,1164.6145181493175,1151.1771897726023,679.411887884]
plt.figure(figsize=(14,9))
plt.plot(np.arange(4,10),results)
plt.title('MAE (CV) vs Max Depth of Decision Tree').
plt.ylabel('Mean Absolute Error')
plt.xlabel('Max Depth of Tree')
plt.show()
```

➤



```
plt.figure(figsize=(14,9))
plt.plot(np.arange(4,10),elapsed)
plt.xlabel('Max Depth of Tree')
plt.ylabel('Training Time (Seconds)')
plt.title('Training Time vs Max Depth')
plt.show()
```




```

model = XGBRegressor(seed=2016,max_depth=7)
model.fit(data,loss,eval_metric='mae')

predictions = np.expml(model.predict(dataTestE))
predLoss = pd.Series(predictions,name='loss')
submission = pd.concat([df_test['id'],predLoss],axis=1)

submission.to_csv('submission-XGBOOST3-7Depth.csv',index=False)
files.download('submission-XGBOOST3-7Depth.csv')

from sklearn.model_selection import cross_val_score

model = XGBRegressor(seed=2016,max_depth=7)
scores = cross_val_score(model, data, loss, cv=4)

```

▼ Cross Validated Accuracy:

scores

```

↳ array([0.5430061 , 0.55242186, 0.55276236, 0.55382748])

```

```

plt.plot(np.arange(1,5),scores).

```

```

↳ [<matplotlib.lines.Line2D at 0x7f1db375d690>]

```

