lab5

September 27, 2016

1 EE 379K Lab 5

1.1 Rohan Nagar and Wenyang Fu

In [25]:
```
import numpy as np
import matplotlib as mpl
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import xgboost as xgb

from mpl_toolkits.mplot3d import Axes3D

%matplotlib inline
```

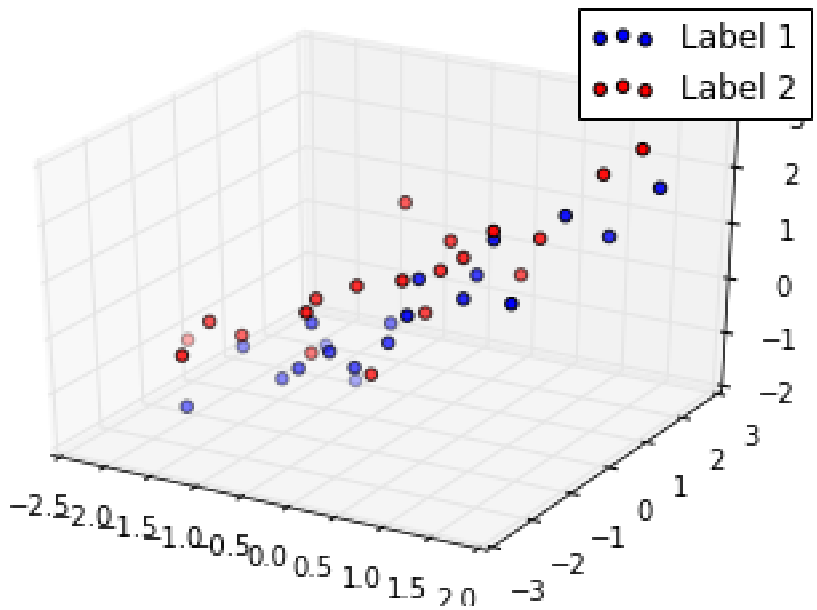
2 Problem 1: LDA

2.0.1 Part 1

Generate 20 random points in $d = 3$, from a Gaussian multivariate distribution with mean $[0, 0, 0]$ and covariance matrix $[[1, 0.9, 0.9], [0.9, 1, 0.9], [0.9, 0.9, 1]]$. Call this data with label 1. Also generate 20 random points in $d = 3$ from another Gaussian with mean $[0, 0, 1]$ and covariance $[[1, 0.8, 0.8], [0.8, 1, 0.8], [0.8, 0.8, 1]]$. Call that data with label 2. Create a three dimensional plot of the clouds of data points, labeled with the two labels.

In [2]:
```
# Covariance matrices
cov1 = [[1, 0.9, 0.9],
        [0.9, 1, 0.9],
        [0.9, 0.9, 1]]
cov2 = [[1, 0.8, 0.8],
        [0.8, 1, 0.8],
        [0.8, 0.8, 1]]

# Generate the samples
label1_samples = np.random.multivariate_normal([0, 0, 0], cov1, 20)
label2_samples = np.random.multivariate_normal([0, 0, 1], cov2, 20)
samples = [label1_samples, label2_samples]

# Plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(label1_samples.T[0], label1_samples.T[1], label1_samples.T[2], label='Label 1')
ax.scatter(label2_samples.T[0], label2_samples.T[1], label2_samples.T[2], c='r', label='Label 2
ax.legend()
fig.show()
```

```
C:\Anaconda3\lib\site-packages\matplotlib\figure.py:397: UserWarning: matplotlib is currently using a no
  "matplotlib is currently using a non-GUI backend, "
```



2.0.2 Part 2

Perform a projection of the data on one dimension using Fischer's Linear Discriminant as explained in class (see also http://research.cs.tamu.edu/prism/lectures/pr/pr_l10.pdf). No sklearn LDA functions here, just friendly linear algebra.

Steps to LDA

(With help from Sebastian Raschka)

1. Compute the d-dimensional mean vectors for the different classes from the dataset.
2. Compute the scatter matrices (in-between-class and within-class scatter matrix).
3. Compute the eigenvectors $(e_1, e_2, ..., e_d)$ and corresponding eigenvalues $(\lambda_1, \lambda_2, ..., \lambda_d)$ for the scatter matrices.
4. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a $d \times k$ dimensional matrix W (where every column represents an eigenvector).
5. Use this $d \times k$ eigenvector matrix to transform the samples onto the new subspace. This can be summarized by the matrix multiplication: $Y = X \times W$ (where X is a $n \times d$-dimensional matrix representing the nn samples, and Y are the transformed $n \times k$-dimensional samples in the new subspace).

```
In [3]: # Find the mean feature vector within each class
        label1_mean = label1_samples.mean(axis=0)
        label2_mean = label2_samples.mean(axis=0)

        mean_vectors = [label1_mean, label2_mean]
```

```
In [4]: # Find the within-class scatter matrix
```

```
        d = label1_mean.shape[0] # number of features
        # Shape is d x d, where d is the number of features.
        S_W = np.zeros((d,d))  # Between-class scatter matrix

        class_sc_matrix = np.zeros((d,d))
        # To compute the within-class scatter matrix, sum the outer products of
        # each row in the matrix (in-class samples - feature mean vector) with itself
        for cl, mv in zip(samples, mean_vectors):
            class_sc_matrix = np.zeros((d,d))
            temp = cl - mv # Subtract mean vector from every row in data matrix
            for row in temp:
                r = row.reshape((d, 1)) # reshape into column vector
                class_sc_matrix += r @ r.T # Outer product
            S_W += class_sc_matrix

        print('within-class Scatter Matrix:\n', S_W)
```

```
within-class Scatter Matrix:
 [[ 36.69900982  31.70222121  31.40593811]
 [ 31.70222121  38.01660886  31.50572518]
 [ 31.40593811  31.50572518  36.52286951]]
```

In [5]:
```
# Find the between-class scatter matrix

        all_samples = np.vstack(samples)
        # Mean feature vector for combined samples between all classes
        combined_mean_vec = all_samples.mean(axis=0).reshape((-1, 1))
        print(combined_mean_vec.shape)

        S_B = np.zeros((d,d))

        for cl, mean_vec in zip(samples, mean_vectors):
            n = cl.shape[0]  #Number of samples within each class
            mean_vec = mean_vec.reshape((-1,1)) # make column vector
            temp = mean_vec - combined_mean_vec
            S_B += n * temp @ temp.T

        print('between-class Scatter Matrix:\n', S_B)
```

```
(3, 1)
between-class Scatter Matrix:
 [[ 0.06564451  0.05293559 -0.49348348]
 [ 0.05293559  0.04268714 -0.39794399]
 [-0.49348348 -0.39794399  3.7097684 ]]
```

2.0.3 Solving the generalized eigenvalue problem for the matrix $S_W^{-1}S_B$

In [6]:
```
eig_vals, eig_vecs = np.linalg.eig(np.linalg.inv(S_W) @ S_B)

        for i in range(len(eig_vals)):
            eigvec_sc = eig_vecs[:, i].reshape(-1, 1)
            print('\nEigenvector {}: \n{}'.format(i+1, eigvec_sc.real))
            print('Eigenvalue {:}: {:.2e}'.format(i+1, eig_vals[i].real))
```

```
Eigenvector 1:
[[ 0.44084404]
```

```
 [ 0.33804066]
 [-0.83149566]]
Eigenvalue 1: 5.85e-01

Eigenvector 2:
[[-0.82148395]
 [-0.00189514]
 [-0.10947932]]
Eigenvalue 2: 1.10e-19

Eigenvector 3:
[[-0.82148395]
 [-0.00189514]
 [-0.10947932]]
Eigenvalue 3: 1.10e-19
```

In [7]:
```
# Sort eigenvectors by their corresponding eigenvalues in descending order:
# aka highest to lowest
# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs = sorted(eig_pairs, key=lambda k: k[0], reverse=True)

# Visually confirm that the list is correctly sorted by decreasing eigenvalues

print('Eigenvalues in decreasing order:\n')
for i in eig_pairs:
    print(i[0])
```

```
Eigenvalues in decreasing order:

0.585147025239
1.19881142332e-17
1.19881142332e-17
```

In [8]:
```
# "Explained variance"

print('Variance explained:\n')
eigv_sum = sum(eig_vals)
for i,j in enumerate(eig_pairs):
    print('eigenvalue {0:}: {1:.2%}'.format(i+1, (j[0]/eigv_sum).real))
```

```
Variance explained:

eigenvalue 1: 100.00%
eigenvalue 2: 0.00%
eigenvalue 3: 0.00%
```

In [9]:
```
# Choose eigenvector with the highest eigenvalue:

W = eig_pairs[0][1]
print('Matrix W:\n', W.real)
```

```
Matrix W:
 [ 0.44084404  0.33804066 -0.83149566]
```

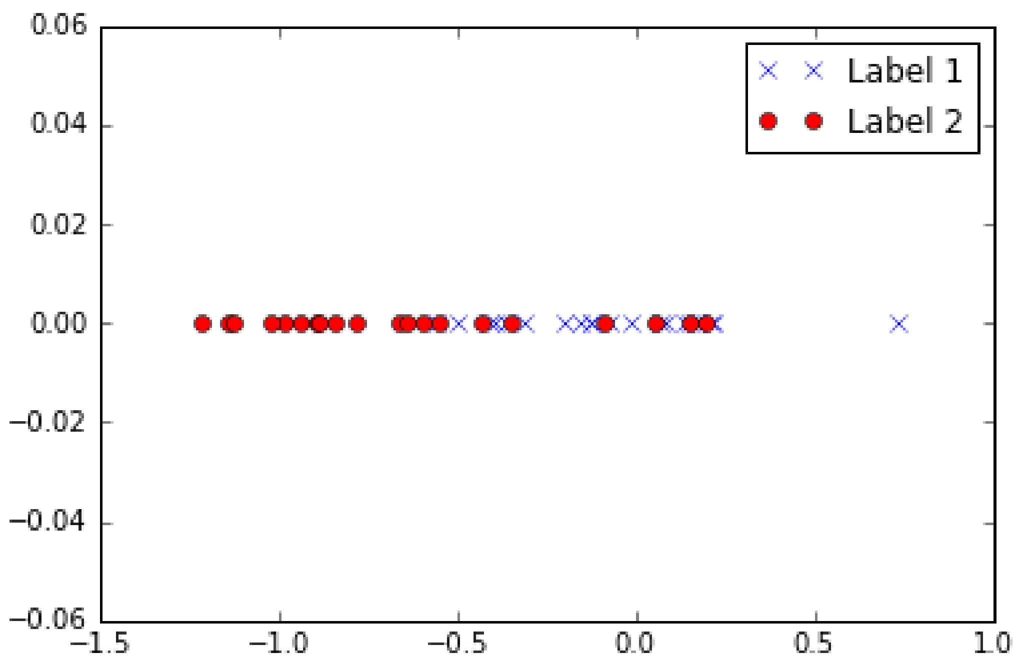
Project data matrix onto new subspace: $Y = X \times W$

In [10]:
```
label1_lda = label1_samples @ W
label2_lda = label2_samples @ W
label1_lda
```

Out[10]:
```
array([ 0.16345270+0.j, -0.12385080+0.j, -0.36595375+0.j, -0.39639235+0.j,
       -0.31178165+0.j,  0.15096145+0.j,  0.21511966+0.j, -0.07142196+0.j,
       -0.55529873+0.j, -0.40983368+0.j,  0.08282818+0.j, -0.15575657+0.j,
       -0.01690460+0.j,  0.13562497+0.j, -0.20025453+0.j,  0.73554682+0.j,
        0.07043355+0.j, -0.12308910+0.j,  0.20857237+0.j, -0.49996972+0.j])
```

In [11]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(label1_lda, np.zeros(label1_lda.shape[0]), label='Label 1', marker='x', linestyle='')
ax.plot(label2_lda, np.zeros(label2_lda.shape[0]), c='r', label='Label 2', marker='o', linestyl
ax.legend()
fig.show()
```

```
C:\Anaconda3\lib\site-packages\numpy\core\numeric.py:482: ComplexWarning: Casting complex values to real
  return array(a, dtype, copy=False, order=order)
C:\Anaconda3\lib\site-packages\matplotlib\figure.py:397: UserWarning: matplotlib is currently using a no
  "matplotlib is currently using a non-GUI backend, "
```



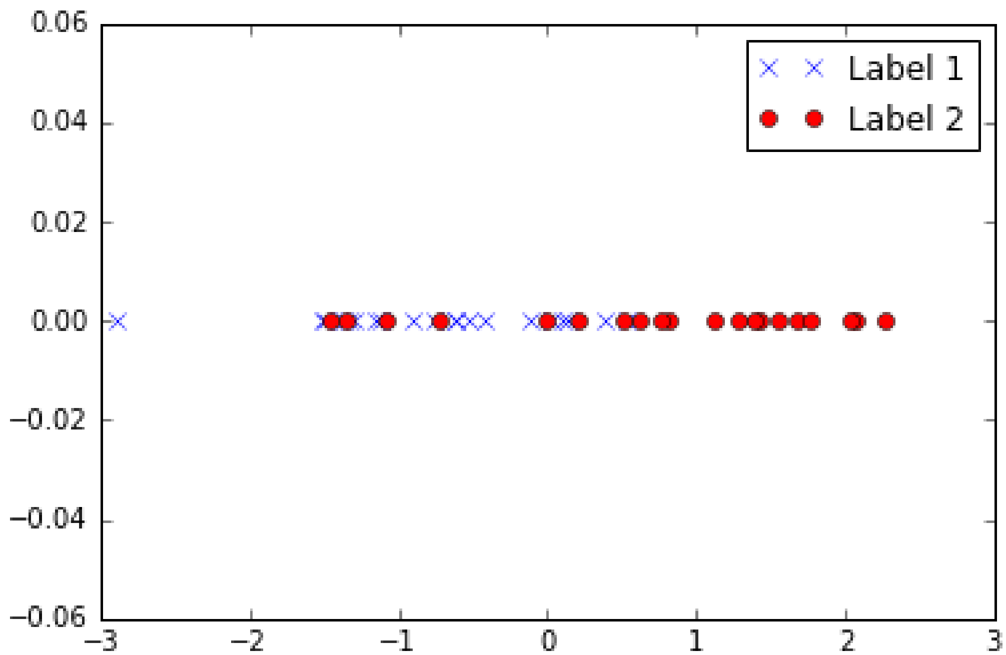
2.0.4 Part 3

Use `sklearn` to perform Linear Discriminant Analysis. Compare the results.

2.0.5 The the variants of LDA produced similar results (with different scaling), and both achieved the goal of separating the two classes very well.

In [12]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

# LDA
n_class1 = label1_samples.shape[0]
n_class2 = label2_samples.shape[0]

sklearn_lda = LDA(n_components=1)
y = np.hstack((np.zeros(n_class1), np.ones(n_class2)))
lda_data = sklearn_lda.fit_transform(all_samples, y)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(lda_data[:n_class1], np.zeros(label1_lda.shape[0]), label='Label 1', marker='x', lines
ax.plot(lda_data[n_class1:], np.zeros(label2_lda.shape[0]),  c='r', label='Label 2', marker='o
ax.legend()
fig.show()
```

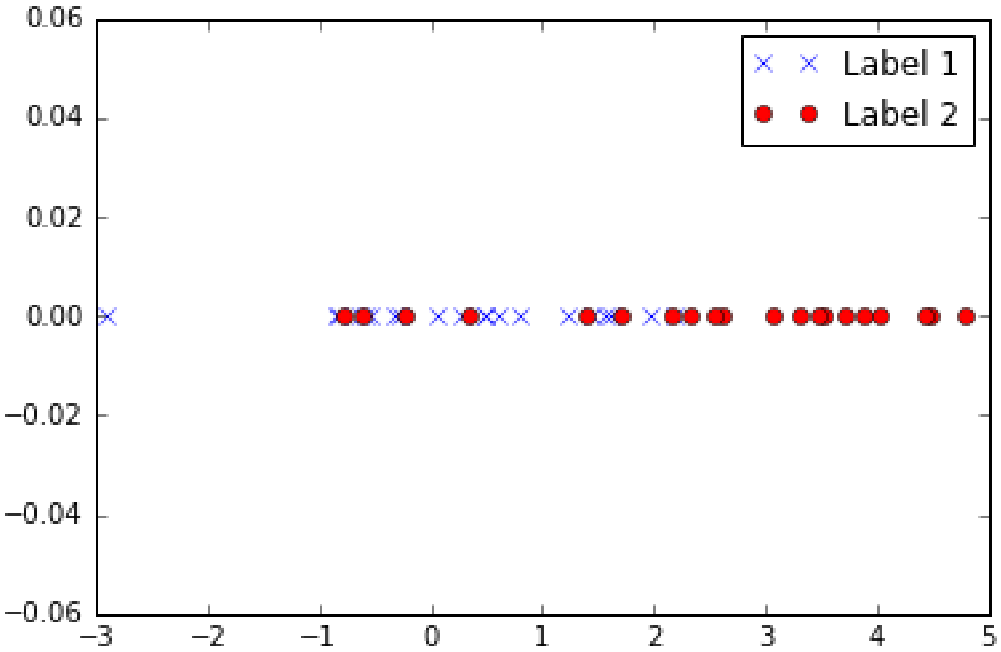
```
C:\Anaconda3\lib\site-packages\matplotlib\figure.py:397: UserWarning: matplotlib is currently using a no
  "matplotlib is currently using a non-GUI backend, "
```



In [20]:
```
W_skl = sklearn_lda.coef_
label1_lda_skl = label1_samples @ W_skl.T
label2_lda_skl = label2_samples @ W_skl.T
y_label1 = np.zeros(label1_lda.shape[0])
y_label2 = np.zeros(label2_lda.shape[0])

fig = plt.figure()
```

```
ax = fig.add_subplot(111)
ax.plot(label1_lda_skl, y_label1, label='Label 1', marker='x', linestyle='')
ax.plot(label2_lda_skl, y_label2,  c='r', label='Label 2', marker='o', linestyle='')
ax.legend()
fig.show()
```

```
C:\Anaconda3\lib\site-packages\matplotlib\figure.py:397: UserWarning: matplotlib is currently using a no
  "matplotlib is currently using a non-GUI backend, "
```




```
In [28]: print('Our LDA implementation\'s subspace vector\n', W.real)

         print('sklearn\'s LDA subspace vector\n', W_skl.reshape(-1))
```

```
Our LDA implementation's subspace vector
 [ 0.44084404  0.33804066 -0.83149566]
sklearn's LDA subspace vector
 [-1.73724856 -1.33212791  3.27670219]
```

3 Problem 2: More Kaggle

3.0.1 Part 1

Goal: Get the best score you can in the Housing prices competition. https://www.kaggle.com/c/house-prices-advanced-regression-techniques/

Kaggle Information Team: `DataScienceSquad`
Usernames: `RohanNagar`, `aetherzephyr`

We need to do some preprocessing first.

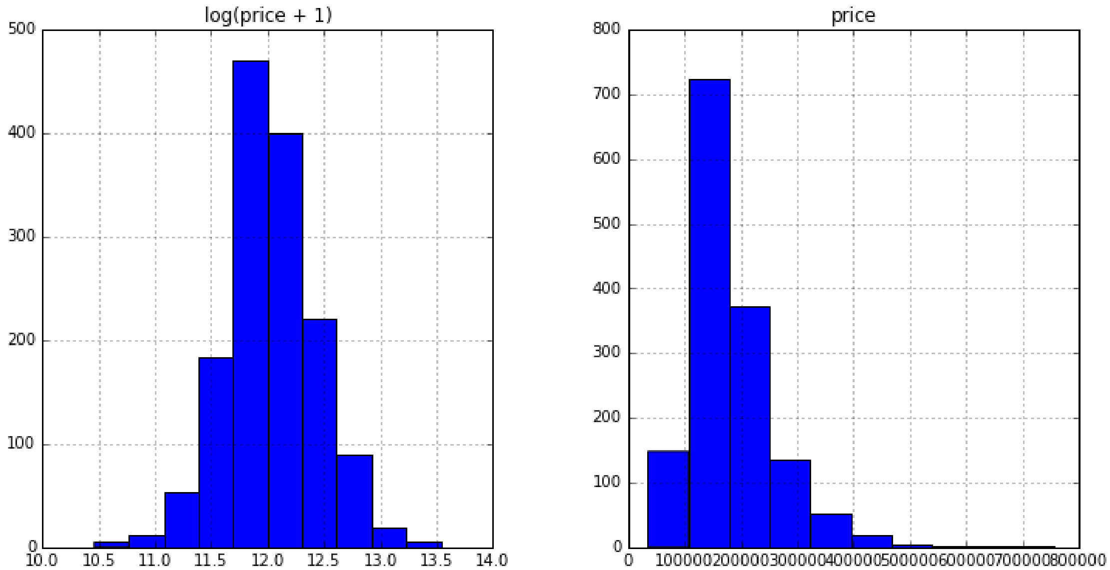
In [88]:
```
import pandas as pd
from scipy.stats import skew

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

train.head()
```

Out[88]:
```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0   1          60       RL           65     8450   Pave   NaN      Reg
1   2          20       RL           80     9600   Pave   NaN      Reg
2   3          60       RL           68    11250   Pave   NaN      IR1
3   4          70       RL           60     9550   Pave   NaN      IR1
4   5          60       RL           84    14260   Pave   NaN      IR1

  LandContour Utilities    ...     PoolArea PoolQC Fence MiscFeature MiscVal  \
0         Lvl    AllPub    ...            0    NaN   NaN         NaN       0
1         Lvl    AllPub    ...            0    NaN   NaN         NaN       0
2         Lvl    AllPub    ...            0    NaN   NaN         NaN       0
3         Lvl    AllPub    ...            0    NaN   NaN         NaN       0
4         Lvl    AllPub    ...            0    NaN   NaN         NaN       0

  MoSold YrSold  SaleType  SaleCondition  SalePrice
0      2   2008        WD         Normal     208500
1      5   2007        WD         Normal     181500
2      9   2008        WD         Normal     223500
3      2   2006        WD        Abnorml     140000
4     12   2008        WD         Normal     250000

[5 rows x 81 columns]
```

In [89]:
```
all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],
                      test.loc[:,'MSSubClass':'SaleCondition']))
```

In [90]:
```
mpl.rcParams['figure.figsize'] = (12.0, 6.0)
prices = pd.DataFrame({"price":train["SalePrice"], "log(price + 1)":np.log1p(train["SalePrice"]
prices.hist()
```

Out[90]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x10ce579e8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x10e17ee80>]], dtype=object)
```




```
In [91]: #log transform the target:
         train["SalePrice"] = np.log1p(train["SalePrice"])

         #log transform skewed numeric features:
         numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index

         skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna())) #compute skewness
         skewed_feats = skewed_feats[skewed_feats > 0.75]
         skewed_feats = skewed_feats.index

         all_data[skewed_feats] = np.log1p(all_data[skewed_feats])

         all_data = pd.get_dummies(all_data)
```

```
In [92]: #filling NA's with the mean of the column:
         all_data = all_data.fillna(all_data.mean())
```

```
In [93]: #creating matrices for sklearn:
         X_train = all_data[:train.shape[0]]
         X_test = all_data[train.shape[0]:]
         y = train.SalePrice
```

3.0.2 Part 2

Train a ridge regression and a lasso regression model. Optimize the alphas using cross validation. What is the best score you can get from a single ridge regression model and from a single lasso model?

```
In [94]: from sklearn.linear_model import RidgeCV, LassoCV, Lasso
         from sklearn.cross_validation import cross_val_score

         model_ridge = RidgeCV(alphas= [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]).fit(X_train, y)
         model_lasso = LassoCV(alphas = [1, 0.1, 0.001, 0.0005]).fit(X_train, y)
```

```
ridge_score = -cross_val_score(model_ridge, X_train, y, scoring="mean_squared_error", cv = 5)
print("Ridge Best RMSE: {}".format(np.sqrt(ridge_score).mean()))

lasso_score = -cross_val_score(model_lasso, X_train, y, scoring="mean_squared_error", cv = 5)
print("Lasso Best RMSE: {}".format(np.sqrt(lasso_score).mean()))
```

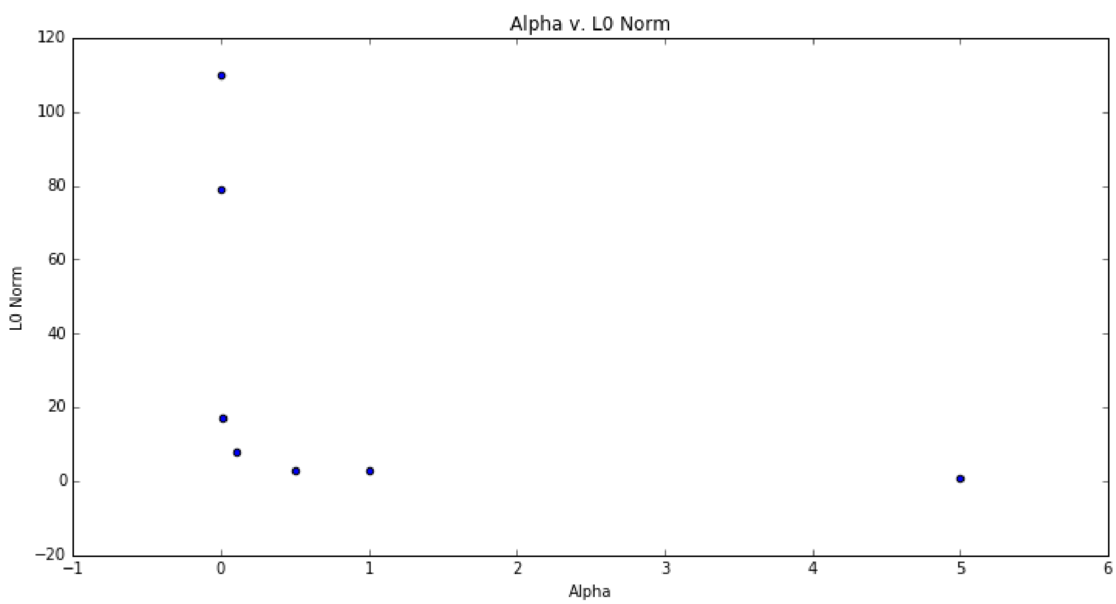
```
Ridge Best RMSE: 0.12775910031598242
Lasso Best RMSE: 0.12314421090977437
```

3.0.3 Part 3

Plot the l_0 norm (number of nonzeros) of the coefficients that lasso produces as you vary alpha.

```
In [95]: def l0_norm(coefs):
             count = 0
             for coef in coefs:
                 if coef != 0:
                     count += 1
             return count

         alphas = [5, 1, 0.5, 0.1, 0.01, 0.001, 0.0005]
         l0_norms = []
         for alpha in alphas:
             model = Lasso(alpha=alpha).fit(X_train, y)
             l0_norms.append(l0_norm(model.coef_))

         plt.scatter(alphas, l0_norms)
         plt.title('Alpha v. L0 Norm')
         plt.xlabel('Alpha')
         plt.ylabel('L0 Norm')
```

```
Out[95]: <matplotlib.text.Text at 0x113907e80>
```



3.0.4 Part 4

Add the outputs of your models as features and train a ridge regression on all the features plus the model outputs (this is called ensembling and stacking). Be careful not to overfit. What score can you get?

In [73]:
```
ridge_preds = model_ridge.predict(X_train)
lasso_preds = model_lasso.predict(X_train)

X_train['RidgeModel'] = ridge_preds
X_train['LassoModel'] = lasso_preds
X_train.head()
```

```
/Users/rohannagar/anaconda/lib/python3.5/site-packages/ipykernel/__main__.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexin
/Users/rohannagar/anaconda/lib/python3.5/site-packages/ipykernel/__main__.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexin
```

Out[73]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	\
0	4.110874	4.189655	9.042040	7	5	2003	
1	3.044522	4.394449	9.169623	6	8	1976	
2	4.110874	4.234107	9.328212	7	5	2001	
3	4.262680	4.110874	9.164401	7	5	1915	
4	4.110874	4.442651	9.565284	8	5	2000	

	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	SaleType_Oth	\
0	2003	5.283204	6.561031	0	...	0	
1	1976	0.000000	6.886532	0	...	0	
2	2002	5.093750	6.188264	0	...	0	
3	1970	0.000000	5.379897	0	...	0	
4	2000	5.860786	6.486161	0	...	0	

	SaleType_WD	SaleCondition_Abnorml	SaleCondition_AdjLand	\
0	1	0	0	
1	1	0	0	
2	1	0	0	
3	1	1	0	
4	1	0	0	

	SaleCondition_Alloca	SaleCondition_Family	SaleCondition_Normal	\
0	0	0	1	
1	0	0	1	
2	0	0	1	
3	0	0	0	
4	0	0	1	

	SaleCondition_Partial	RidgeModel	LassoModel
0	0	12.243911	12.244881
1	0	12.178929	12.160932
2	0	12.288364	12.294685

```
3                        0   12.033395   12.060877
4                        0   12.607881   12.616734

[5 rows x 290 columns]
```

```
In [74]: model_stacked_ridge = RidgeCV(alphas= [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]).fit(X_tra

         ridge_score = -cross_val_score(model_stacked_ridge, X_train, y, scoring="mean_squared_error", 
         print("Stacked Ridge Best RMSE: {}".format(np.sqrt(ridge_score).mean()))
```

```
Stacked Ridge Best RMSE: 0.12407650980109419
```

3.0.5 Part 5

Install XGBoost and train a gradient boosting regression. What score can you get just from a single XGB? (you will need to optimize over its parameters).

```
In [18]: from sklearn.grid_search import RandomizedSearchCV, GridSearchCV

         # Grid search XGB
         parameters = {
             'max_depth': [1, 2, 4, 8],
             'learning_rate': [0.001, 0.01, 0.1, 0.3],
             'n_estimators': [50, 150, 250, 500, 1000]
         }

         xg_clf = GridSearchCV(xgb.XGBRegressor(), parameters, cv=5, n_jobs=-1, scoring='mean_squared_e
         xg_clf.fit(X_train, y)

         print("Best parameter set found on development set with cv=10:\n")
         print(np.sqrt(-xg_clf.best_score_))
         print(xg_clf.best_params_)
         print()
```

```
Best parameter set found on development set with cv=10:

0.124412774751
{'learning_rate': 0.1, 'n_estimators': 1000, 'max_depth': 2}
```

The best score was 0.124412774751.

3.0.6 Part 6

Do your best to win. Try feature engineering and stacking many models. You are allowed to use any public tool in Python. No nonpython tools allowed.

```
In [21]: # Drop some features that were unimportant according to Kaggle forum.
         X_train_dropped = X_train.drop(['MiscVal', 'BsmtHalfBath', 'BsmtFinSF2'], axis=1)
         X_train_dropped.head()
```

```
Out[21]:    MSSubClass  LotFrontage   LotArea  OverallQual  OverallCond  YearBuilt  \
         0    4.110874     4.189655  9.042040            7            5       2003
         1    3.044522     4.394449  9.169623            6            8       1976
         2    4.110874     4.234107  9.328212            7            5       2001
         3    4.262680     4.110874  9.164401            7            5       1915
         4    4.110874     4.442651  9.565284            8            5       2000
```

```
   YearRemodAdd  MasVnrArea  BsmtFinSF1  BsmtUnfSF          ...            \
0          2003    5.283204    6.561031   5.017280          ...
1          1976    0.000000    6.886532   5.652489          ...
2          2002    5.093750    6.188264   6.075346          ...
3          1970    0.000000    5.379897   6.293419          ...
4          2000    5.860786    6.486161   6.196444          ...

   SaleType_ConLw  SaleType_New  SaleType_Oth  SaleType_WD  \
0               0             0             0            1
1               0             0             0            1
2               0             0             0            1
3               0             0             0            1
4               0             0             0            1

   SaleCondition_Abnorml  SaleCondition_AdjLand  SaleCondition_Alloca  \
0                      0                      0                     0
1                      0                      0                     0
2                      0                      0                     0
3                      1                      0                     0
4                      0                      0                     0

   SaleCondition_Family  SaleCondition_Normal  SaleCondition_Partial
0                     0                     1                      0
1                     0                     1                      0
2                     0                     1                      0
3                     0                     0                      0
4                     0                     1                      0

[5 rows x 285 columns]
```

In [22]:
```
# Try to train Ridge on this and see if it is better.
model_feature_engr_ridge = RidgeCV(alphas= [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]).fit(

ridge_score = -cross_val_score(model_feature_engr_ridge, X_train_dropped, y, scoring="mean_squa
print("Feature Engineered Ridge Best RMSE: {}".format(np.sqrt(ridge_score).mean()))
```

```
Feature Engineered Ridge Best RMSE: 0.12813056242722962
```

Clearly, this didn't work. Let's try something else. Increase the influence of the more important features.

In [23]:
```
X_train['OverallQual'] = X_train['OverallQual'] ** 2
X_train.head()
```

```
/Users/rohannagar/anaconda/lib/python3.5/site-packages/ipykernel/__main__.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexin
  if __name__ == '__main__':
```

Out[23]:
```
   MSSubClass  LotFrontage   LotArea  OverallQual  OverallCond  YearBuilt  \
0    4.110874     4.189655  9.042040           49            5       2003
1    3.044522     4.394449  9.169623           36            8       1976
2    4.110874     4.234107  9.328212           49            5       2001
```

```
3    4.262680     4.110874  9.164401           49            5       1915
4    4.110874     4.442651  9.565284           64            5       2000

   YearRemodAdd  MasVnrArea  BsmtFinSF1  BsmtFinSF2          ...            \
0          2003    5.283204    6.561031           0          ...
1          1976    0.000000    6.886532           0          ...
2          2002    5.093750    6.188264           0          ...
3          1970    0.000000    5.379897           0          ...
4          2000    5.860786    6.486161           0          ...

   SaleType_ConLw  SaleType_New  SaleType_Oth  SaleType_WD  \
0               0             0             0            1
1               0             0             0            1
2               0             0             0            1
3               0             0             0            1
4               0             0             0            1

   SaleCondition_Abnorml  SaleCondition_AdjLand  SaleCondition_Alloca  \
0                      0                      0                     0
1                      0                      0                     0
2                      0                      0                     0
3                      1                      0                     0
4                      0                      0                     0

   SaleCondition_Family  SaleCondition_Normal  SaleCondition_Partial
0                     0                     1                      0
1                     0                     1                      0
2                     0                     1                      0
3                     0                     0                      0
4                     0                     1                      0

[5 rows x 288 columns]
```

In [24]:
```python
model_feature_engr2_ridge = RidgeCV(alphas= [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]).fit

ridge_score = -cross_val_score(model_feature_engr2_ridge, X_train, y, scoring="mean_squared_er
print("Feature Engineered Ridge Best RMSE: {}".format(np.sqrt(ridge_score).mean()))
```

```
Feature Engineered Ridge Best RMSE: 0.1282003242765391
```

Didn't help much either. Let's try stacking XGB on top.

In [74]:
```python
xg = xgb.XGBRegressor(learning_rate=0.1, n_estimators=1000, max_depth=2)
xg.fit(X_train, y)
xg_preds = xg.predict(X_train)

X_train_with_xg = X_train
X_train_with_xg['xg_pred'] = xg_preds
X_train_with_xg.head()
```

```
/Users/rohannagar/anaconda/lib/python3.5/site-packages/ipykernel/__main__.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexin
```

```
Out[74]:    MSSubClass  LotFrontage   LotArea  OverallQual  OverallCond  YearBuilt  \
         0    4.110874     4.189655  9.042040            7            5       2003
         1    3.044522     4.394449  9.169623            6            8       1976
         2    4.110874     4.234107  9.328212            7            5       2001
         3    4.262680     4.110874  9.164401            7            5       1915
         4    4.110874     4.442651  9.565284            8            5       2000

            YearRemodAdd  MasVnrArea  BsmtFinSF1  BsmtFinSF2    ...      SaleType_New  \
         0          2003    5.283204    6.561031           0    ...                 0
         1          1976    0.000000    6.886532           0    ...                 0
         2          2002    5.093750    6.188264           0    ...                 0
         3          1970    0.000000    5.379897           0    ...                 0
         4          2000    5.860786    6.486161           0    ...                 0

            SaleType_Oth  SaleType_WD  SaleCondition_Abnorml  SaleCondition_AdjLand  \
         0             0            1                      0                      0
         1             0            1                      0                      0
         2             0            1                      0                      0
         3             0            1                      1                      0
         4             0            1                      0                      0

            SaleCondition_Alloca  SaleCondition_Family  SaleCondition_Normal  \
         0                     0                     0                     1
         1                     0                     0                     1
         2                     0                     0                     1
         3                     0                     0                     0
         4                     0                     0                     1

            SaleCondition_Partial    xg_pred
         0                      0  12.230842
         1                      0  12.099054
         2                      0  12.255805
         3                      0  11.905243
         4                      0  12.534455

         [5 rows x 289 columns]
```

```
In [75]: model_xg_ridge = RidgeCV(alphas= [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]).fit(X_train_wi

         ridge_score = -cross_val_score(model_xg_ridge, X_train_with_xg, y, scoring="mean_squared_error
         print("XG-Ridge Best RMSE: {}".format(np.sqrt(ridge_score).mean()))
```

```
XG-Ridge Best RMSE: 0.05833903058041286
```

Seems to be really good! Maybe overfitting? Let's try to submit and see.

```
In [76]: xg_test_preds = xg.predict(X_test)

         X_test_with_xg = X_test
         X_test_with_xg['xg_pred'] = xg_test_preds
         X_test_with_xg.head()
```

```
/Users/rohannagar/anaconda/lib/python3.5/site-packages/ipykernel/__main__.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing
```

Out[76]:

```
   MSSubClass  LotFrontage   LotArea  OverallQual  OverallCond  YearBuilt  \
0    3.044522     4.394449  9.360741            5            6       1961
1    3.044522     4.406719  9.565775            6            6       1958
2    4.110874     4.317488  9.534668            5            5       1997
3    4.110874     4.369448  9.208238            6            6       1998
4    4.795791     3.784190  8.518392            8            5       1992

   YearRemodAdd  MasVnrArea  BsmtFinSF1  BsmtFinSF2     ...      SaleType_New  \
0          1961    0.000000    6.150603    4.976734     ...                 0
1          1958    4.691348    6.828712    0.000000     ...                 0
2          1998    0.000000    6.674561    0.000000     ...                 0
3          1998    3.044522    6.401917    0.000000     ...                 0
4          1992    0.000000    5.575949    0.000000     ...                 0

   SaleType_Oth  SaleType_WD  SaleCondition_Abnorml  SaleCondition_AdjLand  \
0             0            1                      0                      0
1             0            1                      0                      0
2             0            1                      0                      0
3             0            1                      0                      0
4             0            1                      0                      0

   SaleCondition_Alloca  SaleCondition_Family  SaleCondition_Normal  \
0                     0                     0                     1
1                     0                     0                     1
2                     0                     0                     1
3                     0                     0                     1
4                     0                     0                     1

   SaleCondition_Partial    xg_pred
0                      0  11.708064
1                      0  11.985591
2                      0  12.145906
3                      0  12.185511
4                      0  12.118196

[5 rows x 289 columns]
```

In [77]:

```python
preds = np.expm1(model_xg_ridge.predict(X_test_with_xg))

solution = pd.DataFrame({"id":test.Id, "SalePrice":preds})
solution.to_csv("xg_ridge_solution.csv", index = False)
```

Our public score was 0.14288, so it looks like we were overfitting. Let's try ElasticNet.

In [96]:

```python
from sklearn.linear_model import ElasticNetCV

elastic_model = ElasticNetCV(l1_ratio=[.1, .5, .7, .9, .95, .99, 1], alphas=[0.05, 0.1, 0.3, 1
elastic_score = -cross_val_score(elastic_model, X_train, y, scoring="mean_squared_error", cv =
print("Elastic Best RMSE: {}".format(np.sqrt(elastic_score).mean()))
```

```
Elastic Best RMSE: 0.14732580868817688
```

Pretty good! This is our best yet, but let's try to stack one more time.

```
In [82]: elastic_preds = elastic_model.predict(X_train)

         X_train_with_elastic = X_train
         X_train_with_elastic['elastic_pred'] = elastic_preds
         X_train_with_elastic.head()
```

```
/Users/rohannagar/anaconda/lib/python3.5/site-packages/ipykernel/__main__.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing
```

```
Out[82]:    MSSubClass  LotFrontage   LotArea  OverallQual  OverallCond  YearBuilt  \
         0    4.110874     4.189655  9.042040            7            5       2003
         1    3.044522     4.394449  9.169623            6            8       1976
         2    4.110874     4.234107  9.328212            7            5       2001
         3    4.262680     4.110874  9.164401            7            5       1915
         4    4.110874     4.442651  9.565284            8            5       2000

            YearRemodAdd  MasVnrArea  BsmtFinSF1  BsmtFinSF2      ...       \
         0          2003    5.283204    6.561031           0      ...
         1          1976    0.000000    6.886532           0      ...
         2          2002    5.093750    6.188264           0      ...
         3          1970    0.000000    5.379897           0      ...
         4          2000    5.860786    6.486161           0      ...

            SaleType_Oth  SaleType_WD  SaleCondition_Abnorml  SaleCondition_AdjLand  \
         0             0            1                      0                      0
         1             0            1                      0                      0
         2             0            1                      0                      0
         3             0            1                      1                      0
         4             0            1                      0                      0

            SaleCondition_Alloca  SaleCondition_Family  SaleCondition_Normal  \
         0                     0                     0                     1
         1                     0                     0                     1
         2                     0                     0                     1
         3                     0                     0                     0
         4                     0                     0                     1

            SaleCondition_Partial    xg_pred  elastic_pred
         0                      0  12.279172     12.291481
         1                      0  12.119685     12.124940
         2                      0  12.283498     12.290551
         3                      0  11.983372     12.003270
         4                      0  12.617622     12.638802

         [5 rows x 290 columns]
```

```
In [84]: model_elastic_ridge = RidgeCV(alphas= [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]).fit(X_trai

         ridge_score = -cross_val_score(model_elastic_ridge, X_train_with_elastic, y, scoring="mean_squa
         print("Elastic-Ridge Best RMSE: {}".format(np.sqrt(ridge_score).mean()))
```

```
Elastic-Ridge Best RMSE: 0.06101284414457284
```

```
In [85]: elastic_test_preds = elastic_model.predict(X_test)

         X_test_with_elastic = X_test
         X_test_with_elastic['elastic_pred'] = elastic_test_preds
         X_test_with_elastic.head()
```

```
/Users/rohannagar/anaconda/lib/python3.5/site-packages/ipykernel/__main__.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing
```

```
Out[85]:    MSSubClass  LotFrontage   LotArea  OverallQual  OverallCond  YearBuilt  \
         0    3.044522     4.394449  9.360741            5            6       1961
         1    3.044522     4.406719  9.565775            6            6       1958
         2    4.110874     4.317488  9.534668            5            5       1997
         3    4.110874     4.369448  9.208238            6            6       1998
         4    4.795791     3.784190  8.518392            8            5       1992

            YearRemodAdd  MasVnrArea  BsmtFinSF1  BsmtFinSF2      ...       \
         0          1961    0.000000    6.150603    4.976734      ...
         1          1958    4.691348    6.828712    0.000000      ...
         2          1998    0.000000    6.674561    0.000000      ...
         3          1998    3.044522    6.401917    0.000000      ...
         4          1992    0.000000    5.575949    0.000000      ...

            SaleType_Oth  SaleType_WD  SaleCondition_Abnorml  SaleCondition_AdjLand  \
         0             0            1                      0                      0
         1             0            1                      0                      0
         2             0            1                      0                      0
         3             0            1                      0                      0
         4             0            1                      0                      0

            SaleCondition_Alloca  SaleCondition_Family  SaleCondition_Normal  \
         0                     0                     0                     1
         1                     0                     0                     1
         2                     0                     0                     1
         3                     0                     0                     1
         4                     0                     0                     1

            SaleCondition_Partial    xg_pred  elastic_pred
         0                      0  11.708064     11.843735
         1                      0  11.985591     11.935026
         2                      0  12.145906     12.091564
         3                      0  12.185511     12.218478
         4                      0  12.118196     12.186267

         [5 rows x 290 columns]
```

```
In [86]: preds = np.expm1(model_elastic_ridge.predict(X_test_with_elastic))

         solution = pd.DataFrame({"id":test.Id, "SalePrice":preds})
         solution.to_csv("elastic_ridge_solution.csv", index = False)
```

Wow, this submission was really bad. Clearly out stacking attempt didn't work. Let's just submit the ElasticNet model and see our score.

```
In [97]: preds = np.expm1(elastic_model.predict(X_test))

         solution = pd.DataFrame({"id":test.Id, "SalePrice":preds})
         solution.to_csv("elastic_solution.csv", index = False)
```

Well, it looks like this didn't beat the score we get from a single Lasso model. We get a 0.14801 from a single ElasticNet model.

3.0.7 Part 7

Read (and post) in the Kaggle forums. Include in your report if you find something in the forums that you like, or if you made a post or code, especially if other Kagglers used it afterwards.

3.0.8 Answer

There were a few posts that we found that we really liked. The first is a post that does very detailed exploration of the given data. https://www.kaggle.com/xchmiao/house-prices-advanced-regression-techniques/detailed-data-exploration-in-python. We liked this because the details learned about the features can help us determine which features are important to training a model and which are not. It can show which features could potentially have more influence on the prediction. It also helps understand the data and what each feature means, both by itself and in context with others.

Additionally, this post was very impressive. https://www.kaggle.com/klyusba/house-prices-advanced-regression-techniques/lasso-model-for-regression-problem. The Kaggler uses Lasso regression but is able to get a score of 0.11720. We see a lot of preprocessing, feature selection, and feature extraction used. These examples are very useful and can help us see how to perform good feature selection and extaction to get a very good score even just using a simpler linear model like lasso.

3.0.9 Part 8

Be sure you do not violate the rules of Kaggle! No sharing of code or data outside of the Kaggle forums. Every student should have their own individual Kaggle account and teams can be formed in the Kaggle submissions with your lab partner.

3.0.10 Answer

We have followed the rules! Our team name is `DataScienceSquad` and our Kaggle usernames are `RohanNagar` and `aetherzephyr`.

3.0.11 Part 9

You will be graded based on your public score (include that in your report) and also on the creativity of your solution. In your report, explain what worked and what did not work. Many creative things will not work, but you will get partial credit for developing them. We will invite teams with interesting solutions to present them in class.

3.0.12 Answer

Our public score (screenshot below): 0.12097

One of the things we tried was removing some features that seemed to not have much of an impact on the Housing Price, according to a forum post on Kaggle. However, trying this and running a Ridge model resulted in a worse score. So, this didn't work well. We also tried to change the `OverallQual` feature by raising it to the power of 2, since that feature had a big influence on the price. This didn't work very well either.

Next, we tried running a stacked model with an XGB layer first and then a Ridge regression. This got a better score on the training set, but when we submitted it our public score was not very good. We believe that this overfit the data.

Then, we tried an ElasticNet model. This performed really well on the training data, but not as well on the test data by itself. It was worse than a single Lasso model. We also tried to stack it and run a Ridge regression, but this performed very poorly.

Overall, we were not able to improve on the single Lasso model score with the time that we had. We learned a lot about XGB, feature engineering, stacking, and ElasticNet. Unforuntely, we either didn't stack them in the right way, or the things we tried just didn't work.

In [98]:
```
from IPython.display import Image
Image(filename='leaderboard.png')
```

Out[98]:

66	↓28	senkin13	0.12097	9	Mon, 26 Sep 2016 15:37:36 (-6d)
67	↓28	Arvind Sundaram	0.12097	6	Tue, 27 Sep 2016 18:52:27 (-6.9d)
68	new	JackRobin	0.12097	1	Wed, 21 Sep 2016 07:25:59
69	new	Grandrew	0.12097	3	Tue, 27 Sep 2016 05:28:41 (-0.2h)
70	new	MunmunChowdhury	0.12097	5	Tue, 27 Sep 2016 17:33:10 (-3.6h)
71	↓31	wittmaan	0.12097	2	Thu, 15 Sep 2016 17:45:28 (-0.2h)
72	↓31	ChristopherFrazier	0.12097	8	Tue, 27 Sep 2016 21:03:11 (-7.1d)
73	**↑63**	**DataScienceSquad**	**0.12097**	**6**	**Tue, 27 Sep 2016 21:53:24 (-6.9d)**
74	new	shegokarm	0.12097	2	Thu, 22 Sep 2016 12:13:33 (-1.2h)
75	↑256	jp1976	0.12097	5	Wed, 21 Sep 2016 21:26:14
76	↓33	Alexandru Papiu	0.12097	8	Fri, 23 Sep 2016 18:48:25 (-11.4d)
77	↓33	Ofd	0.12097	4	Tue, 13 Sep 2016 01:19:56 (-0.6h)
78	↓31	crownpku	0.12097	2	Tue, 20 Sep 2016 09:29:31
79	new	Алексей Кораи̮	0.12097	1	Wed, 21 Sep 2016 11:11:38