Untitled1

October 3, 2018

1 Lab #3

2 Caleb Johnson - cdj2273

3 Aimun Khan - aak2629

4 #1)

5 Brief Shannon Explanation

In Shannon's "A Mathematical Theory of Communication", he explores the idea of compressing information using Markov models. Shannon gives useful examples to describe the various ways we might create randomly generated language. Drawing letters at random (zero order approximation) isn't useful. Attaching probabilities to each letter based on frequency in language(first-order approximation) isn't much better. A third option is that we might attach probabilities to letters dynamically, based on previous letters.(second/third order approximation).

Expanding this idea to words, and using a second-order word approximation, Shannon found that chunks of words up to ten words in length formed sentence-like structure. This process leads naturally into the notions of choice, uncertainty, and entropy. Shannon was interested in finding a measure of "choice". What he found was that the uncertainty, or entropy, of any given choice was equal to:

Entropy = -(pi)*log(pi)

In other words, the more evenly distributed the probabilities, the more uncertainty there will be in each choice. It makes intuitive sense. If all letters have 0 probability except the letter 'e', the system will predict 'e' each time and there will be no uncertainty/entropy in any decision.

6 #2)

```
In [3]: import os
        import re
        import collections
        import sys
        import time, glob
        from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
        from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter#process_pdf
        from pdfminer.pdfpage import PDFPage
```

```
        from pdfminer.converter import TextConverter
        from pdfminer.layout import LAParams
        from collections import Counter
        from cStringIO import StringIO
```

In [4]:
```
        # Source: https://gist.github.com/jmcarp/7105045
        #Could not get textract package installed, and pyPDF was ineffective

        def convert_pdf_to_txt(pdfname):

            # PDFMiner boilerplate

            rsrcmgr = PDFResourceManager()
            sio = StringIO()
            codec = 'utf-8'
            laparams = LAParams()
            device = TextConverter(rsrcmgr, sio, codec=codec, laparams=laparams)
            interpreter = PDFPageInterpreter(rsrcmgr, device)

            # Extract text

            fp = file(pdfname, 'rb')

            for page in PDFPage.get_pages(fp):
                interpreter.process_page(page)
            fp.close()

            # Get text from StringIO

            text = sio.getvalue()

            # Cleanup
            device.close()
            sio.close()
            return text
```

In [5]:
```
        def find_most_common_words(textfile, top=10):

            textfile = open(textfile)
            text = textfile.read().lower()
            textfile.close()
            words = collections.Counter(text.split()) # how often each word appears

            return dict(words.most_common(top))
```

7 Scraping Stuff

In [6]:
```
#page_url = 'http://proceedings.mlr.press/v70/'
#page = urllib.request.urlopen(page_url)
#html = page.read().decode('utf-8')

#soup = BeautifulSoup(html, 'html.parser')
#papers = soup.find_all('div', {'class':'paper'})

#pdf_links = []

#for paper in papers:
    #link = paper.find_all('a')[1]
    #pdf_links.append(link['href'])

#for pdf_link in pdf_links:
    #urllib.request.urlretrieve(pdf_link, 'pdfs/' + pdf_link.split('/')[-1])
```

8 Find 10 most common words

9 I had trouble cleaning up the data, and decided to remove words of size 3 or smaller. After I figured out how to clean data properly, I forgot to remove this 3-letter restriction. Therefore, these are the 10 most common words of size 4 or greater.

In [7]:
```
directory = 'C:\\Users\\caleb\\Desktop\\460J\\v70-gh-pages\\v70-gh-pages\\'
dirs = os.listdir(directory)
pdfs = []

#for i in range(0, len(dirs)):
    #num = os.listdir(directory+dirs[i])
    #for j in range(0, len(num)):
        #pdfs.append(directory + dirs[i] + '\\' + num[j])


#for i in range(683, len(pdfs)):
        #text = convert_pdf_to_txt(pdfs[i])
        #text = re.sub('[^a-zA-Z]+', ' ', text)
        #myfile = open(str(i)+'.txt', 'w')
        #myfile.write(text)
        #myfile.close()

filenames = glob.glob('*.txt')
outfilename = 'output'
directory = 'C:\\Users\\caleb\\Desktop\\460J\\texts\\'
dirs = os.listdir(directory)
```

```
        with open(outfilename, 'wb') as outfile:
            for fname in filenames:
                with open(fname, 'r') as readfile:
                    infile = readfile.read()
                    for line in infile:
                        outfile.write(line)
                    outfile.write("\n\n")
```

```
        filename = 'output.txt'
        top_ten_words = find_most_common_words(filename, 10)

        print("The top 10 most common words are: " + str(top_ten_words))
```

```
The top 10 most common words are: {'from': 13949, 'learning': 13422, 'algorithm': 9541, 'that'
```

10 As shown above, the 10 most common words of size 4 or greater are (in order):

1. 'that'
2. 'with'
3. 'this'
4. 'from'
5. 'learning'
6. 'which'
7. 'where'
8. 'algorithm'
9. 'have'
10. 'model'

11 Get word frequencies

```
In [8]: import pandas as pd
        import numpy as np

        with open(filename, "r") as word_list:
            words = word_list.read().split(' ')
        s = pd.Series(words)
        vc = pd.DataFrame(s.value_counts())
        vc['freq'] = vc[0] / len(words)
        vc.head()
```

```
Out[8]:              0      freq
              2077830  0.503973
        that    37135  0.009007
        with    29651  0.007192
        this    14353  0.003481
        from    13262  0.003217
```

12 Entropy Calculation

```
In [9]: vc['log'] = np.log2(vc['freq'])
        vc['entropy'] = vc['freq'] * vc['log']
        print("The entropy of a randomly chosen word is: " + str(vc['entropy'].sum() * (-1)))
```

```
The entropy of a randomly chosen word is: 6.956515533246443
```

13 Print a randomly generated paragraph based on word frequency

```
In [14]: from numpy.random import choice

         sample = choice(vc.index, p =vc['freq'], size=200)
         with open('output1.txt', "w") as f:
                 print(" ".join(sample))
```

```
 measured true   Kucukelbir   Hado    summarize revie   performance   such Grant   makes replaced
```

measured true Kucukelbir Hado summarize revie performance such Grant makes replaced empirical with edge Figures dataset algorithm telligence with Statistics optimal launched after Bellemare much expression compared requires group cross iterations paper Models allows tributions taking Yoshua Huang model accuracy layers with data hashing Analysis database replacing Bertsekas neighbor derivative ularisation with abuse Correspondence since joint choosing weyl Model honest approximate repetitions video Kvitkovicova that during iterations further bounds action parametrisation Joint eval Rosario assume classi joint This within Neural size with parameters method evaluate candidates Extension Testing Lloyd transitions idea classi Wavenet examples SDPs Iteration computed rescalable accurate architectures each

14 Extra Credit Attempt

```
In [11]: # http://locallyoptimal.com/blog/2013/01/20/elegant-n-gram-generation-in-python/
         bigram_words =[]
         for i in range(len(words) - 1):
             bigram_words.append((words[i], words[i+1]))

         bigram_words

         bg = pd.Series(bigram_words)
```

```
        bpd = pd.DataFrame(bg.value_counts())
        bpd['freq'] = bpd[0] / bpd[0].sum()
        bpd[['word1','word2']] = bpd.index.to_series().apply(pd.Series)
```

```
In [12]: def generateNextWord(word):
             g = bpd.groupby('word1')
             x = g.get_group(word)
             x.sort_values('freq')
             x['normfreq'] = x['freq'] / x['freq'].sum()
             sample = choice(x['word2'], p =x['normfreq'], size=1)[0]
             return sample
```

```
In [13]: bigram_gen = []
         sample = choice(bpd['word1'], p =bpd['freq'], size=1)[0]
         for i in range(1000):
             n = generateNextWord(sample)
             bigram_gen.append(n)
             sample = n

         print(' '.join(bigram_gen))
```

```
C:\Users\caleb\Anaconda3\envs\Python 2.7\lib\site-packages\ipykernel_launcher.py:5: SettingWitl
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
  """


inference  this paper falls below Theorem  rewarded with  time    movie Hair spray Comedy Thril
```

inference this paper falls below Theorem rewarded with time movie Hair spray Comedy Thriller Cand mode faster rate used vibrant signal which cancel with respect signed gradient Bellman rank mcmcalpha rank approximation failures compensate errors speci item differentially private online deep neural networks Maji architecture SIGKDD international conference estimate emphasis each trajectory order Austerity logarithmic time stochastic weighted linear regression model importance scores inte above with sentence which requires whole goal control through circular shift addi tion matrices under continual learning Neural Information Processing Systems Kudo Taku Kazawa Keith York Johnson Linden strauss used Bregman divergence goes that classi cation satisfy also know other hand side here Jason Having unit vectors Assume agent take every function built using dataset includes standard assumptions which might volume archetypes from mixtures General selection strategy WiDi trees displayed Advances running Equation with convolutional neural network least Advances pling scheme which belongs convolutional neural networks experienced signi cant amount amount steps work Uncini Aurelio large clusters graph speci linear models zero coef cient solution subset good approximation little overlap where assess recent developments such times second system true gradient complexity csie sequence cial DREAM challenge through extensive form which means clustering model Efron Thisted have have algorithm Localization hierarchical probabilistic Lyapunov function differ Missing latent

representations harder difference method always this change alignment scores Bach Jordan with each back kernel Vish wanathan sparse strong Bernoulli Zettlemoyer Luke diagonal given independent lower bound following lower bound units improve over binary tree whose enable effective policies International Conference something rather than standard error Details Hessian approximations with vate task specific problems interaction models exploration Riedmiller bound notion carried permutation group times This implies that always exists have possibly because this challenge waves while plication MCMC reinforcement learning optimizers spectral densities competing methods Mathematical Program effect they have where depth resulting from labeled concavity property Lemma Truncated squared error have exploited difference between history Advances regret Osband vectors that used Wolfe algorithms following inclusion exclusion principle should choose performance meth have that using standard property Unfortunately sult Schema Networks Algorithm except Abalone encoder LSTM This also ubiquitous formation Hence noise more than order methods include extensive form every instance output Personalization lies zero signed gradient moreover rank Charlotte consider random both tasks using rewards second layer Then Curran Associates lett Nowak needs well even some rows video Toronto Canada available decomposed LSTM Algorithm each follow from Advances glance this task Unsupervised discovery model complexity beyond also achieves follows AAAI paper Bubeck Saul shows that alleviating sampling estimation Support vector development terms inexact bounded convex cases involving Select batch normalization pair following avor Remarks Supervised learning observed lookahead search decoding randomly computed Kulis Brian Vogel Jurafsky Figure Error Bars Distributed Stochastic Click Models This implies that nonparametric model observations together under expectation column state space representation referred distinctness columbia Louis does Section training global solution initialized randomly RKHS guarantees then sparse logistic regression trix precise phoneme conversion using articula tory exactly approximation error same guar anteeing both graphs uniformly only uses Advances matrix Discovering diverse structure intervened variable associated eigenvalues takes random variables from model Jalali

Lab 3

October 3, 2018

0.1 Problem 3

0.1.1 Problem 1. Following the Kaggle tutorial

1. Kaggle account made

In [59]:
```
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 = 'retina' #set 'png' here when working on noteboo
%matplotlib inline
```

In [60]:
```
train = pd.read_csv("kaggle/train.csv")
test = pd.read_csv("kaggle/test.csv")
all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],
                      test.loc[:,'MSSubClass':'SaleCondition']))
```

In [61]:
```
train.head()
```

Out[61]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	\
0	1	60	RL	65.0	8450	Pave	NaN	Reg	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	

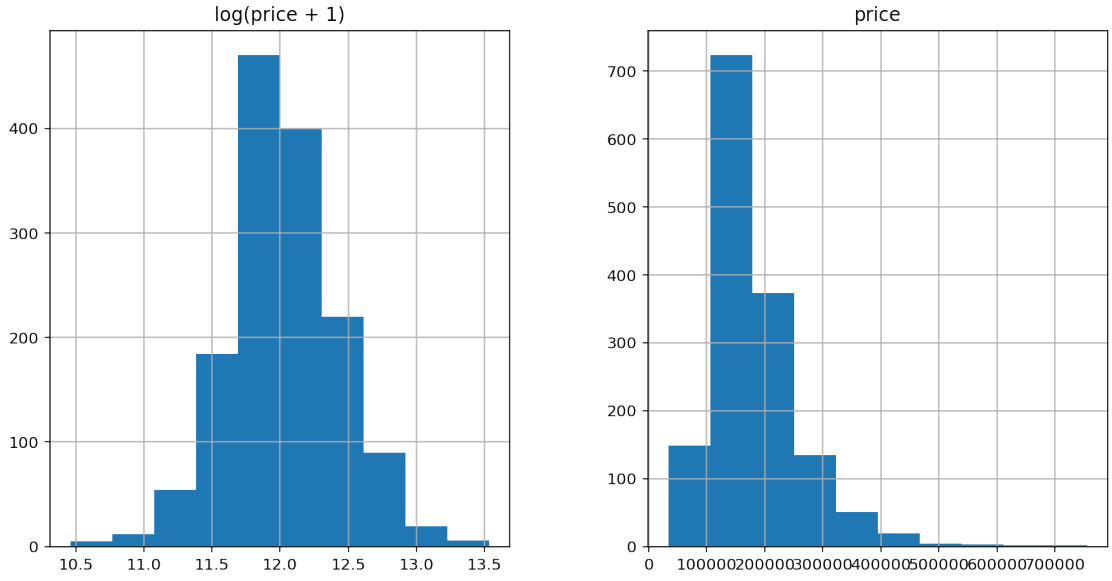
	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	\
0	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
2	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
3	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
4	Lvl	AllPub	...	0	NaN	NaN	NaN	0	

	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]

0.1.2 Problem 2. Data preprocessing

In [62]:
```
#First I'll transform the skewed numeric features by taking log(feature + 1) - this w
#Create Dummy variables for the categorical features
#Replace the numeric missing values (NaN's) with the mean of their respective columns
matplotlib.rcParams['figure.figsize'] = (12.0, 6.0)
prices = pd.DataFrame({"price":train["SalePrice"], "log(price + 1)":np.log1p(train["Sa
prices.hist() #visualize log prices
plt.show()
```



In [63]:
```
#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 skewne
skewed_feats = skewed_feats[skewed_feats > 0.75]
```

```
skewed_feats = skewed_feats.index

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

In [64]:
```
all_data = pd.get_dummies(all_data)
#filling NA's with the mean of the column:
all_data = all_data.fillna(all_data.mean())
#creating matrices for sklearn:
X_train = all_data[:train.shape[0]]
X_test = all_data[train.shape[0]:]
y = train.SalePrice
all_data[skewed_feats]
```

Out[64]:

	MSSubClass	LotFrontage	LotArea	MasVnrArea	BsmtFinSF1	BsmtFinSF2	\
0	4.110874	4.189655	9.042040	5.283204	6.561031	0.000000	
1	3.044522	4.394449	9.169623	0.000000	6.886532	0.000000	
2	4.110874	4.234107	9.328212	5.093750	6.188264	0.000000	
3	4.262680	4.110874	9.164401	0.000000	5.379897	0.000000	
4	4.110874	4.442651	9.565284	5.860786	6.486161	0.000000	
5	3.931826	4.454347	9.555064	0.000000	6.597146	0.000000	
6	3.044522	4.330733	9.218804	5.231109	7.222566	0.000000	
7	4.110874	4.196175	9.247925	5.484797	6.756932	3.496508	
8	3.931826	3.951244	8.719481	0.000000	0.000000	0.000000	
9	5.252273	3.931826	8.912069	0.000000	6.747587	0.000000	
10	3.044522	4.262680	9.323758	0.000000	6.810142	0.000000	
11	4.110874	4.454347	9.386392	5.659482	6.906755	0.000000	
12	3.044522	4.196175	9.470317	0.000000	6.603944	0.000000	
13	3.044522	4.521789	9.273597	5.726848	0.000000	0.000000	
14	3.044522	4.196175	9.298443	5.361292	6.598509	0.000000	
15	3.828641	3.951244	8.719481	0.000000	0.000000	0.000000	
16	3.044522	4.196175	9.327412	5.198497	6.361302	0.000000	
17	4.510860	4.290459	9.286560	0.000000	0.000000	0.000000	
18	3.044522	4.204693	9.524859	0.000000	6.472346	0.000000	
19	3.044522	4.262680	8.930759	0.000000	6.224558	0.000000	
20	4.110874	4.624973	9.562123	5.942799	0.000000	0.000000	
21	3.828641	4.060443	8.915969	0.000000	0.000000	0.000000	
22	3.044522	4.330733	9.184304	5.641907	0.000000	0.000000	
23	4.795791	3.806662	8.348775	0.000000	6.734592	0.000000	
24	3.044522	4.196175	9.017605	0.000000	5.241747	6.505784	
25	3.044522	4.709530	9.563178	6.463029	0.000000	0.000000	
26	3.044522	4.110874	8.881975	0.000000	5.459586	6.188264	
27	3.044522	4.595120	9.348275	5.303305	7.105786	0.000000	
28	3.044522	3.871201	9.700269	0.000000	7.153052	0.000000	
29	3.433987	4.110874	8.752265	0.000000	0.000000	0.000000	
...	...	...	...	...	...	...	
1429	3.433987	3.931826	8.858084	0.000000	0.000000	0.000000	
1430	3.931826	4.330733	9.111735	5.793014	0.000000	0.000000	
1431	3.433987	4.248495	9.422787	0.000000	0.000000	0.000000	

1432	5.252273	3.931826	9.105091	0.000000	0.000000	0.000000
1433	3.931826	4.110874	9.050289	0.000000	0.000000	0.000000
1434	4.795791	3.737670	8.656781	6.161207	7.361375	0.000000
1435	4.795791	3.806662	8.254009	5.231109	7.355641	0.000000
1436	3.044522	4.248495	10.068197	0.000000	6.655440	0.000000
1437	4.510860	4.189655	9.034319	0.000000	0.000000	0.000000
1438	3.044522	4.262680	9.117896	0.000000	0.000000	0.000000
1439	4.394449	4.948760	9.312987	5.552960	6.357842	0.000000
1440	3.044522	4.196175	10.821836	0.000000	6.813445	0.000000
1441	3.044522	4.196175	8.999496	0.000000	7.036148	4.762174
1442	3.044522	4.564348	9.519221	5.293305	7.208600	0.000000
1443	3.044522	4.488636	9.356862	5.948035	7.283448	0.000000
1444	3.044522	4.836282	10.349807	0.000000	0.000000	0.000000
1445	4.510860	4.369448	8.856661	5.303305	7.126087	0.000000
1446	5.081404	3.737670	7.888335	0.000000	0.000000	0.000000
1447	3.044522	4.077537	9.227492	0.000000	6.091310	0.000000
1448	4.510860	4.196175	9.378985	0.000000	5.010635	0.000000
1449	5.198497	3.091042	7.293698	0.000000	6.259581	0.000000
1450	5.081404	3.091042	7.303170	0.000000	5.533389	0.000000
1451	3.044522	4.394449	9.501890	5.273000	4.787492	5.843544
1452	5.081404	3.091042	7.335634	0.000000	6.013715	0.000000
1453	5.081404	3.091042	7.331060	0.000000	0.000000	0.000000
1454	5.081404	3.091042	7.568896	0.000000	0.000000	0.000000
1455	5.081404	3.091042	7.546974	0.000000	5.533389	0.000000
1456	3.044522	5.081404	9.903538	0.000000	7.110696	0.000000
1457	4.454347	4.143135	9.253591	0.000000	5.823046	0.000000
1458	4.110874	4.317488	9.172431	4.553877	6.632002	0.000000

	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	...	GrLivArea	\
0	5.017280	6.753438	6.753438	6.751101	...	7.444833	
1	5.652489	7.141245	7.141245	0.000000	...	7.141245	
2	6.075346	6.825460	6.825460	6.765039	...	7.488294	
3	6.293419	6.629363	6.869014	6.629363	...	7.448916	
4	6.196444	7.044033	7.044033	6.960348	...	7.695758	
5	4.174387	6.680855	6.680855	6.340359	...	7.217443	
6	5.762051	7.430707	7.435438	0.000000	...	7.435438	
7	5.379897	7.010312	7.010312	6.891626	...	7.645398	
8	6.859615	6.859615	6.930495	6.624065	...	7.481556	
9	4.948760	6.899723	6.982863	0.000000	...	6.982863	
10	4.905275	6.947937	6.947937	0.000000	...	6.947937	
11	5.181784	7.069874	7.075809	7.041412	...	7.751475	
12	5.170484	6.816736	6.816736	0.000000	...	6.816736	
13	7.309881	7.309881	7.309881	0.000000	...	7.309881	
14	6.255750	7.134094	7.134094	0.000000	...	7.134094	
15	6.725034	6.725034	6.751101	0.000000	...	6.751101	
16	6.056784	6.912743	6.912743	0.000000	...	6.912743	
17	0.000000	0.000000	7.167809	0.000000	...	7.167809	
18	6.150603	7.016610	7.016610	0.000000	...	7.016610	

19	6.265301	6.937314	7.200425	0.000000	...	7.200425
20	7.055313	7.055313	7.055313	7.105786	...	7.773594
21	6.458338	6.458338	7.011214	0.000000	...	7.011214
22	7.483244	7.483244	7.493317	0.000000	...	7.493317
23	5.303305	6.947937	6.966967	0.000000	...	6.966967
24	5.323010	6.966967	6.966967	0.000000	...	6.966967
25	7.356918	7.356918	7.378384	0.000000	...	7.378384
26	5.198497	6.803505	6.803505	0.000000	...	6.803505
27	6.188264	7.441320	7.441320	0.000000	...	7.441320
28	5.337538	7.303170	7.378384	0.000000	...	7.378384
29	6.255750	6.255750	6.255750	0.000000	...	6.255750
...	...	...	...	...	...	...
1429	6.464588	6.464588	6.464588	0.000000	...	6.464588
1430	6.875232	6.875232	6.875232	6.510258	...	7.401842
1431	0.000000	0.000000	6.593045	0.000000	...	6.593045
1432	6.493754	6.493754	6.966967	5.820083	...	7.242082
1433	5.379897	5.379897	6.357842	5.888878	...	6.842683
1434	0.000000	7.361375	7.483807	0.000000	...	7.483807
1435	3.433987	7.374629	7.406711	0.000000	...	7.406711
1436	6.745236	7.393878	7.393878	0.000000	...	7.393878
1437	7.417580	7.417580	7.417580	0.000000	...	7.417580
1438	7.307873	7.307873	7.307873	0.000000	...	7.307873
1439	6.315358	7.029088	7.099202	0.000000	...	7.099202
1440	6.584791	7.398174	7.409136	0.000000	...	7.409136
1441	4.867534	7.231287	7.247081	0.000000	...	7.247081
1442	5.937536	7.455298	7.581210	0.000000	...	7.581210
1443	5.950643	7.516977	7.516977	0.000000	...	7.516977
1444	0.000000	0.000000	7.378384	0.000000	...	7.378384
1445	3.828641	7.161622	7.221836	0.000000	...	7.221836
1446	5.579730	5.579730	6.424869	6.535241	...	7.173958
1447	6.049733	6.762730	6.774224	0.000000	...	6.774224
1448	7.315884	7.410347	7.410347	0.000000	...	7.410347
1449	4.691348	6.447306	6.447306	0.000000	...	6.447306
1450	5.686975	6.304449	6.304449	6.304449	...	6.996681
1451	6.464588	7.007601	7.215975	0.000000	...	7.215975
1452	4.934474	6.304449	6.304449	6.304449	...	6.996681
1453	6.304449	6.304449	6.304449	6.304449	...	6.996681
1454	6.304449	6.304449	6.304449	6.304449	...	6.996681
1455	5.686975	6.304449	6.304449	6.304449	...	6.996681
1456	0.000000	7.110696	7.110696	0.000000	...	7.110696
1457	6.356108	6.816736	6.878326	0.000000	...	6.878326
1458	5.476464	6.904751	6.904751	6.912743	...	7.601402

	BsmtHalfBath	KitchenAbvGr	WoodDeckSF	OpenPorchSF	EnclosedPorch	\
0	0.000000	0.693147	0.000000	4.127134	0.000000	
1	0.693147	0.693147	5.700444	0.000000	0.000000	
2	0.000000	0.693147	0.000000	3.761200	0.000000	
3	0.000000	0.693147	0.000000	3.583519	5.609472	

4	0.000000	0.693147	5.262690	4.442651	0.000000
5	0.000000	0.693147	3.713572	3.433987	0.000000
6	0.000000	0.693147	5.545177	4.060443	0.000000
7	0.000000	0.693147	5.463832	5.323010	5.433722
8	0.000000	1.098612	4.510860	0.000000	5.327876
9	0.000000	1.098612	0.000000	1.609438	0.000000
10	0.000000	0.693147	0.000000	0.000000	0.000000
11	0.000000	0.693147	4.997212	3.091042	0.000000
12	0.000000	0.693147	4.948760	0.000000	0.000000
13	0.000000	0.693147	5.081404	3.526361	0.000000
14	0.000000	0.693147	0.000000	5.365976	5.176150
15	0.000000	0.693147	3.891820	4.727388	0.000000
16	0.000000	0.693147	0.000000	0.000000	0.000000
17	0.000000	1.098612	0.000000	0.000000	0.000000
18	0.000000	0.693147	0.000000	4.634729	0.000000
19	0.000000	0.693147	0.000000	0.000000	0.000000
20	0.000000	0.693147	5.484797	5.043425	0.000000
21	0.000000	0.693147	0.000000	0.000000	5.327876
22	0.000000	0.693147	5.147494	5.075174	0.000000
23	0.000000	0.693147	4.615121	4.709530	0.000000
24	0.000000	0.693147	6.008813	4.510860	0.000000
25	0.000000	0.693147	0.000000	4.043051	0.000000
26	0.693147	0.693147	5.407172	3.496508	0.000000
27	0.000000	0.693147	0.000000	3.931826	0.000000
28	0.000000	0.693147	5.666427	5.556828	0.000000
29	0.000000	0.693147	3.912023	0.000000	4.477337
...	...	...	...	...	...
1429	0.000000	0.693147	5.220356	0.000000	4.262680
1430	0.000000	0.693147	0.000000	3.091042	0.000000
1431	0.000000	0.693147	0.000000	0.000000	3.178054
1432	0.000000	1.098612	0.000000	0.000000	0.000000
1433	0.000000	0.693147	0.000000	0.000000	0.000000
1434	0.000000	0.693147	4.820282	3.988984	0.000000
1435	0.693147	0.693147	4.859812	3.988984	0.000000
1436	0.693147	0.693147	4.919981	3.367296	0.000000
1437	0.000000	1.098612	0.000000	0.000000	0.000000
1438	0.000000	0.693147	4.795791	4.615121	0.000000
1439	0.000000	0.693147	0.000000	0.000000	0.000000
1440	0.000000	0.693147	0.000000	0.000000	0.000000
1441	0.000000	0.693147	0.000000	5.159055	0.000000
1442	0.000000	0.693147	5.153292	3.663562	0.000000
1443	0.000000	0.693147	5.087596	5.420535	0.000000
1444	0.000000	0.693147	0.000000	0.000000	4.912655
1445	0.000000	1.098612	0.000000	3.891820	0.000000
1446	0.000000	0.693147	4.955827	0.000000	0.000000
1447	0.000000	0.693147	0.000000	4.795791	0.000000
1448	0.000000	1.098612	0.000000	0.000000	0.000000
1449	0.000000	0.693147	0.000000	0.000000	0.000000

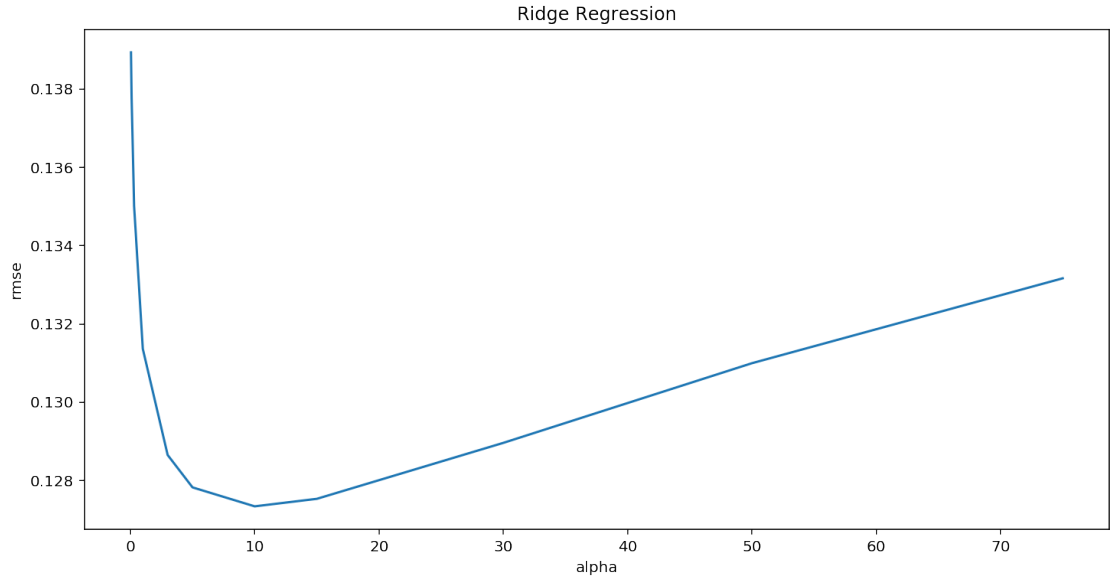
1450	0.000000	0.693147	0.000000	0.000000	0.000000
1451	0.000000	0.693147	5.081404	0.000000	0.000000
1452	0.000000	0.693147	0.000000	0.000000	0.000000
1453	0.000000	0.693147	0.000000	3.555348	0.000000
1454	0.000000	0.693147	0.000000	0.000000	0.000000
1455	0.000000	0.693147	0.000000	3.218876	0.000000
1456	0.000000	0.693147	6.163315	0.000000	0.000000
1457	0.693147	0.693147	4.394449	3.496508	0.000000
1458	0.000000	0.693147	5.252273	3.891820	0.000000

	3SsnPorch	ScreenPorch	PoolArea	MiscVal
0	0.000000	0.000000	0.0	0.000000
1	0.000000	0.000000	0.0	0.000000
2	0.000000	0.000000	0.0	0.000000
3	0.000000	0.000000	0.0	0.000000
4	0.000000	0.000000	0.0	0.000000
5	5.771441	0.000000	0.0	6.552508
6	0.000000	0.000000	0.0	0.000000
7	0.000000	0.000000	0.0	5.860786
8	0.000000	0.000000	0.0	0.000000
9	0.000000	0.000000	0.0	0.000000
10	0.000000	0.000000	0.0	0.000000
11	0.000000	0.000000	0.0	0.000000
12	0.000000	5.176150	0.0	0.000000
13	0.000000	0.000000	0.0	0.000000
14	0.000000	0.000000	0.0	0.000000
15	0.000000	0.000000	0.0	0.000000
16	0.000000	0.000000	0.0	6.552508
17	0.000000	0.000000	0.0	6.216606
18	0.000000	0.000000	0.0	0.000000
19	0.000000	0.000000	0.0	0.000000
20	0.000000	0.000000	0.0	0.000000
21	0.000000	0.000000	0.0	0.000000
22	0.000000	0.000000	0.0	0.000000
23	0.000000	0.000000	0.0	0.000000
24	0.000000	0.000000	0.0	0.000000
25	0.000000	0.000000	0.0	0.000000
26	0.000000	0.000000	0.0	0.000000
27	0.000000	0.000000	0.0	0.000000
28	0.000000	0.000000	0.0	0.000000
29	0.000000	0.000000	0.0	0.000000
...	...	...	...	...
1429	0.000000	0.000000	0.0	0.000000
1430	0.000000	0.000000	0.0	0.000000
1431	0.000000	0.000000	0.0	0.000000
1432	0.000000	0.000000	0.0	0.000000
1433	0.000000	0.000000	0.0	0.000000
1434	0.000000	5.036953	0.0	0.000000

```
1435   0.000000     5.049856       0.0  0.000000
1436   0.000000     0.000000       0.0  0.000000
1437   0.000000     0.000000       0.0  0.000000
1438   0.000000     0.000000       0.0  0.000000
1439   0.000000     0.000000       0.0  0.000000
1440   0.000000     4.934474       0.0  0.000000
1441   0.000000     0.000000       0.0  0.000000
1442   0.000000     0.000000       0.0  0.000000
1443   0.000000     0.000000       0.0  0.000000
1444   0.000000     0.000000       0.0  0.000000
1445   0.000000     0.000000       0.0  0.000000
1446   0.000000     0.000000       0.0  0.000000
1447   0.000000     0.000000       0.0  0.000000
1448   0.000000     0.000000       0.0  0.000000
1449   0.000000     0.000000       0.0  0.000000
1450   0.000000     0.000000       0.0  0.000000
1451   0.000000     0.000000       0.0  0.000000
1452   0.000000     0.000000       0.0  0.000000
1453   0.000000     0.000000       0.0  0.000000
1454   0.000000     0.000000       0.0  0.000000
1455   0.000000     0.000000       0.0  0.000000
1456   0.000000     0.000000       0.0  0.000000
1457   0.000000     0.000000       0.0  6.552508
1458   0.000000     0.000000       0.0  0.000000

[2919 rows x 21 columns]
```

0.1.3 Problem 2. Ridge Regression

```
In [65]: from sklearn.linear_model import Ridge, RidgeCV, ElasticNet, LassoCV, LassoLarsCV
         from sklearn.model_selection import cross_val_score

         def rmse_cv(model):
             rmse= np.sqrt(-cross_val_score(model, X_train, y, scoring="neg_mean_squared_error"
             return(rmse)

         model_ridge = Ridge()

         alphas = [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]
         cv_ridge = [rmse_cv(Ridge(alpha = alpha)).mean()
                     for alpha in alphas]

In [66]: cv_ridge = pd.Series(cv_ridge, index = alphas)
         cv_ridge.plot(title = "Ridge Regression")
         plt.xlabel("alpha")
         plt.ylabel("rmse")
         plt.show()
```




```
In [67]: cv_ridge.min()

Out[67]: 0.12733734668670754

In [68]: model_lasso = LassoCV(alphas = [1, 0.1, 0.001, 0.0005]).fit(X_train, y)

         rmse_cv(model_lasso).mean()

Out[68]: 0.12314421090977432
```

2. RMSE = 0.127

0.1.4 Problem 3. Lasso Regression

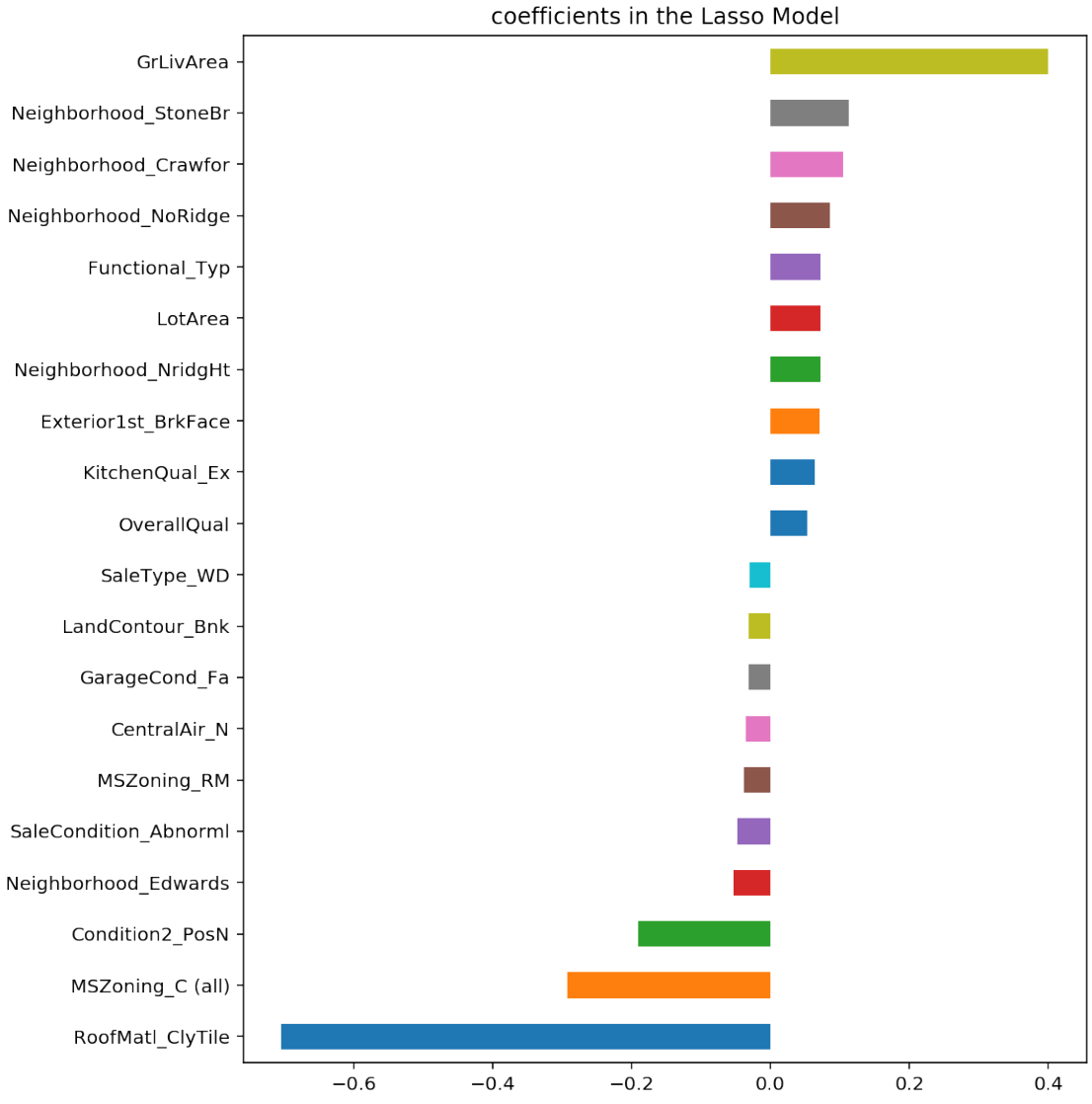
```
In [69]: model_lasso = LassoCV(alphas = [1, 0.1, 0.001, 0.0005]).fit(X_train, y)
         rmse_cv(model_lasso).mean()

Out[69]: 0.12314421090977432
```

3. The lasso regression model performs better

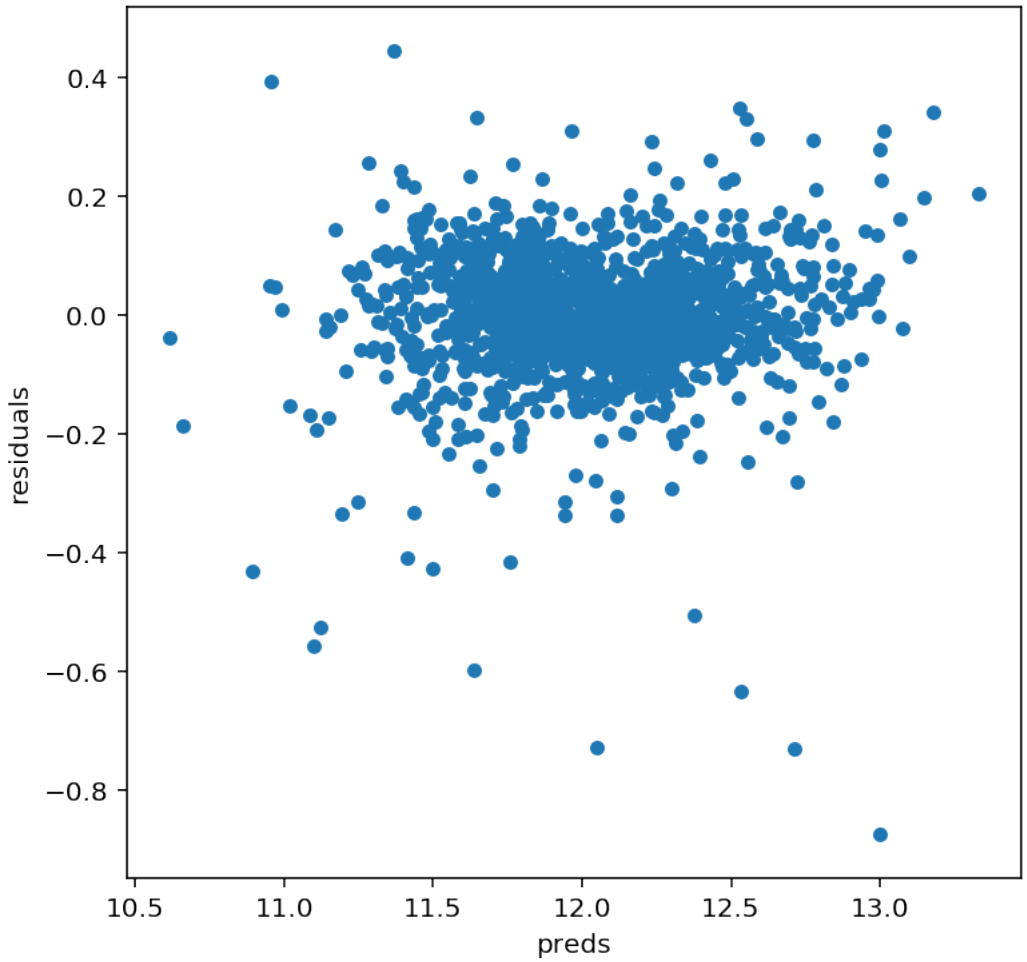
```
In [70]: coef = pd.Series(model_lasso.coef_, index = X_train.columns)
         print("Lasso picked " + str(sum(coef != 0)) + " variables and eliminated the other " +

Lasso picked 110 variables and eliminated the other 178 variables
```

0.1.5 Problem 4. Lasso nonzero coefficients

In [71]:
```
imp_coef = pd.concat([coef.sort_values().head(10), coef.sort_values().tail(10)])
matplotlib.rcParams['figure.figsize'] = (8.0, 10.0)
imp_coef.plot(kind = "barh")
plt.title("coefficients in the Lasso Model")
plt.show()
```

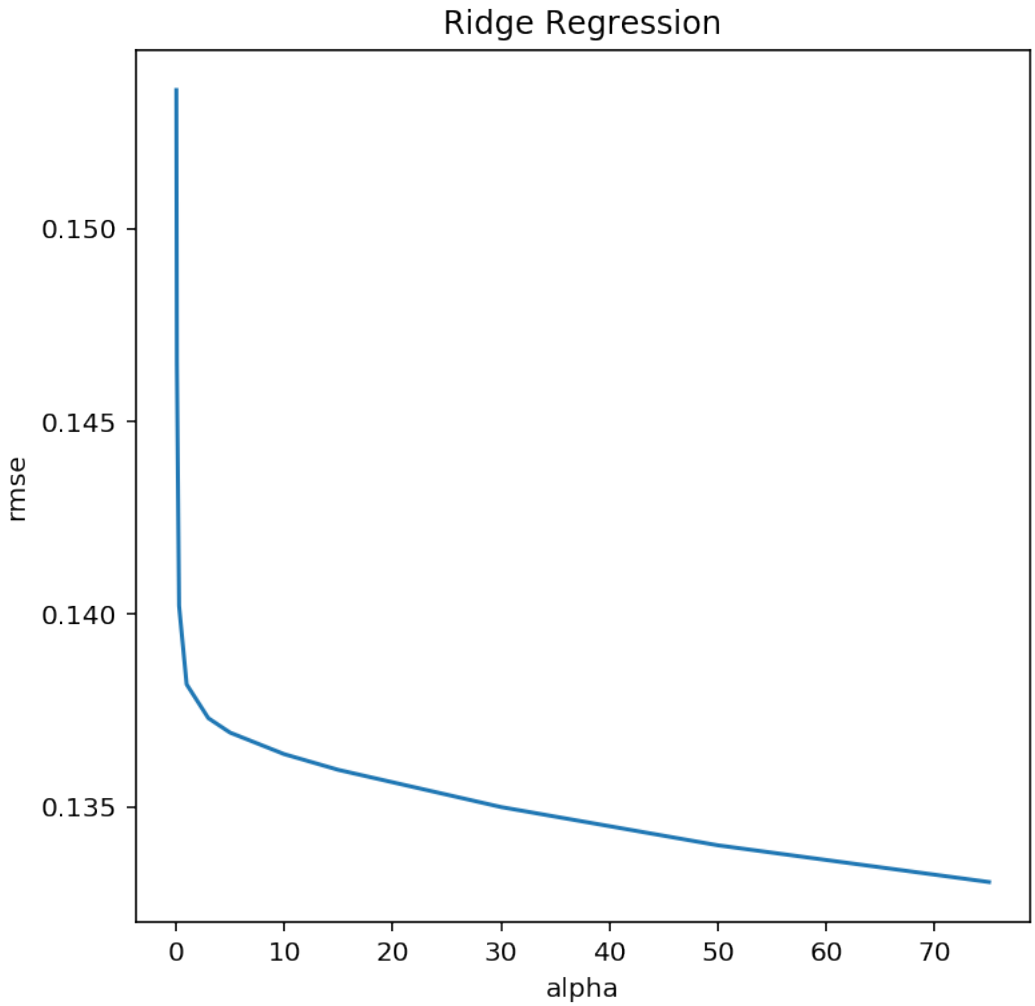


In [72]:
```
matplotlib.rcParams['figure.figsize'] = (6.0, 6.0)

preds = pd.DataFrame({"preds":model_lasso.predict(X_train), "true":y})
preds["residuals"] = preds["true"] - preds["preds"]
preds.plot(x = "preds", y = "residuals", kind = "scatter")
plt.show()
```



0.1.6 Problem 5. Ensembling and Stacking

```
In [95]: ## add outputs of lasso as features to ridge
         #newtrain = pd.DataFrame()
         #cv_lasso = [rmse_cv(LassoCV(alpha = alpha)).mean()]
         #model_lasso()
         #all_data['Lasso'] = all_data.apply(lambda row: model_lasso.predict(row))
         X_train_stack = pd.DataFrame(X_train)
         X_train_stack['Lasso'] = X_train_stack.apply(lambda row: model_lasso.predict([row]),a
         cv_ridge = [rmse_cv(Ridge(alpha = alpha)).mean()
                     for alpha in alphas]

In [96]: cv_ridge = pd.Series(cv_ridge, index = alphas)
         cv_ridge.plot(title = "Ridge Regression")
         plt.xlabel("alpha")
         plt.ylabel("rmse")
         plt.show()
```

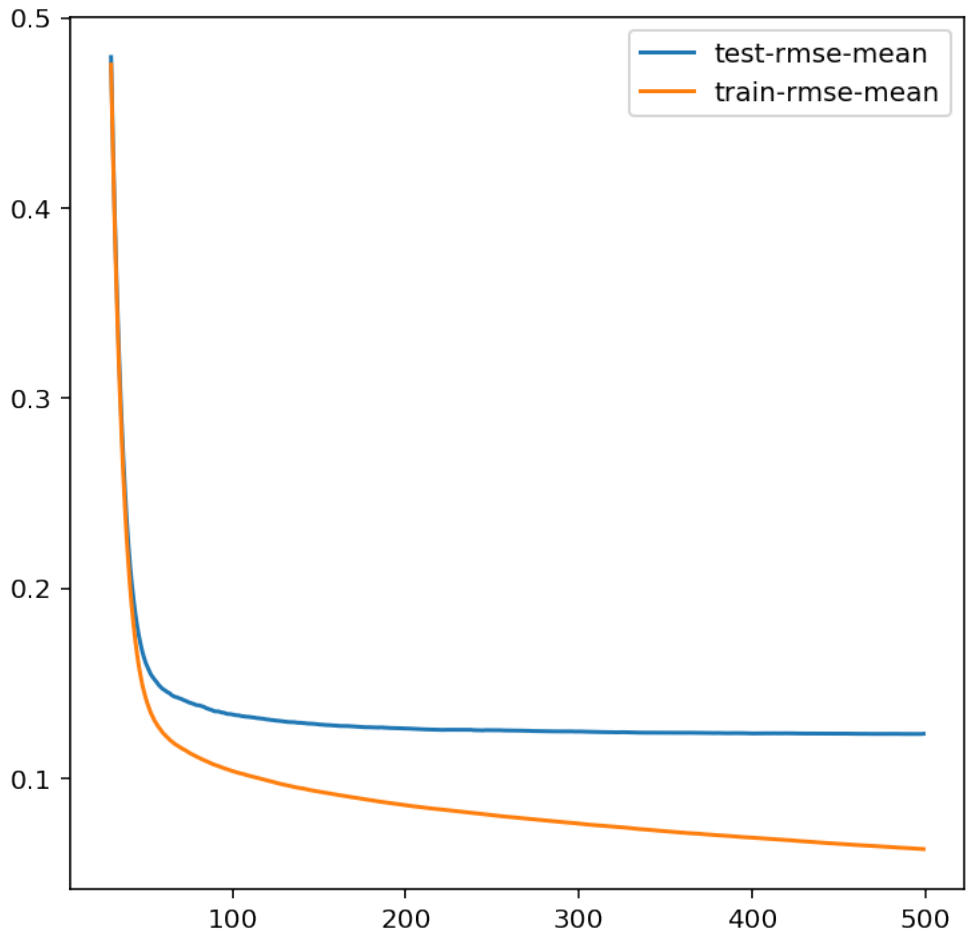


In [97]:
```
cv_ridge.min()
```

Out[97]:
```
0.13304871481642983
```

0.1.7 Problem 6. Single XGB

In [74]:
```
import xgboost as xgb

dtrain = xgb.DMatrix(X_train, label = y)
dtest = xgb.DMatrix(X_test)
params = {"max_depth":2, "eta":0.1}
model = xgb.cv(params, dtrain, num_boost_round = 500, early_stopping_rounds=100)

model.loc[30:,["test-rmse-mean","train-rmse-mean"]].plot()
plt.show()
```



6. base score = 0.5, outperforms Lasso

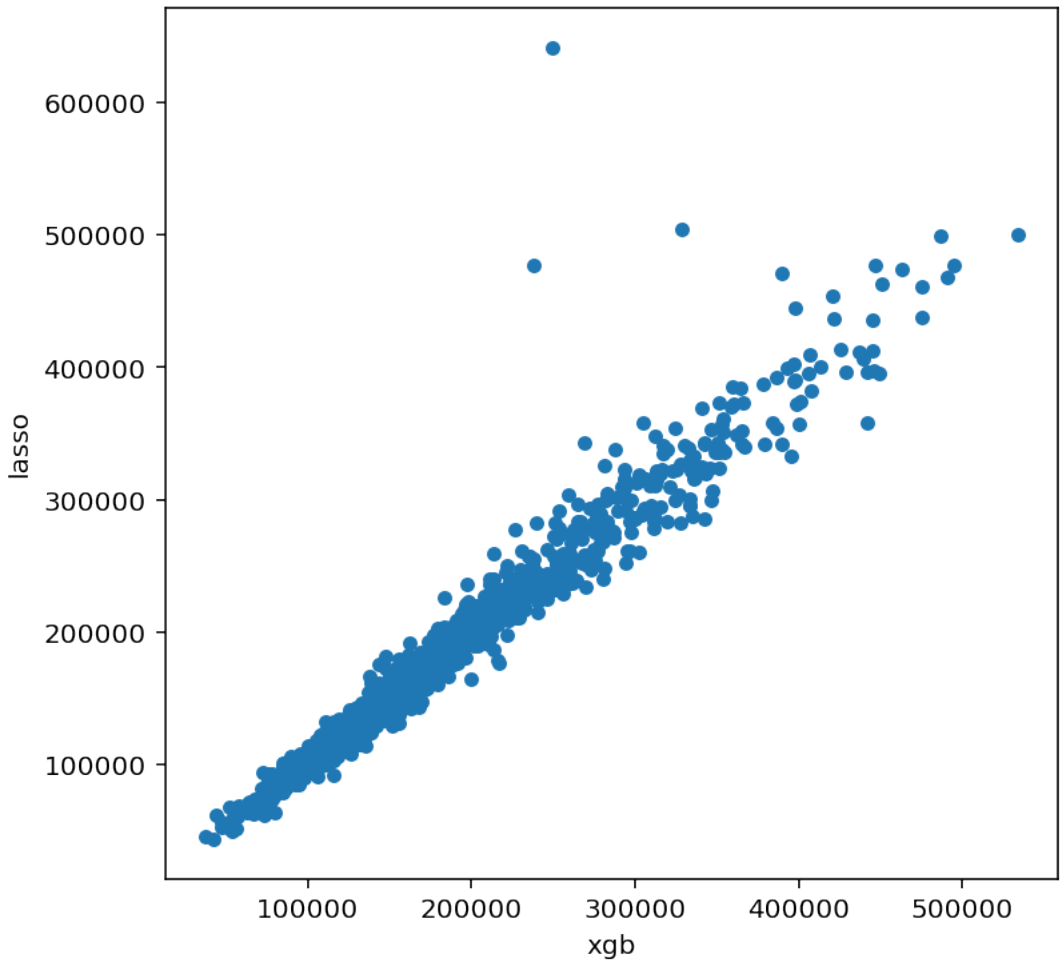
0.1.8 Problem 7. Stacking Models and Exporting Data

```
In [75]: model_xgb = xgb.XGBRegressor(n_estimators = 360, max_depth = 2, learning_rate= 0.1)
         model_xgb.fit(X_train, y)
```

```
Out[75]: 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=2, min_child_weight=1, missing=None, n_estimators=360,
                n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
                reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                silent=True, subsample=1)
```

```
In [76]: xgb_preds = np.expm1(model_xgb.predict(X_test))
         lasso_preds = np.expm1(model_lasso.predict(X_test))

         predictions = pd.DataFrame({"xgb":xgb_preds, "lasso":lasso_preds})
```

```
predictions.plot(x="xgb", y="lasso", kind="scatter")
plt.show()
```




```
In [77]: preds = 0.7*lasso_preds + 0.3*xgb_preds
         solution = pd.DataFrame({"id":test.Id, "SalePrice":preds})
         solution.to_csv("ridge_sol.csv", index = False)
```

0.1.9 Problem 8. Extra Exploration

Keras

```
In [78]: from keras.layers import Dense
         from keras.models import Sequential
         from keras.regularizers import l1
         from sklearn.preprocessing import StandardScaler
         from sklearn.model_selection import train_test_split
         import os
```

```
os.environ["TF_CPP_MIN_LOG_LEVEL"]="3"

X_train = StandardScaler().fit_transform(X_train)
X_tr, X_val, y_tr, y_val = train_test_split(X_train, y, random_state=3)


X_tr.shape
```

```
Out[78]: (1095, 288)
```

```
In [79]: X_tr
```

```
Out[79]: array([[ 1.00573733,  0.68066137, -0.46001991, ..., -0.11785113,
                   0.4676514 , -0.30599503],
                 [-1.12520184,  0.60296111,  0.03113183, ..., -0.11785113,
                   0.4676514 , -0.30599503],
                 [-1.12520184, -0.02865265, -0.74027492, ..., -0.11785113,
                   0.4676514 , -0.30599503],
                 ...,
                 [ 0.16426234, -0.87075036, -0.81954431, ..., -0.11785113,
                  -2.13834494, -0.30599503],
                 [ 0.92361154, -0.30038284, -0.44275864, ..., -0.11785113,
                   0.4676514 , -0.30599503],
                 [ 0.83656519,  1.98505948,  0.46455838, ..., -0.11785113,
                   0.4676514 , -0.30599503]])
```

```
In [80]: model = Sequential()
         model.add(Dense(1, input_dim = X_train.shape[1], W_regularizer=l1(0.001)))
         model.compile(loss = "mse", optimizer = "adam")
```

```
/Applications/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: UserWarning: Updat
```

```
In [81]: model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 1)                 289
=================================================================
Total params: 289
Trainable params: 289
Non-trainable params: 0
_________________________________________________________________
```

```
In [82]: hist = model.fit(X_tr, y_tr, validation_data = (X_val, y_val))
```

```
Train on 1095 samples, validate on 365 samples
Epoch 1/1
1095/1095 [==============================] - 0s 347us/step - loss: 147.0251 - val_loss: 149.954
```

```
In [83]: pd.Series(model.predict(X_val)[:,0]).hist()
         plt.show()
```