EE460J_Lab_3

September 25, 2020

1 EE460J Lab 3

Jianchen Gu - jg68927
Anuv Gupta - ag68799
Pulkit Mahajan - pm28838

1.1 Problem 1

> Read Shannon's 1948 paper 'A Mathematical Theory of Communication'. Focus on pages 1-19 (up to Part II), the remaining part is more relevant for communication.http://math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdfSummarize what you learned briefly (e.g. half a page)

A Mathematical Theory of Communication teaches us about the mathematical models behind current data transmission. Currently, the bit, or binary digit, is the chosen unit of data with which to express and model communications. Since the bit is binary, most of the models follow some form of the logarithmic function, which makes the models make sense mathematically, intuitively, and practically. Typically, an information system consists of the following parts: the information source, the transmitter, the channel, the receiver, and the destination. Given a set of predetermined, finite, symbols, we can use mathematical models to predict sequences of symbols governed by the same statistical properties - this is a stochastic process. Stochastic processes are used with n-order approximations to simulate potential messages - generating messages using information such as the probability of certain letters in a language, languages structures (-ed, qu-, etc), or even probabilities of words in a language themselves. Graphically, our information source can also be represented as a Markov Process, where each state of the information has a finite set of states it can go to, and has a state from which it came (the previous state). Amongst Markov Proceses, there exists a special class which we can use for communication theory - a class based off of ergodic properties. Ergodic simply means that every sequence produced using the same finite set of symbols (which symbols are used in the actual message doesn't matter) has the same statistical properties. Ergodic Markov processes also help us when measuring our uncertainty of the outcome. We know that the uncertainty of an event y is not increased by knowledge of an event x for example. We can also use our knowledge of ergodic processes to calculate entropy. This in turn gives us our max possible compression when we encode into an alphabet - the relative entropy, or the ratio of entropy of a source to its maximum value. After the information source has predicted a message, a transducer is needed to either encode or decode the message (in terms of the input and output symbols). A transducer has the an entropy per unit time less than or equal to that of it's input. We can use all of this information to determine that the most efficient coding of a chanel is determined by C/H, where C is the capacity in bits per second, and H is the entry in bits per symbol of the source.

[ ]:

```

```

1.2 Problem 2: Scraping, Entropy and ICML papers

ICML is a top research conference in Machine Learning. Scrape all the pdfs of all ICML 2017 papers from http://proceedings.mlr.press/v70/. 1. What are the top 10 common words in the ICML papers? 2. Let Z be a randomly selected word in a randomly selected ICML paper. Estimate the entropy of Z. 3. Synthesize a random paragraph using the marginal distribution over words. 4. (Extra credit) Synthesize a random paragraph using an n-gram model on words. Synthesize a random paragraph using any model you want. Top five synthesized text paragraphs win bonus (+30 points). #### Download all required PDFs from ICML 2017 - Run Once!

[2]:

```
# import requests
# from bs4 import BeautifulSoup

# # get the contents of the ICML 2017 articles web page and disable encryption␣
↪verification (bc website lacks it)
# getpage= requests.get('https://proceedings.mlr.press/v70/', verify=False)
# # create bs4 object to parse the web page contents
# getpage_soup= BeautifulSoup(getpage.text, 'html.parser')

# # find all hyperlink <a> tags w/ tag string: 'Download PDF'
# all_links= getpage_soup.findAll('a', string='Download PDF')

# # download all pdfs and save on system
# for i, link in enumerate(all_links):
#     # link is bs4 <tag> object, so we access the link itself via it's <href>␣
↪key
#     file_url = link['href']
#     r = requests.get(file_url, stream = True)
#     # download contents by chunks to our designated pdf
#     with open(f'ICML_2017_pdfs/pdf_{i}.pdf','wb') as pdf:
#         for chunk in r.iter_content(chunk_size=1024):
#             if chunk:
#                 pdf.write(chunk)
```

Parse and convert PDFs to text files - Run Once!

[45]:

```
import os
from io import StringIO

from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from pdfminer.pdfdocument import PDFDocument
from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.pdfpage import PDFPage
```

```python
from pdfminer.pdfparser import PDFParser


def convert_pdf_to_string(file_path):
    """
    Function that uses pdfminer.six to parse and convert a pdf
    stored on local machine to a text file
    """
    output_string = StringIO()

    with open(file_path, 'rb') as in_file:
        parser = PDFParser(in_file)
        doc = PDFDocument(parser)
        rsrcmgr = PDFResourceManager()
        device = TextConverter(rsrcmgr, output_string, laparams=LAParams())
        interpreter = PDFPageInterpreter(rsrcmgr, device)
        for page in PDFPage.create_pages(doc):
            interpreter.process_page(page)

    return(output_string.getvalue())


# iterate over all pdfs in ICML 2017 directory
directory = 'ICML_2017_pdfs'
for filename in os.listdir(directory):
    # for generating text files of pdfs
    text_file = open(f"ICML_2017_texts/{filename.split('.')[0]}.txt", 'w')
    n = text_file.write(convert_pdf_to_string(f'ICML_2017_pdfs/{filename}'))
    text_file.close()
```

```
      ␣
↪---------------------------------------------------------------------------

       KeyboardInterrupt                         Traceback (most recent call␣
↪last)

       <ipython-input-45-d7a32c08c7f5> in <module>
        34     # for generating text files of pdfs
        35     text_file = open(f"ICML_2017_texts/{filename.split('.')[0]}.
↪txt", 'w')
   ---> 36     n = text_file.write(convert_pdf_to_string(f'ICML_2017_pdfs/
↪{filename}'))
        37     text_file.close()


       <ipython-input-45-d7a32c08c7f5> in convert_pdf_to_string(file_path)
```

```
     24         interpreter = PDFPageInterpreter(rsrcmgr, device)
     25         for page in PDFPage.create_pages(doc):
---> 26             interpreter.process_page(page)
     27
     28     return(output_string.getvalue())


    ~/anaconda3/lib/python3.7/site-packages/pdfminer/pdfinterp.py in process_page(self, page)
    840         self.device.begin_page(page, ctm)
    841         self.render_contents(page.resources, page.contents, ctm=ctm)
--> 842         self.device.end_page(page)
    843         return
    844


    ~/anaconda3/lib/python3.7/site-packages/pdfminer/converter.py in end_page(self, page)
     46         assert isinstance(self.cur_item, LTPage)
     47         if self.laparams is not None:
---> 48             self.cur_item.analyze(self.laparams)
     49         self.pageno += 1
     50         self.receive_layout(self.cur_item)


    ~/anaconda3/lib/python3.7/site-packages/pdfminer/layout.py in analyze(self, laparams)
    678         textboxes = list(self.group_textlines(laparams, textlines))
    679         if -1 <= laparams.boxes_flow and laparams.boxes_flow <= +1 and textboxes:
--> 680             self.groups = self.group_textboxes(laparams, textboxes)
    681             assigner = IndexAssigner()
    682             for group in self.groups:


    ~/anaconda3/lib/python3.7/site-packages/pdfminer/layout.py in group_textboxes(self, laparams, boxes)
    655             plane.remove(obj1)
    656             plane.remove(obj2)
--> 657             dists = [ (c,d,obj1,obj2) for (c,d,obj1,obj2) in dists
    658                       if (obj1 in plane and obj2 in plane) ]
    659             for other in plane:


    ~/anaconda3/lib/python3.7/site-packages/pdfminer/layout.py in <listcomp>(.0)
    656             plane.remove(obj2)
```

```
    657             dists = [ (c,d,obj1,obj2) for (c,d,obj1,obj2) in dists
--> 658                       if (obj1 in plane and obj2 in plane) ]
    659             for other in plane:
    660                 dists.append((0, dist(group, other), group, other))


    KeyboardInterrupt:
```

1. Text analysis, Top 10 Most Common Words and Their Occurences

[47]:
```python
from sklearn.feature_extraction.text import CountVectorizer
import re

corpus = []
directory = 'ICML_2017_texts'
for filename in os.listdir(directory):
    # stringify all text files and add them to the list of documents
    with open(f'{directory}/{filename}', 'r') as file:
        # clean out newlines and numbers
        text = file.read().replace('\n', ' ')
        text = re.sub(r'[\d]+', ' ', text)
        text = re.sub(r"[.!,'\";:\?]+", '', text)
        corpus.append(text)

# Create a Vectorizer Object, with lowercase specified
vectorizer = CountVectorizer(lowercase=True)
vectorizer.fit(corpus)

# transform our corpus and tally + sort word frequencies
bag_of_words = vectorizer.transform(corpus)
sum_words = bag_of_words.sum(axis=0)
# generate tuples of words and their frequencies
words_freq = [(word, sum_words[0, idx]) for word, idx in vectorizer.vocabulary_.
 ↪items()]
# sort tuples using frequencies as key value
words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)

print('Top 10 Most Common Words:')
counter = 0
for word, freq in words_freq:
    # watch out for 'cid' pdfminer conversion text (potential math character␣
 ↪conversion)
    if word == 'cid':
        continue
    if counter == 10:
        break
    print(f'{word} ->', freq)
```

```
        counter += 1
```

```
Top 10 Most Common Words:
the -> 90206
of -> 44751
and -> 39194
in -> 32221
to -> 29902
is -> 24286
for -> 21540
we -> 19926
that -> 14728
with -> 13109
```

2. Entropy of Z, a Randomly Selected Word in a Randomly Selected ICML Paper

[49]:
```
import numpy as np
from scipy.stats import entropy

# known article probability distributions (uniformly distributed)
article_probabilities = [1/434 for i in range(434)]
# list of marginal probabilities for EACH set of words in EACH article
word_prob_by_article = []

directory = 'ICML_2017_texts'
for filename in os.listdir(directory):
    # stringify all text files and add them to the list of documents
    with open(f'{directory}/{filename}', 'r') as file:
        # clean out newlines and numbers
        text = file.read().replace('\n', ' ')
        text = re.sub(r'[\d]+', ' ', text)
        text = re.sub(r"[.!,'\";:\?]+", '', text)
        # Create a Vectorizer Object, with lowercase specified
        vectorizer = CountVectorizer(lowercase=True)
        vectorizer.fit([text])
        # transform our corpus and find word frequencies
        bag_of_words = vectorizer.transform([text])
        sum_words = bag_of_words.sum(axis=0)
        # generate tuples of words and their frequencies
        words_freq = [(word, sum_words[0, idx]) for word, idx in vectorizer.
↪vocabulary_.items()]

        # set up for finding marginal probabilities of words in this current␣
↪article
        words = []
        word_probabilities = []
        num_words = 0
```

```
        # get a list of all words
        for word, freq in words_freq:
            words.append(word)
            num_words += freq

        # calculate marginal probabilities of words in this current article
        for word, freq in words_freq:
            word_probabilities.append(freq/num_words)

        # add list of word probability distribution to the main list
        word_prob_by_article.append(word_probabilities)

# now we have a list of EACH word probability distribution for EACH article
# calculate conditional entropy using known article distribution and article␣
↪word distribution
entropy = 0
for i, a_p in enumerate(article_probabilities):
    entropy += a_p * sum(-p * np.log2(p) for p in word_prob_by_article[i])

print(f'Estimated Entropy of Z: {entropy}')
```

```
      ␣
↪---------------------------------------------------------------------------

       ValueError                                Traceback (most recent call␣
↪last)

       <ipython-input-49-257c4350b1e8> in <module>
        17         # Create a Vectorizer Object, with lowercase specified
        18         vectorizer = CountVectorizer(lowercase=True)
    ---> 19         vectorizer.fit([text])
        20         # transform our corpus and find word frequencies
        21         bag_of_words = vectorizer.transform([text])


       ~/anaconda3/lib/python3.7/site-packages/sklearn/feature_extraction/text.
↪py in fit(self, raw_documents, y)
      1163         """
      1164         self._warn_for_unused_params()
   -> 1165         self.fit_transform(raw_documents)
      1166         return self
      1167


       ~/anaconda3/lib/python3.7/site-packages/sklearn/feature_extraction/text.
↪py in fit_transform(self, raw_documents, y)
```

```
   1197
   1198         vocabulary, X = self._count_vocab(raw_documents,
-> 1199                                           self.fixed_vocabulary_)
   1200
   1201         if self.binary:


 ~/anaconda3/lib/python3.7/site-packages/sklearn/feature_extraction/text.
↪py in _count_vocab(self, raw_documents, fixed_vocab)
   1127             vocabulary = dict(vocabulary)
   1128             if not vocabulary:
-> 1129                 raise ValueError("empty vocabulary; perhaps the␣
↪documents only"
   1130                                  " contain stop words")
   1131


 ValueError: empty vocabulary; perhaps the documents only contain stop␣
↪words
```

3. Synthesize a Random Paragraph Using the Marginal Distribution Over Words

[50]:
```
import random

# track list of words and their marginal probabilities
words = []
word_probabilities = []
num_words = 0

# get a list of all words
for word, freq in words_freq:
    words.append(word)
    num_words += freq

# calculate their marginal probabilities
for word, freq in words_freq:
    word_probabilities.append(freq/num_words)

# generate random 50 word paragraph using the words' marginal probabilities
generated_words = random.choices(words, weights=word_probabilities, k=50)
print(' '.join(generated_words))
```

```
better visited query perhaps st combines etl com of standard the log query which
difference mk entries paulev log we less and dense there order triplets space
priority priority time in use in the direction trees approximation foundations
obstacle the different the of like closest than algo random reason index
```

1.3 Problem 3: Starting in Kaggle

1. Let's start with our first Kaggle submission in a playground regression competition. Make an account to Kaggle and find https://www.kaggle.com/c/house-prices-advanced-regression-techniques/

[17]:
```
import pandas as pd
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␣
↪notebook
%matplotlib inline

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

[17]:
```
   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 MoSold  \
0         Lvl    AllPub  …        0    NaN   NaN         NaN       0      2
1         Lvl    AllPub  …        0    NaN   NaN         NaN       0      5
2         Lvl    AllPub  …        0    NaN   NaN         NaN       0      9
3         Lvl    AllPub  …        0    NaN   NaN         NaN       0      2
4         Lvl    AllPub  …        0    NaN   NaN         NaN       0     12

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

[5 rows x 81 columns]
```

[18]:
```
test.head()
```

[18]:
```
     Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0  1461          20       RH         80.0    11622   Pave   NaN      Reg
1  1462          20       RL         81.0    14267   Pave   NaN      IR1
2  1463          60       RL         74.0    13830   Pave   NaN      IR1
3  1464          60       RL         78.0     9978   Pave   NaN      IR1
4  1465         120       RL         43.0     5005   Pave   NaN      IR1

  LandContour Utilities  … ScreenPorch PoolArea PoolQC  Fence MiscFeature  \
0         Lvl    AllPub  …         120        0    NaN  MnPrv         NaN
1         Lvl    AllPub  …           0        0    NaN    NaN        Gar2
2         Lvl    AllPub  …           0        0    NaN  MnPrv         NaN
3         Lvl    AllPub  …           0        0    NaN    NaN         NaN
4         HLS    AllPub  …         144        0    NaN    NaN         NaN

  MiscVal MoSold  YrSold  SaleType  SaleCondition
0       0      6    2010        WD         Normal
1   12500      6    2010        WD         Normal
2       0      3    2010        WD         Normal
3       0      6    2010        WD         Normal
4       0      1    2010        WD         Normal

[5 rows x 80 columns]
```

[20]:
```
all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],
                      test.loc[:,'MSSubClass':'SaleCondition']))
```

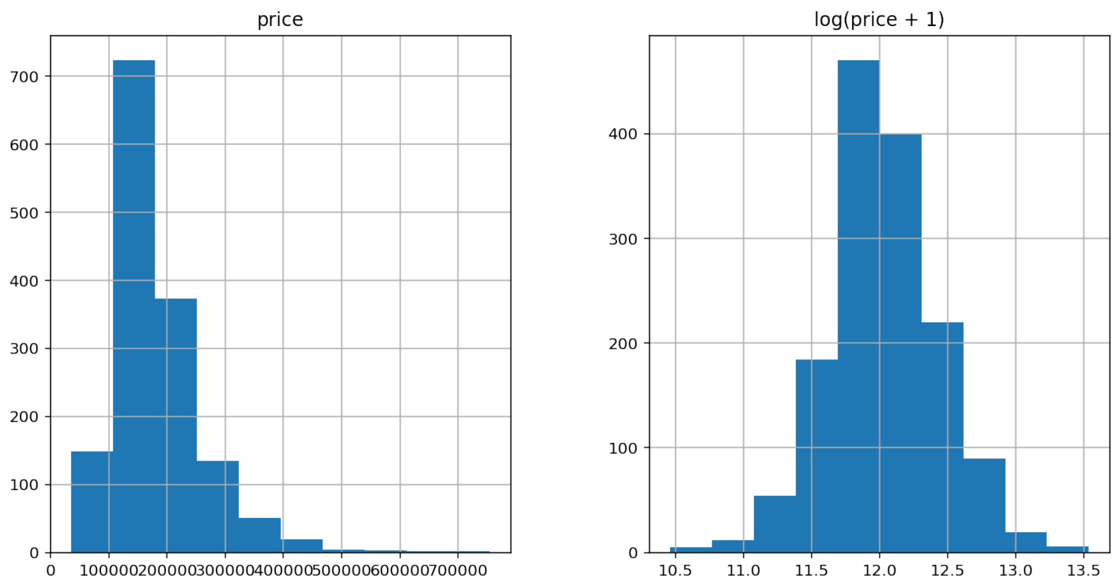
Data preprocessing:

We're not going to do anything fancy here:

```
First I'll transform the skewed numeric features by taking log(feature + 1) - this will make t
Create Dummy variables for the categorical features
Replace the numeric missing values (NaN's) with the mean of their respective columns
```

[21]:
```
matplotlib.rcParams['figure.figsize'] = (12.0, 6.0)
prices = pd.DataFrame({"price":train["SalePrice"], "log(price + 1)":np.
 ↪log1p(train["SalePrice"])})
prices.hist()
```

[21]:
```
array([[<AxesSubplot:title={'center':'price'}>,
        <AxesSubplot:title={'center':'log(price + 1)'}>]], dtype=object)
```



[23]:
```
#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])
```

[24]:
```
all_data = pd.get_dummies(all_data)
```

[25]:
```
#filling NA's with the mean of the column:
all_data = all_data.fillna(all_data.mean())
```

[26]:
```
#creating matrices for sklearn:
X_train = all_data[:train.shape[0]]
X_test = all_data[train.shape[0]:]
y = train.SalePrice
```

2. Follow the data preprocessing steps from https://www.kaggle.com/apapiu/house-prices-advanced-regression-techniques/regularized-linear-models. Then run a ridge regression using = 0.1. Make a submission of this prediction, what is the RMSE you get?(Hint: remember to exponentiate np.expm1(ypred) your predictions)

[27]:
```python
from sklearn.linear_model import Ridge, RidgeCV, ElasticNet, LassoLarsCV
from sklearn.model_selection import cross_val_score
import csv

def rmse_cv(model):
    rmse= np.sqrt(-cross_val_score(model, X_train, y,␣
 ↪scoring="neg_mean_squared_error", cv = 5))
    return(rmse)

# set up a simple ridge regression with alpha=0.1
model_ridge = Ridge(alpha=0.1)
model_ridge.fit(X_train, y)
model_ridge_prediction = np.expm1(model_ridge.predict(X_test))
print('Ridge Regression w/ alpha=0.1')
print(model_ridge_prediction)

# set up for csv file
fields = ['Id', 'SalePrice']
rows = []
for i,p in enumerate(model_ridge_prediction):
    rows.append([test['Id'][i],p])

# writing to csv file
with open('lab_3_q3_pt2.csv', 'w') as csvfile:
    # creating a csv writer object
    csvwriter = csv.writer(csvfile)
    # writing the fields
    csvwriter.writerow(fields)
    # writing the data rows
    csvwriter.writerows(rows)

# Kaggle score
print('\nKaggle submission RMSE: 0.13565')
```

```
Ridge Regression w/ alpha=0.1
[11.72517592 11.96283179 12.12635587 … 12.06416499 11.6994246
 12.2948578 ]

Kaggle submission RMSE: 0.13565
```

3. Compare 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?

[28]:
```python
from sklearn.linear_model import Lasso, LassoCV

# optimize the alpha value with multiple reruns using prior selected alpha
# alphas = [0.05, 0.1, 0.3, 1, 3, 5, (10), 15, 30, 50, 75]
```

```
# alphas = [5, 5.5, 6, 6.5, 7, 7.5, (8), 8.5, 9, 9.5, 10]
alphas = [7.5, 7.6, 7.7, 7.8, 7.9, 8, 8.1, 8.2, 8.3, 8.4, 8.5]

model_ridgeCV = RidgeCV(alphas=alphas).fit(X_train, y)
print(f'Est RidgeCV RMSE: {rmse_cv(model_ridgeCV).mean()}')
print(f'RidgeCV selected alpha={model_ridgeCV.alpha_}\n')

# alphas = [1, 0.1, 0.001, (0.0005)]
alphas = [0.0003, 0.0005, 0.001, 0.0015]

model_lassoCV = LassoCV(alphas=alphas).fit(X_train, y)
print(f'Est LassoCV RMSE: {rmse_cv(model_lassoCV).mean()}')
print(f'LassoCV selected alpha={model_lassoCV.alpha_}\n')

# generate predictions with cross validated optimized alphas for ridge and␣
↪lasso models
model_ridgeCV_prediction = np.expm1(model_ridgeCV.predict(X_test))
print(f'Ridge Regression using Cross Validation w/ alpha={model_ridgeCV.
↪alpha_}')
print(model_ridgeCV_prediction)
model_lassoCV_prediction = np.expm1(model_lassoCV.predict(X_test))
print(f'\nLasso Regression using Cross Validation w/ alpha={model_lassoCV.
↪alpha_}')
print(model_lassoCV_prediction)
```

```
Est RidgeCV RMSE: 0.010753687745495396
RidgeCV selected alpha=7.5

Est LassoCV RMSE: 0.012008632069007461
LassoCV selected alpha=0.0003

Ridge Regression using Cross Validation w/ alpha=7.5
[11.71773367 11.87078544 12.01694751 … 11.99813083 11.61329855
 12.36838636]

Lasso Regression using Cross Validation w/ alpha=0.0003
[11.7661303  11.8542499  12.00409147 … 12.06818024 11.70325298
 12.44335173]
```

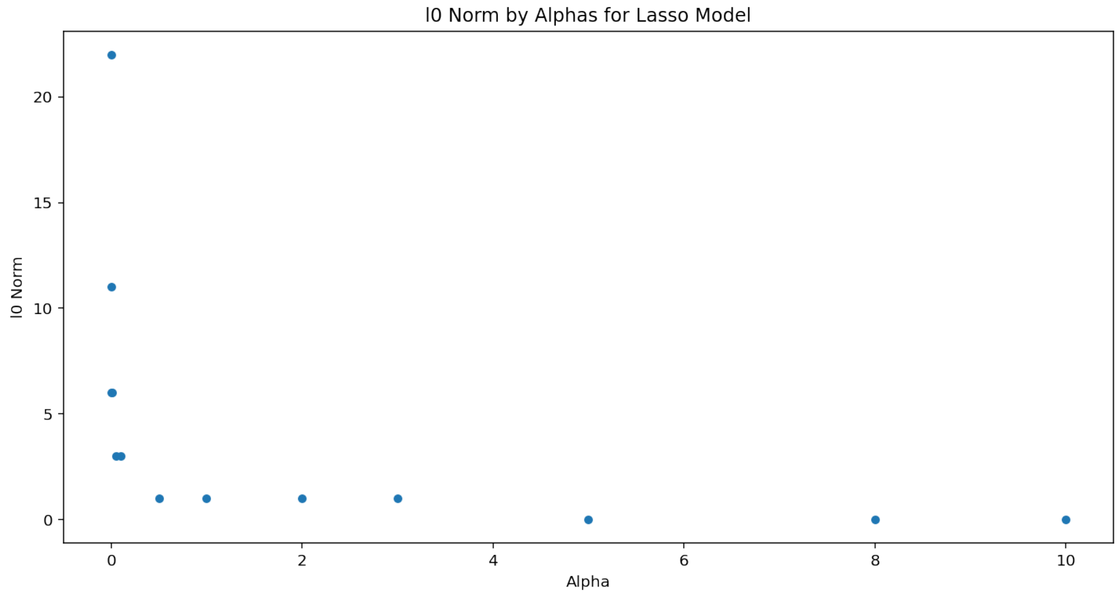
4. Plot the l0 norm (number of nonzeros) of the coefficients that lasso produces as you vary the strength of regularization parameter alpha.

[29]:
```
# declare a range of alphas to be tried
alphas = [10, 8, 5, 3, 2, 1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005]
l0_norms_list = []

# calculate l0 Norms
for a in alphas:
```

```python
    model_lasso = Lasso(alpha=a).fit(X_train, y)
    l0_norm = 0
    for coef in model_lasso.coef_:
        if coef != 0:
            l0_norm += 1
    l0_norms_list.append([l0_norm, a])

# plot l0 Norms to their respective alphas
df_l0_norms = pd.DataFrame(l0_norms_list, columns = ['l0 Norm', 'Alpha'])
df_l0_norms.plot(title='l0 Norm by Alphas for Lasso Model', x='Alpha', y='l0␣
 ↪Norm', kind = 'scatter')
plt.show()
```



5. 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? (We will be discussing ensembling more, later in the class, but you can start playing with it now).

[31]:
```python
import math

def stack(X_train, X_test, y, num_sections, col_suffix):
    ridge_col_name = f'ridge_regression_{col_suffix}'
    lasso_col_name = f'lasso_regression_{col_suffix}'
    alphas_r = [7.5, 7.6, 7.7, 7.8, 7.9, 8, 8.1, 8.2, 8.3, 8.4, 8.5]
    alphas_l = [0.0003, 0.0005, 0.001, 0.0015]
    # augment X_train with regression columns
```

```
    X_train_aug = X_train.reindex(columns = X_train.columns.tolist() +␣
↪[ridge_col_name,lasso_col_name])
    training_length = len(X_train)
    section_size = math.floor(training_length / num_sections)
    last_section_size = training_length - (section_size * (num_sections - 1))
    for i in range(num_sections):
        x_section = None
        y_section = None
        x_test_section = None
        if i == num_sections - 1:
            x_section = X_train.iloc[i*section_size:]
            y_section = y[i*section_size:]
            x_test_section = X_train.iloc[0:i*section_size]
        else:
            x_section = X_train.iloc[i*section_size:(i+1)*section_size]
            y_section = y[i*section_size:(i+1)*section_size]
            x_test_section = X_train.iloc[np.r_[0:i*section_size,␣
↪(i+1)*section_size:training_length]]
        # print(list(ranges(x_test_section.index.values.tolist())),␣
↪len(x_test_section))
        section_ridgeCV_model = RidgeCV(alphas=alphas_r).fit(x_section,␣
↪y_section)
        section_ridgeCV_prediction = np.expm1(section_ridgeCV_model.
↪predict(x_test_section))
        section_lassoCV_model = LassoCV(alphas=alphas_r, tol=0.01).
↪fit(x_section, y_section)
        section_lassoCV_prediction = np.expm1(section_lassoCV_model.
↪predict(x_test_section))
        curr_s_s = section_size
        if i == num_sections - 1:
            curr_s_s = last_section_size
        k = 0
        for j in range(i*section_size, i*section_size + curr_s_s):
            if k >= len(section_ridgeCV_prediction):
                break
            X_train_aug.loc[j,ridge_col_name] = section_ridgeCV_prediction[k]
            X_train_aug.loc[j,lasso_col_name] = section_lassoCV_prediction[k]
            k += 1
    # augment X_test with regression columns
    X_test_aug = X_test.reindex(columns = X_test.columns.tolist() +␣
↪[ridge_col_name,lasso_col_name])
    test_ridgeCV_model = RidgeCV(alphas=alphas_r).fit(X_train, y)
    test_ridgeCV_prediction = np.expm1(test_ridgeCV_model.predict(X_test))
    test_lassoCV_model = LassoCV(alphas=alphas_l, tol=0.01).fit(X_train, y)
    test_lassoCV_prediction = np.expm1(test_lassoCV_model.predict(X_test))
    for i in range(len(X_test_aug)):
```

```
        X_test_aug.loc[i, ridge_col_name] = test_ridgeCV_prediction[i]
        X_test_aug.loc[i, lasso_col_name] = test_lassoCV_prediction[i]
    return [ RidgeCV(alphas=alphas_r).fit(X_train_aug, y),␣
↪LassoCV(alphas=alphas_l).fit(X_train_aug, y), X_train_aug, X_test_aug ]
```

[32]:
```
stacked_models = stack(X_train, X_test, y, 5, 'round_1')
X_train_aug = stacked_models[2]
X_test_aug = stacked_models[3]
stacked_ridgeCV_model = stacked_models[0]
stacked_ridgeCV_prediction = np.expm1(stacked_ridgeCV_model.predict(X_test_aug))
stacked_lassoCV_model = stacked_models[1]
stacked_lassoCV_prediction = np.expm1(stacked_lassoCV_model.predict(X_test_aug))
```

[34]:
```
fields = ['Id', 'SalePrice']
rows = []
for i,p in enumerate(stacked_ridgeCV_prediction):
    rows.append([test['Id'][i],p])
with open('lab_3_q3_pt5_r.csv', 'w') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(fields)
    csvwriter.writerows(rows)
```

[35]:
```
fields = ['Id', 'SalePrice']
rows = []
for i,p in enumerate(stacked_lassoCV_prediction):
    rows.append([test['Id'][i],p])
with open('lab_3_q3_pt5_l.csv', 'w') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(fields)
    csvwriter.writerows(rows)
```

[36]:
```
print(f'Est Stacked RidgeCV RMSE: {rmse_cv(stacked_ridgeCV_model).mean()}')
print(f'Est Stacked LassoCV RMSE: {rmse_cv(stacked_lassoCV_model).mean()}')

print(f'Stacked RidgeCV RMSE Kaggle Score: 0.12517')
print(f'Stacked LassoCV RMSE Kaggle Score: 0.12458')
```

```
Est Stacked RidgeCV RMSE: 0.010753687745495396
Est Stacked LassoCV RMSE: 0.012008632069007461
Stacked RidgeCV RMSE Kaggle Score: 0.12517
Stacked LassoCV RMSE Kaggle Score: 0.12458
```

[ ]:
```

```

6. Install XGBoost (Gradient Boosting) and train a gradient boosting regression. What score can you get just from a single XGB? (you will need to optimize over its parameters). We will discuss boosting and gradient boosting in more detail later. XGB is a great friend to all good Kagglers!

[38]:
```
import xgboost as xgb

dtrain = xgb.DMatrix(X_train_aug, label = y)
dtest = xgb.DMatrix(X_test_aug)

params = {"max_depth":2, "eta":0.1}
# tune params
xgb_cv_model = xgb.cv(params, dtrain,  num_boost_round=500,␣
↪early_stopping_rounds=100)
xgb_model = xgb.XGBRegressor(n_estimators=360, max_depth=2, learning_rate=0.1)
xgb_model.fit(X_train_aug, y)
xgb_model_preds = np.expm1(xgb_model.predict(X_test_aug))
```

[39]:
```
fields = ['Id', 'SalePrice']
rows = []
for i,p in enumerate(xgb_model_preds):
    rows.append([test['Id'][i],p])
with open('lab_3_q3_pt6.csv', 'w') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(fields)
    csvwriter.writerows(rows)
```

[40]:
```
print(f'Est XGBoost RMSE: {rmse_cv(xgb_model).mean()}')
print(f'XGBoost RMSE Kaggle Score: 0.13213')
```

```
Est XGBoost RMSE: 0.009768892961427213
XGBoost RMSE Kaggle Score: 0.13213
```

7. Do your best to get the more accurate model. Try feature engineering and stacking many models. You are allowed to use any public tool in python. No non-python tools allowed.

[41]:
```
# stacked_models = stack(X_train_aug, X_test_aug, y, 5, 'round_2')
```

[42]:
```
weighted_predictions = 0.7 * stacked_lassoCV_prediction + 0.3 * xgb_model_preds
```

[43]:
```
fields = ['Id', 'SalePrice']
rows = []
for i,p in enumerate(weighted_predictions):
    rows.append([test['Id'][i],p])
with open('lab_3_q3_pt7.csv', 'w') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(fields)
    csvwriter.writerows(rows)
```

[44]:
```
print(f'XGBoost RMSE Kaggle Score: 0.12258')
```

```
XGBoost RMSE Kaggle Score: 0.12258
```

We attempted stacking more than the two suggested models, including multiple lasso and ridge models; we created a stacking function to allow repeated stacking of multiple data sets, and also to allow splitting the training set into various sizes of sections (to avoid overfitting). We did not see any improvement in the RMSE Kaggle score with these methods. XGBoost at first seemed to yield a high RMSE score, close to the simple ridge regression's RMSE score. XGBoost did not yield different results when used on the stacked training & test data. However, the final method we attempted was a weighted combination of predictions from multiple models. We experimented with various weights on three models: the stacked lasso regression predictions, stacked ridge regression predictions, and XGBoost model predictions. The method that improved our Kaggle RMSE score the most was a weighted combination of predictions from the stacked lasso regression model (70%) and the XGBoost model (30%).

Our initial ridge regression RMSE Kaggle score was 0.13565.
Our stacked ridge regression RMSE Kaggle score was 0.12517.
Our stacked lasso regression RMSE Kaggle score was 0.12458.
Our XGBoost model RMSE Kaggle score was 0.12854.
Our weighted model (0.7 * stacked_lasso_reg + 0.3 * xgb_model) RMSE Kaggle score was **0.12258**.
The final weighted model produced the best score we could achieve.

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```