Lab3 for EE460J

Author: Brandon Pham, Adriana Pedroza, Jack Diao

Data-preprocessing based on: https://www.kaggle.com/apapiu/regularized-linear-models (https://www.kaggle.com/apapiu/regularized-linear-models)

Document Formatting

Please note that some cells are not to be run in Jupyter/other notebook. For example, the cell with `ICMLCrawler` class is meant to be run in a terminal with scrapy, with the line `scrapy runspider <file_containing_spider_code_*only*.py>` . Other cells *are* meant to be run in the notebook. The generated lab document (i.e., the PDF) makes these cells easily identifiable.

Q1: Paper Summary

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. Summarize what you learned briefly (e.g. half a page).

The engineering communication question largely concerns how to measure the capacity to transmit information. With discrete systems, this is given by a logarithmic function of the number of allowed signals of duration T. If the communication channel is considered a state machine with a finite number of states, this can be solved by the determinant. The information source can also be described mathematically, as a Markoff or other stochastic process, with probabilities for each respective message. For example, the English language can be "approximated" with respect to letter order, grammatical structures, word frequencies, and so on. These representations get exponentially more difficult to solve as the complexity of the "alphabet" of the channel increases. Because these concepts can be described mathematically we can also compute the choice, uncertainty, and entropy of the communication channel and information source. The same can be said for the encoding and decoding of the message.

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/ (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 (+50 points for homeworks and labs).

Part 1. Parse Documents for Top 10 Words

Spider for Crawling ICML

ICMLCrawler downloads all pdfs to the listed directory.

In [ ]:

```
from scrapy import Spider
from scrapy.selector import Selector
from paper import PapercrawlerItem
import pdfminer
import urllib.request
from pdfminer.pdfparser import PDFParser
from pdfminer.pdfdocument import PDFDocument
from pdfminer.pdfpage import PDFPage
from pdfminer.pdfpage import PDFTextExtractionNotAllowed
from pdfminer.pdfinterp import PDFResourceManager
from pdfminer.pdfinterp import PDFPageInterpreter
from pdfminer.pdfdevice import PDFDevice

class ICMLCrawler(Spider):
    name = "ICMLCrawler"
    start_urls = ["http://proceedings.mlr.press/v70/",]
    prefix = r"C:\Users\chrom\Documents\HW\data sci\lab3\files"
    allowed_domains = ["proceedings.mlr.press"]
    def parse(self, response):
        papers = Selector(response).xpath('//*[@id="content"]/div/div[@class="paper"]')

        titles = Selector(response).xpath('//*[@id="content"]/div/div[@class="paper"]/p[1]')
        paper_url = Selector(response).xpath('//*[@id="content"]/div/div[@class="paper"]/p[3]/a[2]')

        for title, pdf, sup in zip(titles, papers, paper_url):
            item = PapercrawlerItem()
            item['title'] = title.xpath('text()').extract()[0]

            item['url'] = sup.xpath('@href').extract()[0]

            file_url = response.css('.downloadline::attr(href)').get()
            file_url = response.urljoin(file_url)

            path = item['url'].split('/')[-1]

            urllib.request.urlretrieve(item['url'], r"C:/Users/chrom/Documents/HW/data sci/lab3/files/" + path)
            yield item
```

In [ ]:

```
import os, glob

import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
nltk.download('punkt')

from tika import parser
import warnings
warnings.filterwarnings("ignore")

words = []
path = '/Users/chrom/Documents/HW/data sci/lab3/files/'
#write a for-loop to open many files
print('starting')
num_files = 0;
for filename in glob.glob(os.path.join(path, '*.pdf')):
    num_files += 1
    print('reading pdf number ' + str(num_files))

    data_pdf = parser.from_file(filename)
    text = data_pdf["content"]
    #The word_tokenize() function will break our text phrases into individual words
    tokens = word_tokenize(text)
    #list which contains punctuation we wish to clean
    punctuations = ['(',')',';',':','[',']',',','.','?','=']
    #We create a list comprehension which only returns a list of words that are and
NOT IN punctuations.
    _words = [word for word in tokens if not word in punctuations]
    words = words + _words
    print('total number of words: ' + str(len(words)))
    #print(words)
```

In [1]:

```
def sort_by_frequency(alist):
    dict = {}
    count, itm = 0, ''
    for item in reversed(alist):
        dict[item] = dict.get(item, 0) + 1
    list = sorted(dict.items(), key=lambda x: x[1], reverse=True)
    return list

words = []

# open file and read the content in a list
with open('listOfPDFWords.txt', 'r', encoding='utf8') as filehandle:
    for line in filehandle:
        # remove linebreak which is the last character of the string
        currentWord = line[:-1]

        # add item to the list
        words.append(currentWord)

reverse_words = sort_by_frequency(words)
for i in range(0,10):
    print(reverse_words[i])
```

```
('the', 137516)
('of', 76551)
('and', 66822)
('to', 48514)
('a', 48226)
('is', 40861)
('in', 39211)
('for', 32531)
('that', 24843)
('1', 24397)
```

Part 2. Calculate Entropy of Word

In [2]:

```
from scipy.stats import entropy
# open file and read the content in a list
with open('./gitstuff/data_science_lab/Lab3/listOfPDFWords.txt', 'r', encoding="utf8") as filehandle:
    for line in filehandle:
        # remove linebreak which is the last character of the string
        currentWord = line[:-1]

        # add item to the list
        words.append(currentWord)

num_words = len(words)
reverse_words = sort_by_frequency(words)
list_of_frequency = []
for word_freq_pair in reverse_words:
    list_of_frequency.append(word_freq_pair[1]/num_words)

print("entropy: " + str(entropy(list_of_frequency, base=num_words)))
```

```
entropy: 0.5076717274179208
```

Part 3. Synthesize Random Paragraph

In [3]:

```
from random import randint
num_words_para = 100
paragraph = ""
for word in range(num_words_para):
    pick = randint(0, num_words)
    i = 0
    done = False
    total = 0
    while i < num_words and not done:
        pair = reverse_words[i]
        total += pair[1]
        if total > pick:
            paragraph += pair[0]
            paragraph += " "
            done = True
        i += 1
print(paragraph)
```

```
of range systematic the finite-length user of where component DeepGraph in way c
onsider priors randomised A row method train of sleep where assumption Figure co
mplexity explanation noting .001 m � not · the adversary 2012 & In descent concu
rrent 0.1 * filter this 1973 the with complicated Ensemble we to learning and and
a batching > in the independently marginal If even 0 the and Bachem �-spectral i
s route 5 2016a of 5. problematic learning ” A. loss Engineering or is Laplacian
which are dis- theory order turn Arora original kernel g to The a :484-489 – Est
imating sequence with
```

Q2: Data Preprocessing

In [4]:

```
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 note
book
%matplotlib inline
```

In [5]:

```
train = pd.read_csv("./input/train.csv")
test = pd.read_csv("./input/test.csv")
```

In [6]:

```
train.head()
```

Out[6]:

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

5 rows × 81 columns

In [7]:

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

In [8]:

```
#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 skew
ness
skewed_feats = skewed_feats[skewed_feats > 0.75]
skewed_feats = skewed_feats.index

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

```
C:\Users\chrom\Anaconda3\lib\site-packages\ipykernel_launcher.py:11: RuntimeWarn
ing: invalid value encountered in log1p
  # This is added back by InteractiveShellApp.init_path()
```

In [9]:

```
all_data = pd.get_dummies(all_data)
```

In [10]:

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

In [11]:

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

Q3: Lasso and Ridge Regression

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?

In [12]:

```
from sklearn.linear_model import Ridge, Lasso, RidgeCV, ElasticNet, LassoCV, LassoL
arsCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
```

In [13]:

```
model_ridge = Ridge(alpha = 0.1)
```

In [14]:

```
model_ridge.fit(X_train,y)
pred = model_ridge.predict(X_train) # using X_train since X_test has no SalesPrice
to compare to
rmse = np.sqrt(mean_squared_error(y,pred))
```

In [15]:

```
print("Ridge Pred: " + str(pred))
print("Ridge RMSE: " + str(rmse))
```

```
Ridge Pred: [12.2418857  12.18340581 12.28628096 ... 12.54982583 11.8645212
 11.85610025]
Ridge RMSE: 0.0921195558564051
```

In [16]:

```
model_lasso = Lasso(alpha = 0.1)
model_lasso.fit(X_train, y)
pred = model_lasso.predict(X_train) # using X_train since X_test has no SalesPrice
to compare to
rmse = np.sqrt(mean_squared_error(y,pred))
```

In [17]:

```
print("Lasso Pred: " + str(pred))
print("Lasso RMSE: " + str(rmse))
```

```
Lasso Pred: [12.32599498 11.99049435 12.28477897 ... 12.06396816 11.78011099
 11.75582923]
Lasso RMSE: 0.20753905105566584
```

In [18]:

```
alphas = [0.0005, 0.001, 0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]
```

In [19]:

```
model_ridge = RidgeCV(alphas = alphas).fit(X_train, y)
best_alpha = model_ridge.alpha_
model_ridge = Ridge(alpha=best_alpha)
model_ridge.fit(X_train,y)
print("Best alpha for Ridge:  " + str(best_alpha))
print("Score from best alpha: " + str(model_ridge.score(X_train, y)))
```

```
Best alpha for Ridge:  10.0
Score from best alpha: 0.9299587783131811
```

In [20]:

```
model_lasso = LassoCV(alphas = alphas).fit(X_train, y)
best_alpha = model_lasso.alpha_
model_lasso = Lasso(alpha=best_alpha)
model_lasso.fit(X_train,y)
print("Best alpha for Ridge:  " + str(best_alpha))
print("Score from best alpha: " + str(model_lasso.score(X_train, y)))
```

```
C:\Users\chrom\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:197
8: FutureWarning: The default value of cv will change from 3 to 5 in version 0.2
2. Specify it explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

```
Best alpha for Ridge:  0.0005
Score from best alpha: 0.9308222956470585
```

Q4: Plot l0-norm vs alphas

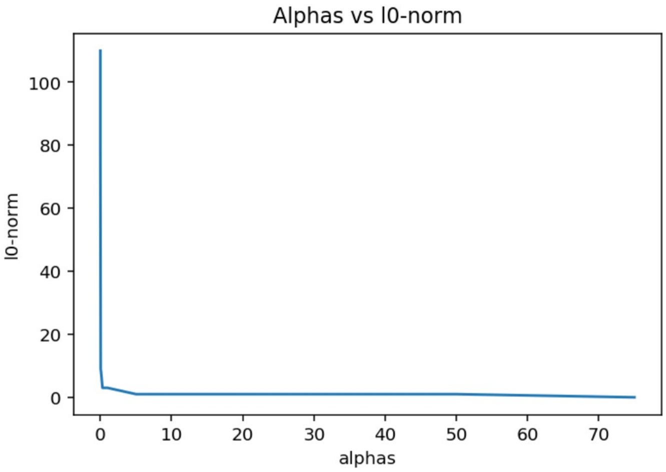
Plot the l0 norm (number of nonzeros) of the coefficients that lasso produces as you vary the strength of regularization parameter alpha.

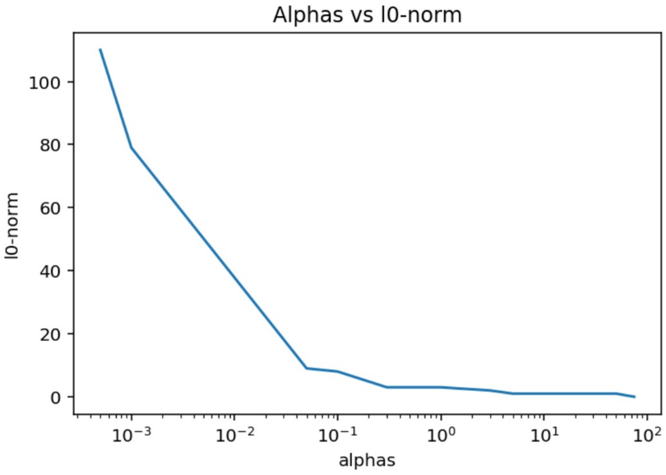
In [21]:

```
l0_norms = []

for alpha in alphas:
    model_lasso = Lasso(alpha=alpha)
    model_lasso.fit(X_train, y)
    coefs = model_lasso.coef_
    l0_norm = 0
    for coef in coefs:
        if coef != 0:
            l0_norm += 1
    l0_norms.append(l0_norm)

plt.plot(alphas, l0_norms)
plt.title("Alphas vs l0-norm")
plt.xlabel("alphas")
plt.ylabel("l0-norm")
plt.show()
plt.plot(alphas, l0_norms)
plt.title("Alphas vs l0-norm")
plt.xscale('log')
plt.xlabel("alphas")
plt.ylabel("l0-norm")
plt.show()
```





Q5: Ensembling and Stacking

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).

In [22]:

```
model_lasso = Lasso(alpha=10)
model_lasso.fit(X_train, y)
output_lasso = model_lasso.predict(X_train)
model_ridge = Ridge(alpha=0.0005)
model_ridge.fit(X_train, y)
output_ridge = model_ridge.predict(X_train)
```

In [23]:

```
X_train['lasso'] = pd.Series(output_lasso, index=X_train.index)
X_train['ridge'] = pd.Series(output_ridge, index=X_train.index)

model_ridge = Ridge(alpha=0.0005)
model_ridge.fit(X_train, y)
print("Ensemble Ridge Score: " + str(model_ridge.score(X_train,y)))
```

```
Ensemble Ridge Score: 0.9473151078894951

C:\Users\chrom\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithC
opyWarning:
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/stabl
e/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
C:\Users\chrom\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWithC
opyWarning:
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/stabl
e/indexing.html#indexing-view-versus-copy
```

Q6: XGBoost

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!

In [24]:

```
import xgboost as xgb
```

In [25]:

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

In [26]:

```
model_xgb = xgb.XGBRegressor(max_depth=2) #the params were tuned using xgb.cv
model_xgb.fit(X_train, y)
```

Out[26]:

```
XGBRegressor(base_score=0.5, booster=None, colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints=None,
             learning_rate=0.300000012, max_delta_step=0, max_depth=2,
             min_child_weight=1, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=0, num_parallel_tree=1,
             objective='reg:squarederror', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=None,
             validate_parameters=False, verbosity=None)
```

In [27]:

```
xgb_preds = model_xgb.predict(X_train)

def score(true_values, pred):
    u = mean_squared_error(true_values,pred)
    true_mean = [true_values.mean()] * true_values.shape[0]
    v = mean_squared_error(true_values,true_mean)
    score = 1 - (u/v)

    return score

print("XGB Score: " + str(score(y,xgb_preds)))
```

```
XGB Score: 0.9521292802011088
```

Q7: Improve model

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.

In [28]:

```
dtrain = xgb.DMatrix(X_train, label = y)
dtest = xgb.DMatrix(X_test)

params = {"max_depth":2, "eta":0.1}
```

In [29]:

```
n_estimators = list(range(1,500,200))
max_depths = list(range(2,11,3))
etas = list(range(1,10,3))
etas = [x / 10 for x in etas]

print(n_estimators)
print(max_depths)
print(etas)
```

```
[1, 201, 401]
[2, 5, 8]
[0.1, 0.4, 0.7]
```

In [30]:

```
best_score = 100
best_params = None
for estimator in n_estimators:
    for depth in max_depths:
        for eta in etas:
            params = {"n_estimators": estimator, "max_depth": depth, "eta": eta}
            print(params)
            model = xgb.cv(params, dtrain,  num_boost_round=250, early_stopping_rou
nds=100)
            if model['test-rmse-mean'].min() <= best_score:
                print("New best params!")

                best_score = model['test-rmse-mean'].min()
                best_params = params
print(best_score)
print(best_params)
```

```
{'n_estimators': 1, 'max_depth': 2, 'eta': 0.1}
New best params!
{'n_estimators': 1, 'max_depth': 2, 'eta': 0.4}
{'n_estimators': 1, 'max_depth': 2, 'eta': 0.7}
{'n_estimators': 1, 'max_depth': 5, 'eta': 0.1}
{'n_estimators': 1, 'max_depth': 5, 'eta': 0.4}
{'n_estimators': 1, 'max_depth': 5, 'eta': 0.7}
{'n_estimators': 1, 'max_depth': 8, 'eta': 0.1}
{'n_estimators': 1, 'max_depth': 8, 'eta': 0.4}
{'n_estimators': 1, 'max_depth': 8, 'eta': 0.7}
{'n_estimators': 201, 'max_depth': 2, 'eta': 0.1}
New best params!
{'n_estimators': 201, 'max_depth': 2, 'eta': 0.4}
{'n_estimators': 201, 'max_depth': 2, 'eta': 0.7}
{'n_estimators': 201, 'max_depth': 5, 'eta': 0.1}
{'n_estimators': 201, 'max_depth': 5, 'eta': 0.4}
{'n_estimators': 201, 'max_depth': 5, 'eta': 0.7}
{'n_estimators': 201, 'max_depth': 8, 'eta': 0.1}
{'n_estimators': 201, 'max_depth': 8, 'eta': 0.4}
{'n_estimators': 201, 'max_depth': 8, 'eta': 0.7}
{'n_estimators': 401, 'max_depth': 2, 'eta': 0.1}
New best params!
{'n_estimators': 401, 'max_depth': 2, 'eta': 0.4}
{'n_estimators': 401, 'max_depth': 2, 'eta': 0.7}
{'n_estimators': 401, 'max_depth': 5, 'eta': 0.1}
{'n_estimators': 401, 'max_depth': 5, 'eta': 0.4}
{'n_estimators': 401, 'max_depth': 5, 'eta': 0.7}
{'n_estimators': 401, 'max_depth': 8, 'eta': 0.1}
{'n_estimators': 401, 'max_depth': 8, 'eta': 0.4}
{'n_estimators': 401, 'max_depth': 8, 'eta': 0.7}
0.125162
{'n_estimators': 401, 'max_depth': 2, 'eta': 0.1}
```

In [31]:

```
n_estimators = best_params["n_estimators"]
optimal_depth = best_params["max_depth"]
optimal_eta = best_params['eta']

print(n_estimators)
print(optimal_depth)
print(optimal_eta)
```

```
401
2
0.1
```

Q10: Submit

As in the real in-class Kaggle competition (which will be next), 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 (that you will submit as a pdf file), 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.

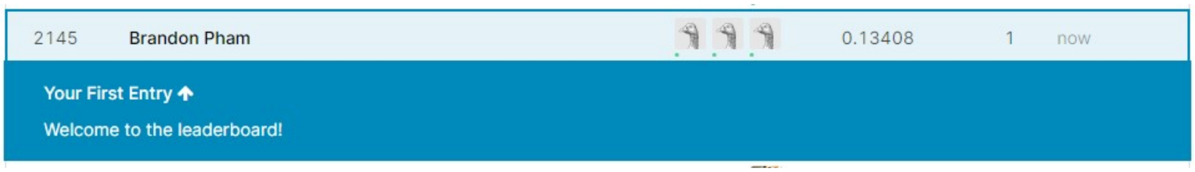
In [32]:
```
model_xgb = xgb.XGBRegressor(n_estimators=401, max_depth=2, eta=0.1) #the params we
re tuned using xgb.cv
model_xgb.fit(X_train, y)
preds = np.expm1(model_xgb.predict(X_test))
print(preds)
```

```
[120981.55 157237.64 186772.23 ... 155967.28 118881.76 226109.53]
```

In [33]:
```
solution = pd.DataFrame({"id":test.Id, "SalePrice":preds})
solution.to_csv("ridge_sol.csv", index = False)
```

In [35]:
```
from IPython.display import Image
Image(filename='submission.png')
```

Out[35]:



2145 Brandon Pham 0.13408 1 now

Your First Entry

Welcome to the leaderboard!

In [ ]: