Final Project

Zining Wang, Wenxuan Wang, Wenda Zheng

First, we read and preprocess the data:

In [69]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
from scipy.stats import skew
from scipy.stats.stats import pearsonr

%config InlineBackend.figure_format = 'png'
%matplotlib inline

train = pd.read_csv("./data/train.csv")
test = pd.read_csv("./data/test.csv")
testID = test.Id
train.head()

print train.shape

# from collections import Counter
# Counter(train.MiscFeature)
# Adjust House Price based on CPI index, Convert to 2010 December dollar
s (CPI indices are from Bureau of Labor Statistics)
"""
train.ix[(train.YrSold == 2010) &
         ((train.MoSold == 7)|(train.MoSold == 6)|(train.MoSold <= 4)),
         'SalePrice'] = train.SalePrice * 1.01
train.ix[(train.YrSold == 2009) &
         ((train.MoSold == 1)|(train.MoSold == 6)|(train.MoSold <= 4)),
         'SalePrice'] = train.SalePrice * 1.04
train.ix[(train.YrSold == 2009) &
         ((train.MoSold == 2)|(train.MoSold == 3)|(train.MoSold == 4)),
         'SalePrice'] = train.SalePrice * 1.03
train.ix[(train.YrSold == 2009) &
         ((train.MoSold == 5)|(train.MoSold == 6)|(train.MoSold == 7)|(t
rain.MoSold == 8)),
         'SalePrice'] = train.SalePrice * 1.02
train.ix[(train.YrSold == 2009) &
         ((train.MoSold >= 9)),
         'SalePrice'] = train.SalePrice * 1.01
train.ix[(train.YrSold == 2008) &
         ((train.MoSold == 1)|(train.MoSold == 12)),
         'SalePrice'] = train.SalePrice * 1.04
train.ix[(train.YrSold == 2008) &
```

```
         ((train.MoSold == 3)|(train.MoSold == 11)),
         'SalePrice'] = train.SalePrice * 1.03
train.ix[(train.YrSold == 2008) &
         ((train.MoSold == 5)|(train.MoSold == 10)),
         'SalePrice'] = train.SalePrice * 1.01
train.ix[(train.YrSold == 2007) &
         ((train.MoSold == 1)|(train.MoSold == 2)),
         'SalePrice'] = train.SalePrice * 1.08
train.ix[(train.YrSold == 2007) &
         ((train.MoSold == 3)),
         'SalePrice'] = train.SalePrice * 1.07
train.ix[(train.YrSold == 2007) &
         ((train.MoSold == 4)),
         'SalePrice'] = train.SalePrice * 1.06
train.ix[(train.YrSold == 2007) &
         ((train.MoSold == 5)|(train.MoSold == 6)|(train.MoSold == 7)|(t
rain.MoSold == 8)|(train.MoSold == 9)|(train.MoSold == 10)),
         'SalePrice'] = train.SalePrice * 1.05
train.ix[(train.YrSold == 2007) &
         ((train.MoSold == 11)|(train.MoSold == 12)),
         'SalePrice'] = train.SalePrice * 1.04
train.ix[(train.YrSold == 2006) &
         ((train.MoSold == 1)),
         'SalePrice'] = train.SalePrice * 1.11
train.ix[(train.YrSold == 2006) &
         ((train.MoSold == 2)|(train.MoSold == 3)),
         'SalePrice'] = train.SalePrice * 1.10
train.ix[(train.YrSold == 2006) &
         ((train.MoSold == 4)|(train.MoSold >= 10)),
         'SalePrice'] = train.SalePrice * 1.09
train.ix[(train.YrSold == 2006) &
         ((train.MoSold == 5)|(train.MoSold == 6)|(train.MoSold == 7)|(t
rain.MoSold == 9)),
         'SalePrice'] = train.SalePrice * 1.08
train.ix[(train.YrSold == 2006) &
         ((train.MoSold == 8)),
         'SalePrice'] = train.SalePrice * 1.07

"""
# Converting features and filling missing values...
train['MSSubClass'] = train['MSSubClass'].astype(str)
test['MSSubClass'] = test['MSSubClass'].astype(str)
test['MSZoning'] = test['MSZoning'].fillna(train['MSZoning'].mode()[0])
train['LotFrontage'] =
train['LotFrontage'].fillna(train['LotFrontage'].mean())
test['LotFrontage'] = test['LotFrontage'].fillna(train['LotFrontage'].me
an())
train['Alley'] = train['Alley'].fillna('NoAlleyAccess')
test['Alley'] = test['Alley'].fillna('NoAlleyAccess')
train['MasVnrType'] = train['MasVnrType'].fillna(train['MasVnrType'].mod
e()[0])
test['MasVnrType'] =
test['MasVnrType'].fillna(train['MasVnrType'].mode()[0])

# Impute test data with the most common category
for col in ('BsmtFullBath','BsmtHalfBath','Exterior1st','Exterior2nd','F
unctional'):
```

```
    test[col] = test[col].fillna(train[col].mode()[0])
# Impute test data with mean
test['BsmtUnfSF'] = test['BsmtUnfSF'].fillna(train['BsmtUnfSF'].mean())

train['Fence'] = train['Fence'].fillna('NoFence')
test['Fence'] = test['Fence'].fillna('NoFence')

for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'Bsm
tFinType2'):
    train[col] = train[col].fillna('NoBasement')
    test[col] = test[col].fillna('NoBasement')
for col in ('BsmtFinSF1','BsmtFinSF2'):
    test[col] = test[col].fillna(0.0)
test['TotalBsmtSF'] = test['TotalBsmtSF'].fillna(0)
train['Electrical'] = train['Electrical'].fillna(train['Electrical'].mod
e()[0])
test['KitchenQual'] = test['KitchenQual'].fillna(train['KitchenQual'].mo
de()[0])
train['FireplaceQu'] = train['FireplaceQu'].fillna('NoFirePlace')
test['FireplaceQu'] = test['FireplaceQu'].fillna('NoFirePlace')
train['PoolQC'] = train['PoolQC'].fillna('NoPool')
test['PoolQC'] = test['PoolQC'].fillna('NoPool')
train['MiscFeature'] = train['MiscFeature'].fillna('NoMisc')
test['MiscFeature'] = test['PoolQC'].fillna('NoMisc')

for col in ('GarageType', 'GarageFinish', 'GarageQual','GarageCond','Gar
ageYrBlt'):
    train[col] = train[col].fillna('NoGarage')
    test[col] = test[col].fillna('NoGarage')
test['GarageCars'] = test['GarageCars'].fillna(0.0)
test['GarageArea'] = test['GarageArea'].fillna(0.0)

train['MasVnrArea'] = train['MasVnrArea'].fillna(0.0)
test['MasVnrArea'] = test['MasVnrArea'].fillna(0.0)


train['YrSold'] = train['YrSold'].astype(str)
test['MoSold'] = test['MoSold'].astype(str)


test['SaleType'] = test['SaleType'].fillna(train['SaleType'].mode()[0])


train = train.drop('Id',1)
test = test.drop('Id',1)
train = train.drop('Utilities',1)
test = test.drop('Utilities',1)

#print train.SalePrice


train_len = len(train)

trainX = train.drop('SalePrice',1)
trainX = train[:int(train_len * 0.75)]
testX = train[int(train_len * 0.75):]
```

```
trainY = train.SalePrice[:int(train_len * 0.75)]
testY = train.SalePrice[int(train_len * 0.75):]

"""
trainX = train.drop('SalePrice',1)
trainY = train.SalePrice
testX = test
"""

trainX.head()
```

```
(1460, 81)
```

Out[69]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	Land(
0	60	RL	65.0	8450	Pave	NoAlleyAccess	Reg	Lvl
1	20	RL	80.0	9600	Pave	NoAlleyAccess	Reg	Lvl
2	60	RL	68.0	11250	Pave	NoAlleyAccess	IR1	Lvl
3	70	RL	60.0	9550	Pave	NoAlleyAccess	IR1	Lvl
4	60	RL	84.0	14260	Pave	NoAlleyAccess	IR1	Lvl

5 rows × 79 columns

Dummy code all categorical variables(46 out of 79 variables are categorical)

In [70]:
```
#print train.LotArea.dtype
count = 0
train_len = len(train)
alldata = pd.concat(objs=[trainX, testX], axis=0)
for col in alldata.columns:
    if alldata[col].dtype != 'int64' and alldata[col].dtype !=
'float64':
        #count += 1
        #print 'The attribute', col, 'is',alldata[col].dtype, ' not nume
rical types. So we will drop it..'

        # concatenate the dummy variables and drop the duplicates
        alldata =
pd.concat([alldata,pd.get_dummies(alldata[col]).iloc[:, 1:]], axis=1)
        alldata = alldata.drop(col,1)
    else:
        Xmin = min(alldata[:train_len][col])
        Xmax = max(alldata[:train_len][col])
        alldata[col] = [(x - Xmin+0.0)/(Xmax - Xmin) for x in alldata[co
l]]
# train_preprocessed = dataset_preprocessed[:train_objs_num]
# test_preprocessed = dataset_preprocessed[train_objs_num:]
#print count


trainX = alldata[:int(train_len * 0.75)]
trainX_measure = alldata[int(train_len * 0.75): train_len]
testX = alldata[int(train_len * 0.75):]

"""
trainX = alldata[:train_len]
testX = alldata[train_len:]
"""

print alldata.columns
alldata.head()
```

```
Index([ u'LotFrontage',      u'LotArea',  u'OverallQual',  u'OverallCon
d',
         u'YearBuilt', u'YearRemodAdd',   u'MasVnrArea',   u'BsmtFinSF
1',
        u'BsmtFinSF2',    u'BsmtUnfSF',
       ...
            u'ConLI',        u'ConLw',          u'New',          u'Ot
h',
               u'WD',      u'AdjLand',       u'Alloca',       u'Famil
y',
           u'Normal',      u'Partial'],
      dtype='object', length=371)
```

Out[70]:

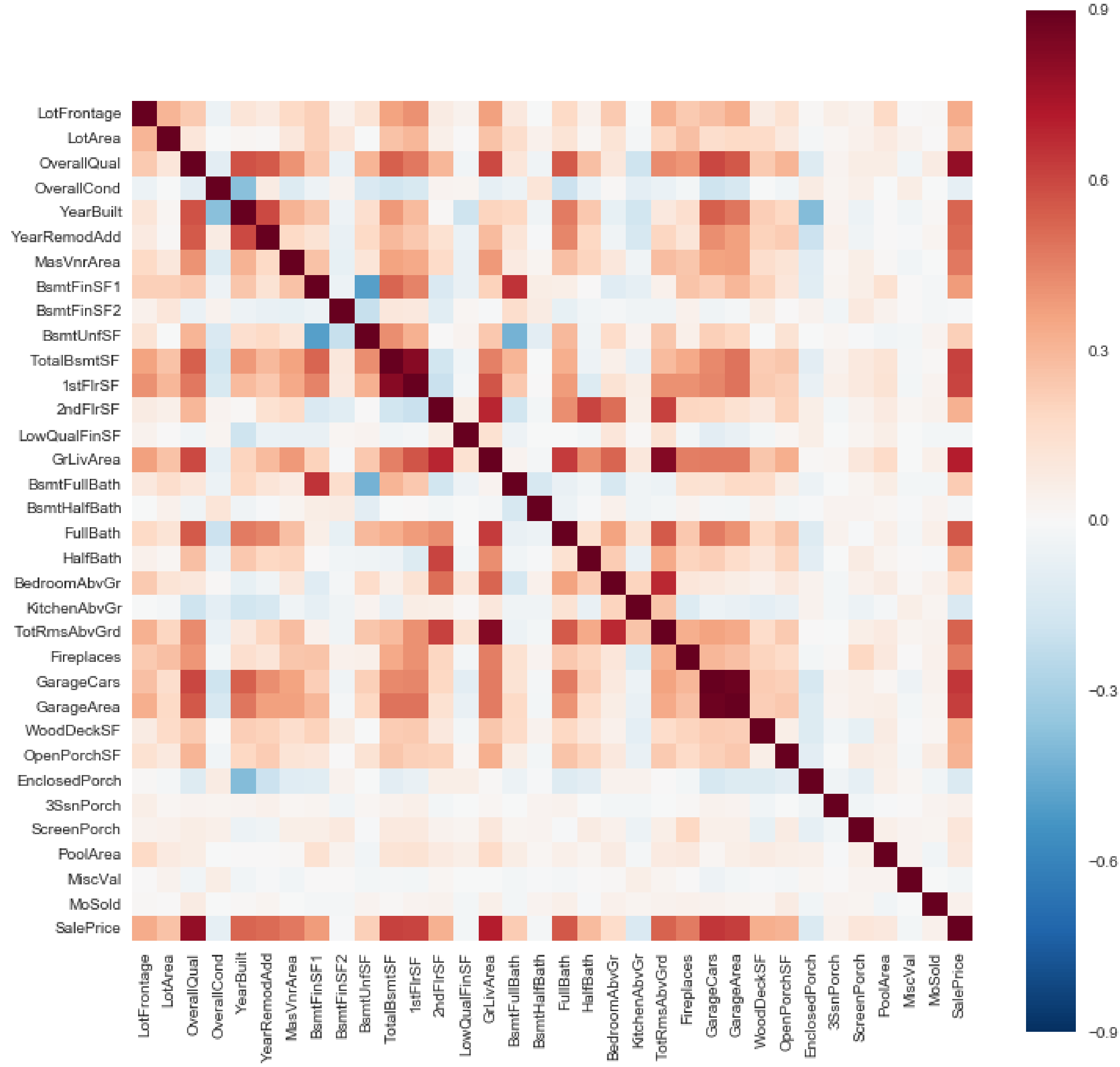
	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrAr
0	0.150685	0.033420	0.666667	0.500	0.949275	0.883333	0.12250
1	0.202055	0.038795	0.555556	0.875	0.753623	0.433333	0.00000
2	0.160959	0.046507	0.666667	0.500	0.934783	0.866667	0.10125
3	0.133562	0.038561	0.666667	0.500	0.311594	0.333333	0.00000
4	0.215753	0.060576	0.777778	0.500	0.927536	0.833333	0.21875

5 rows × 371 columns

Do some basic plots to see correlations

In [71]:
```
# # Check numbers of NA..
# NAs = pd.concat([train.isnull().sum(), test.isnull().sum()], axis=1, k
eys=['Train', 'Test'])
# print NAs[NAs.sum(axis=1) > 0]

# # drop columns with over 500 missing values...
# for mis in NAs[NAs.sum(axis=1) > 500].index:
#     train = train.drop(mis,1)

# Check numbers of NA..
# NAs = pd.concat([train.isnull().sum(), test.isnull().sum()], axis=1, k
eys=['Train', 'Test'])
# print NAs[NAs.Train > 0]

# Plot the correlation of Ground Living Area
# Make a correlation map to determine which features are not very correl
ated with SalePrice
corrmat = train.corr()
corrmat.head()
plt.subplots(figsize=(12,12))
sns.heatmap(corrmat, vmax=0.9, square=True)
```

Out[71]: <matplotlib.axes._subplots.AxesSubplot at 0x14b5d8d0>



In [72]:
```
# See top 15 most important numerical features
#Contributed by Wenxuan
corrmat_val = corrmat.ix['SalePrice']
corrmat_val.sort_values(inplace = True, ascending = False)
most_correlated = corrmat_val[0:16]
most_correlated
core_attributes = []
for x in most_correlated.index:
    core_attributes.append(x)
train_core = train[[x for x in core_attributes]]
train_core.head()
```

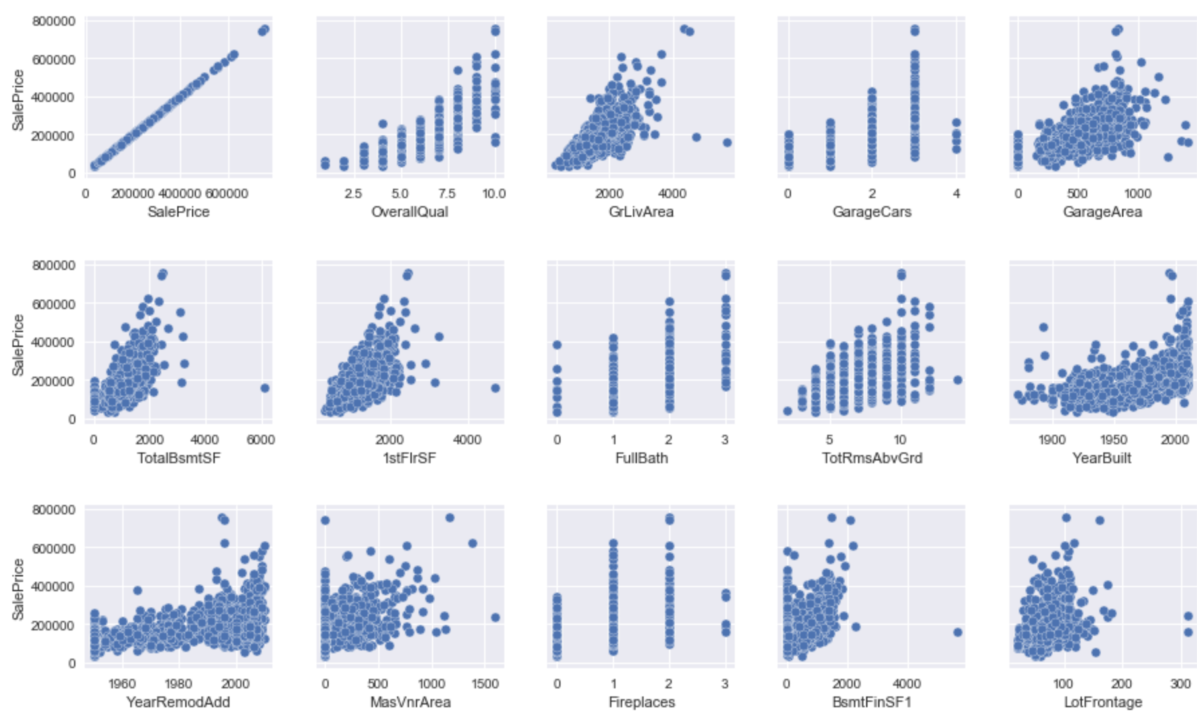
Out[72]:

	SalePrice	OverallQual	GrLivArea	GarageCars	GarageArea	TotalBsmtSF	1stFlrSF	Fu
0	208500	7	1710	2	548	856	856	2
1	181500	6	1262	2	460	1262	1262	2
2	223500	7	1786	2	608	920	920	2
3	140000	7	1717	3	642	756	961	1
4	250000	8	2198	3	836	1145	1145	2

Run Regularized Linear Regression on the selected attributes. We evaluate scoring metrics using mean squared error.

In [73]:
```
print 'Pairwise Correlation'
sns.set()
attributes = []
for i in xrange(5):
    attributes.append(core_attributes[i])
sns.pairplot(data=train,
                  x_vars=attributes,
                  y_vars=['SalePrice'])
plt.show()

sns.set()
attributes = []
for i in xrange(5,10):
    attributes.append(core_attributes[i])
sns.pairplot(data=train,
                  x_vars=attributes,
                  y_vars=['SalePrice'])
plt.show()

sns.set()
attributes = []
for i in xrange(10,15):
    attributes.append(core_attributes[i])
sns.pairplot(data=train,
                  x_vars=attributes,
                  y_vars=['SalePrice'])
plt.show()
```

```
Pairwise Correlation
```

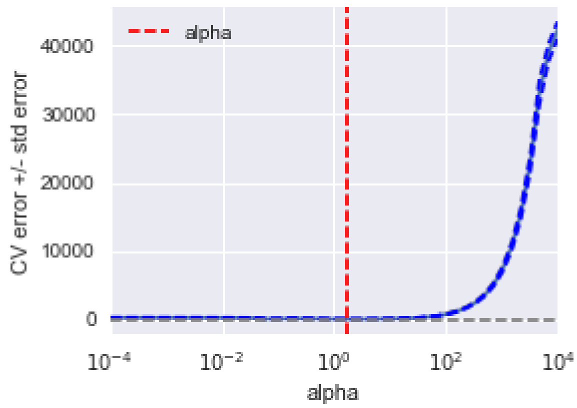


In [74]:
```
# function to estimate alpha using cross validation
from sklearn import metrics
from sklearn import linear_model
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
```

```
def estimate_alpha(alpha_list, n_folds):
    scores = list()
    scores_std = list()
    min_score = 100000
    # run the the list of alphas
    for alpha in alpha_list:
        lassoModel = linear_model.Lasso(alpha=alpha)
        this_scores = -cross_val_score(lassoModel, trainX, trainY, scori
ng="neg_mean_absolute_error", cv=n_folds, n_jobs=1)
        scores.append(np.mean(this_scores))
        scores_std.append(np.std(this_scores))

    # find the minimum of the scores and the index

    optAlphaIdx = np.argmin(scores)
    optAlpha = alpha_list[optAlphaIdx]
    lowerBound = scores[optAlphaIdx] +
(scores_std[optAlphaIdx]/np.sqrt(n_folds))
    # get the smallest alpha within +/- std error
    for i, alpha in enumerate(alpha_list):
        if scores[i] <= lowerBound and i>optAlphaIdx:
            oneStdAlpha = alpha

            break
    return scores, scores_std, optAlpha, oneStdAlpha

# function to plot the cross-validation error curve
def plot_cv_curve(alphas, scores, scores_std, optAlpha, n_folds):
    scores, scores_std = np.array(scores), np.array(scores_std)
    plt.figure().set_size_inches(4, 3)
    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, alp
ha=0.2)
    plt.ylabel('CV error +/- std error')
    plt.xlabel('alpha')
    plt.axhline(np.min(scores), linestyle='--', color='.5')
    plt.axvline(optAlpha, linestyle='--', color='r', label='alpha')
    plt.legend()
    plt.xlim([alphas[0], alphas[-1]])


alphas = np.logspace(-4, 4, 50)
scores, scores_std, k5optalpha, k5osralpha = estimate_alpha(alphas, 5)
print ("usual rule: alpha = %f \none stand error rule: alpha = %f"%(k5op
talpha,k5osralpha))
```

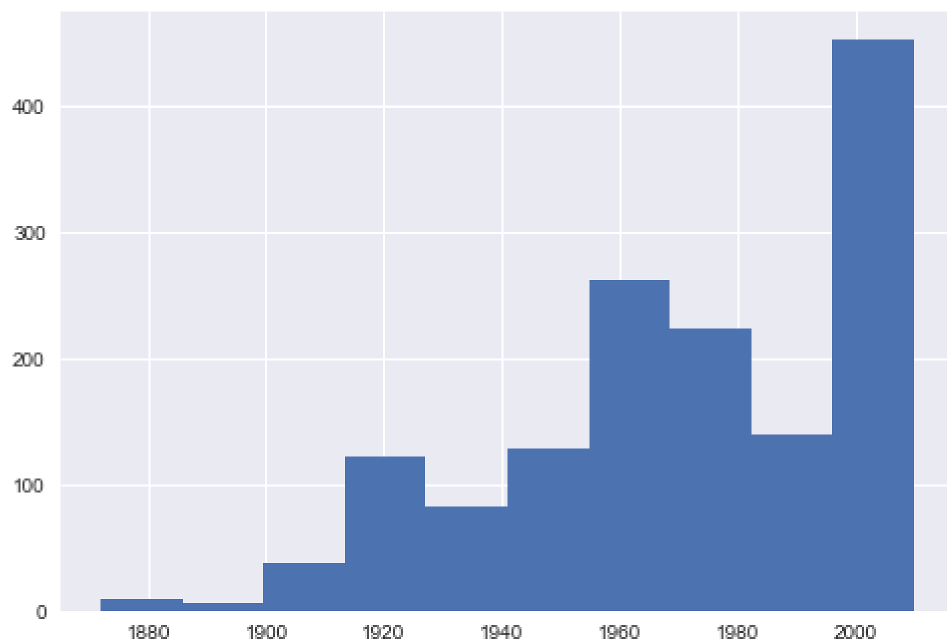
```
plot_cv_curve(alphas, scores, scores_std, k5optalpha, 5)
```

```
usual rule: alpha = 1.757511
one stand error rule: alpha = 2.559548
```



In [75]:
```
train_core = pd.get_dummies(train_core)
train_core = train_core.fillna(train_core.mean())
train_core['YearBuilt'].hist()
```

Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x14b5df28>



Use the alpha from CV to do the regression for test set:

In [76]:
```
lassoModel = linear_model.Lasso(alpha=0.001)
lassoModel.fit(trainX,trainY)
lasso_preds = lassoModel.predict(testX)
#print lasso_preds
```

In [77]:
```
from sklearn import ensemble
XGBoost = ensemble.GradientBoostingRegressor(n_estimators=3600, learning
_rate=0.05,loss='huber')
XGBoost.fit(trainX, trainY)
XGBoost_preds = XGBoost.predict(testX)
#print XGBoost_preds
```

In [78]:
```
preds = lasso_preds * 0.3 + XGBoost_preds * 0.7
print preds
"""
multi = np.repeat(1.0, len(testID))
for i,id in enumerate(testID):
    year = test.loc[i]['YrSold']
    mo = test.loc[i]['MoSold']

    if year == 2010 :
        if (mo == 7) or (mo ==6) or (mo<=4):
            multi[i] /=1.01
    elif year == 2009:
        if (mo ==1) or (mo ==6) or(mo<=4):
            multi[i] /= 1.04
        elif (mo==2) or(mo==3) or (mo==4):
            multi[i] /= 1.03
        elif (mo==5)or(mo==6)or(mo==7)or(mo==8):
             multi[i] /=1.02
        elif mo >=9:
             multi[i] /= 1.01
    elif year ==2008:
        if (mo ==1) or (mo ==12):
            multi[i] /= 1.04
        elif (mo==3) or(mo==11):
            multi[i] /= 1.03
        elif (mo==5)or(mo==10):
             multi[i] /=1.01
    elif year ==2007:
        if (mo ==1) or (mo ==2):
            multi[i] /= 1.08
        elif (mo==3):
            multi[i] /= 1.07
        elif (mo==4):
             multi[i] /=1.06
        elif (mo>=5)and (mo<=10):
            multi[i] /= 1.05
        elif (mo>10):
             multi[i] /=1.04
    elif year ==2006:
        if (mo ==1):
            multi[i] /= 1.11
        elif (mo==2) or (mo==3):
            multi[i] /= 1.10
        elif (mo==4) or (mo>-10):
             multi[i] /=1.09
        elif ((mo>=5)and (mo<=7)) or (m0==9):
            multi[i] /= 1.08
        elif (mo==8):
             multi[i] /=1.07

preds = preds*multi
"""
```

```
[ 176458.31472782  126928.21781292  170058.20946741  128107.38752103
  157050.52998574   60017.12937774  119522.42922282  135080.85442612
  159450.64320108  106119.71175407  325093.70387128  179852.35658332
  274252.21555096  181015.95746145  279942.88792462  187992.29787469
  204921.58417716  130021.9801795   134421.27300636  116818.71906282
  319101.96265353  183875.26929799  130034.33014036  140136.41693599
  133779.30979914  118213.87380213  212938.8526323   111942.53563428
  117679.55169275  163872.21622421  114959.31546996  174049.26083642
  259240.3486446   214993.16289395  140151.53608052  135137.59949318
   93816.44648497  117503.7964411   239774.52397833  169152.2557578
  101700.93126424  119099.76067231   94030.14966513  195933.27250095
  143917.55522561  139020.36915761  197616.29606144  425589.4614057
   79745.65202502   80096.6422044   149220.34798709  180179.52510091
  174813.03025503  116714.31183638  142928.48103548  123882.19462821
  150222.2685639   229810.72087984  120506.32044293  202282.43509486
  217194.48618542  179919.87501079  230013.09285531  235301.13421205
  185095.56696029  146044.36723494  223874.69005199  129027.69580338
  108887.46307776  194081.68214462  233211.28733563  245746.48568887
  173019.17146017  235106.60001798  644988.11716107  172063.24203105
  162580.00323323  172164.88244847  200489.39793475  239066.45631664
  285908.19000274  119592.62003887  115032.45796346  154741.32517724
   92794.5189346   250132.08988284  391720.5232667   713264.94344442
  119883.78544251  186509.96206394  105028.2464989    95201.44529427
  262554.69756101  194991.82514267  188858.90999683  167574.99135957
  174001.46980492  124784.46955239  165033.78243064  157742.6584008
  175995.08332671  219505.22363187  143996.36992444  178068.90593753
  148096.35062313  115902.67022243  197931.30024361  116880.72570671
  212962.75911178  153415.64144841  273317.51301025  106991.80035121
  200051.17793944  140039.10801295  289623.40866946  188795.7108632
  164075.77831483  112999.36590993  144787.03564522  134281.28070246
  124820.56480297  112006.8473788   229247.76465068   80083.18606345
   91676.05067261  115094.29067994  134052.71080094  143039.7755467
  137818.63133938  183872.52292033  144975.95624319  213891.05360457
  146756.66421626  368679.29636195  126919.22053684  190220.85248795
  132644.68545507  101728.97046131  141980.17267909  130218.6000453
  139116.78840403  175335.47349673  194855.52601592  142665.02321111
  265944.66253284  224960.4511954   248413.90639903  170047.37007928
  472556.57999086  230054.33059816  178091.15527002  186099.7638996
  170115.1072873   129197.62547459  119060.33498478  245205.88707081
  171530.1676054   130099.99561017  293332.69064992  165433.74472033
  127228.61878448  301299.61127663   99865.31583248  189991.02711129
  150930.59876184  181188.03318001  128901.08502884  161540.79213385
  180443.72065716  181076.3924318   183821.93392598  121820.06192135
  378242.54029787  380287.51888671  144119.93328279  259310.25126456
  185728.64241373  137105.58555429  176869.00605619  139003.22285677
  137154.83940253  161912.27079975  197933.04546862  236717.0058302
   68365.93842736  227107.08517892  179960.48782698  150729.36333679
  138970.0618176   169051.51221188  132370.90472929  143006.52946846
  190287.40787398  276691.226731    280183.38255567  180557.45343653
  119597.96572397  106939.00083893  162607.45299311  115099.19496259
  138470.99849503  155118.63293893  139943.48573893  160115.16054467
  153817.49329404  224997.17734737  177634.56451475  290279.24972478
  231807.12836886  130027.33902298  324781.05154762  202511.76505574
  137713.21741418  146969.24302539  179443.53479959  334937.55221227
  202967.15721195  301705.75077166  334434.92527994  119077.27504567
  207025.10812285  295960.55316623  209136.71966973  274803.83939143
  111143.724298    156047.63674796   72743.8386821   189921.1363164
```

```
  82163.45399299  147255.62797974   54760.10992855   79214.85376363
 130613.22660904  256219.89588801  176528.72285881  226897.72480051
 132492.07137146  100063.18099664  125874.57968907  125024.52339487
 167890.59476505  135005.27806582   52013.80477407  199976.21247146
 128509.02534103  122877.66798495  154910.76090648  228153.89367518
 176703.9822178   155714.35898836  108227.85799113  262542.2446872
 283354.51462859  215296.57549567  121878.26807301  200155.67303525
 170995.47954984  134905.72516099  411576.05466334  235114.75788906
 170071.10893846  109980.14990005  150095.8029741   177774.20774265
 314327.36138137  189359.99055494  259464.07117102  104955.99439101
 157005.87059876  144166.6710386   216242.93047443  193431.13409829
 126956.95716683  144082.31479419  231616.17572391  104842.06512938
 165402.07403873  274219.19430841  475723.66337888  250170.10556995
 238932.17360827   91018.04123651  116857.52815221   83255.89749378
 167559.5484052    59487.95754742  236797.71261308  157093.92152639
 112027.03577982  105066.33806413  125867.14987765  248484.6202592
 135460.98896331  377768.09643464  130859.16346597  234634.1897384
 124030.54537189  122884.63697855  162637.61856169  246081.99007864
 280194.6336556   160181.98415089  137517.64472458  137879.17867921
 137567.35876405  119853.26768064  193349.01720673  193939.59557339
 283478.97406005  105021.76757483  274275.87666606  133084.06079449
 111810.25024806  125839.1383564   215055.35694999  229945.01704023
 139994.08830936   89618.31190332  257847.99211104  206972.85909262
 175938.02466392  122354.77399167  340067.4922132   123705.1716386
 222633.53902115  179925.63367902  127189.57199851  137027.64185062
 277579.4252909   143943.24667615  141886.36796968  270139.52609308
 139815.47546718  119069.35863098  183140.05559392  192367.36394726
 143991.18268196   66241.02644605  186064.7883557   160137.62367982
 174060.82156007  120633.2315753   394672.48575384  149054.77007637
 197020.28216386  190939.24195763  149095.54934172  309809.1931834
 120734.27420556  179688.18724376  128954.85011084  157792.42093699
 240032.88935576  111926.73507755   91673.16475975  135751.20792278
 286691.61479338  145033.95850823   84423.43108816  184945.90073673
 175022.41139357  209998.49728605  265874.0343295   141911.21789913
 147938.72248231]
```

Out[78]:

```
"\nmulti = np.repeat(1.0, len(testID))\nfor i,id in enumerate(testI
D):\n    year = test.loc[i]['YrSold']\n    mo = test.loc[i]['MoSold']\n
    \n    if year == 2010 :\n        if (mo == 7) or (mo ==6) or (mo<=
4):\n            multi[i] /=1.01  \n    elif year == 2009:\n        if
 (mo ==1) or (mo ==6) or(mo<=4):\n            multi[i] /= 1.04\n
elif (mo==2) or(mo==3) or (mo==4):\n            multi[i] /= 1.03\n
   elif (mo==5)or(mo==6)or(mo==7)or(mo==8):\n            multi[i] /=1.
02\n        elif mo >=9:\n            multi[i] /= 1.01\n    elif year
 ==2008:\n        if (mo ==1) or (mo ==12):\n            multi[i] /= 1.
04\n        elif (mo==3) or(mo==11):\n            multi[i] /= 1.03\n
     elif (mo==5)or(mo==10):\n            multi[i] /=1.01\n    elif ye
ar ==2007:\n        if (mo ==1) or (mo ==2):\n            multi[i] /=
 1.08\n        elif (mo==3):\n            multi[i] /= 1.07\n        eli
f (mo==4):\n            multi[i] /=1.06\n        elif (mo>=5)and (mo<=
10):\n            multi[i] /= 1.05\n        elif (mo>10):\n
multi[i] /=1.04\n    elif year ==2006:\n        if (mo ==1):\n
   multi[i] /= 1.11\n        elif (mo==2) or (mo==3):\n            mult
i[i] /= 1.10\n        elif (mo==4) or (mo>-10):\n            multi[i]
 /=1.09\n        elif ((mo>=5)and (mo<=7)) or (m0==9):\n            mul
ti[i] /= 1.08\n        elif (mo==8):\n            multi[i] /=1.07\n
     \npreds = preds*multi\n"
```