Lab 3

Kevin Liang - kgl392

Matthew Tan - mmt2338

In [50]:

```
import numpy as np
import pandas as pd
import sklearn as sk
import matplotlib.pyplot as plt
import seaborn as sns
import math
import sympy
import scipy
import urllib2
import urllib
from bs4 import BeautifulSoup
import pdfminer
from cStringIO import StringIO
from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.converter import TextConverter
from sklearn.feature_extraction.text import CountVectorizer
from pdfminer.layout import LAParams
from pdfminer.pdfpage import PDFPage
from collections import Counter
import random
```

In [150]:

```
# Projects vector 'b' onto vector 'a'
def proj(b, a):
    return (np.dot(a, b) / np.linalg.norm(a) ** 2) * a

# Question 1
v1 = [1,2,3,4]
v2 = [0,1,0,1]
v3 = [1,4,3,6]
v4 = [2,11,6,15]

matrix = sympy.Matrix([v1,v2,v3,v4])

#*********************************************************************


# Part 1 - Create a vector not in S.
print "Question 1 - Part 1 (Create Vector not in S) \n"

# calculate the null space of the matrix
nullspace_matrix = matrix.nullspace()

# any vector in the nullspace matrix is orthogonal to the subspace S
```

```
# and therefore not a vector in S.
print np.array(nullspace_matrix[0])
print

#**********************************************************************


# Part 1 - how to check if a new vector is in S?
print "Question 1 - Part 1 (Check if vector is in S) \n"

random_vector = np.random.rand(1,4)[0]
check_matrix = sympy.Matrix([v1,v2,v3,v4,random_vector])
rref_check_matrix = check_matrix.T.rref()

print "Generated random vector. Then I ran an rref on the matrix. If there
was a pivot in the last column, then the vector is in S.\n"
if 4 in rref_check_matrix[1]:
    print "Vector is in S"
else:
    print "Vector is not in S"
print
#**********************************************************************


# Part 2 - Find the dimension of the subspace S
print "Question 1 - Part 2 \n"

col_space = matrix.columnspace()

print "Dimension: " + str(len(col_space)) + '\n'

#**********************************************************************


# Part 3 - Find an orthonormal basis for the subspace S
print "Question 1 - Part 3 \n"

# calculate orthonormal basis
arr = np.array(matrix).astype(np.float)
orth_basis = scipy.linalg.orth(arr)

#print orth_basis

# obtain orthonormal basis vectors (columns of previously calculated matrix
)
vec0 = []
vec1 = []
for i in range(len(orth_basis)):
    vec0.append(orth_basis[i][0])
    vec1.append(orth_basis[i][1])

vec0 = np.array(vec0).astype(np.float)
vec1 = np.array(vec1).astype(np.float)
orth_basis_vectors = np.array([vec0,vec1])

print "Orthonormal Basis:" + "\n" + str(orth_basis_vectors)
print

#**********************************************************************
```

```
# Part 4 - Solve the optimization problem
print "Question 1 - Part 4 \n"

zStar = [1,0,0,0]

# Generate a random 1x4 vector
random_vector = np.random.rand(1, 4)[0]

S = np.array(matrix.rref()[0]).astype(np.float)      # S is a basis for {v1,
v2, v3, v4}

# Generate an orthoganal basis for S, then project 'random_vector' onto eac
h of them
orth_comp = S[0] - proj(S[0], S[1])     # the component of S[0] orthogonal tc
S[1]
vec_in_space = proj(random_vector, orth_comp) + proj(random_vector, S[1])

random_vector = np.random.rand(1, 4)[0]
v0 = vec_in_space
v1 = proj(random_vector, orth_comp) + proj(random_vector, S[1])

norm_v0 = np.linalg.norm(v0)
norm_v1 = np.linalg.norm(v1)

b0 = (1 / (norm_v0**2)) * (np.dot(v0.transpose(), zStar))
b1 = (1 / (norm_v0**2)) * (np.dot(v1.transpose(), zStar))
print "V0: " + str(v0)
print "V1: " + str(v1)
print "X = Beta1*V0 + Beta2*V1"
print "Beta: " + str((b0, b1))
```

```
Question 1 - Part 1 (Create Vector not in S)

[[-3]
 [0]
 [1]
 [0]]

Question 1 - Part 1 (Check if vector is in S)

Generated random vector. Then I ran an rref on the matrix. If there was a p
ivot in the last column, then the vector is in S.

Vector is in S

Question 1 - Part 2

Dimension: 2

Question 1 - Part 3

Orthonormal Basis:
[[-0.24011927 -0.05990306 -0.35992538 -0.89955994]
 [ 0.8581727  -0.29094143  0.27628983 -0.32024463]]

Question 1 - Part 4

V0: [ 0.34784761  0.22771828  1.04354284  0.9234135 ]
V1: [ 0.27929964  0.33413116  0.83789892  0.89273044]
```

```
X = Beta1*V0 + Beta2*V1
Beta: (0.16450368955552808, 0.132086061315878)
```

In [12]:

```
# Question 2

# download all documents
url = "http://proceedings.mlr.press/v70/"
html = urllib2.urlopen(url).read()
soup = BeautifulSoup(html, "html.parser")
pdf_names = []
text_names = []

tags = soup('a')
for tag in tags:
    name = tag.get('href')
    if ".pdf" in name and "supp" not in name:
        dest = "D:/Kevin
Liang/Documents/1_UT_SENIOR/UT_AUSTIN_FALL_2017/EE_379K/Lab3/PDFS/" + name.
split("/")[-1]
        urllib.urlretrieve(name, dest)
        pdf_names.append(dest)
        text_names.append("D:/Kevin
Liang/Documents/1_UT_SENIOR/UT_AUSTIN_FALL_2017/EE_379K/Lab3/TEXTS/" + name
.split("/")[-1].split(".")[0] + " .txt")
```

In [21]:

```
# Convert PDF to Text
def convert(fname, pages=None):
    if not pages:
        pagenums = set()
    else:
        pagenums = set(pages)

    output = StringIO()
    manager = PDFResourceManager()
    converter = TextConverter(manager, output, laparams=LAParams())
    interpreter = PDFPageInterpreter(manager, converter)

    infile = file(fname, 'rb')
    for page in PDFPage.get_pages(infile, pagenums):
        interpreter.process_page(page)
    infile.close()
    converter.close()
    text = output.getvalue()
    output.close
    return text
```

In [22]:

```
for name in pdf_names:
    text = convert(name)
    f = open("D:/Kevin
Liang/Documents/1_UT_SENIOR/UT_AUSTIN_FALL_2017/EE_379K/Lab3/TEXTS/" + name
.split("/")[-1].split(".")[0] + " .txt", "w")
    f.write(text)
    f.close()
```

In [145]:

```
# Question 2 - Part 1 - 10 most common words in the ICML papers
path = "D:/Kevin
Liang/Documents/1_UT_SENIOR/UT_AUSTIN_FALL_2017/EE_379K/Lab3/TEXTS/"
# three_txt = [path + "achab17a .txt", path + "acharya17a .txt", path +
"achiam17a .txt"]

dictionary_BABY = {}

vec = CountVectorizer(input='filename', max_df = .95, min_df = .05, token_pa
ttern = "[a-z]{4,}")
dtm = vec.fit_transform(text_names)
vocab = vec.get_feature_names()
dtm_a = dtm.toarray().tolist()
for index,word in enumerate(vocab):
    dictionary_BABY[word] = dtm[:,index].sum()

print dict(Counter(dictionary_BABY).most_common(10))

# To calculate the 10 most frequent words, we used CounterVectorizer where
we set specific restrictions to capture important words.
# If a word occured more than 95% of the time or less than 5% of the time w
e would ignore it.
# All words only contain alphabetical characters and must be 4 characters o
r longer.
```

```
{u'function': 5569, u'matrix': 4607, u'algorithm': 7158, u'neural': 5206, u
'models': 4237, u'problem': 4128, u'time': 5077, u'model': 8149, u'data': 7
658, u'networks': 4236}
```

In [153]:

```
# Part 2 - find entropy of randomly selected word in a randomly selected
ICML paper
random_text = []
random_text.append(text_names[random.randrange(0,len(text_names))])

dictionary_BABY = {}

vec = CountVectorizer(input='filename', token_pattern = "[a-z]{4,}")
dtm = vec.fit_transform(random_text)
vocab = vec.get_feature_names()
dtm_a = dtm.toarray().tolist()
for index,word in enumerate(vocab):
    dictionary_BABY[word] = dtm[:,index].sum()

total_words = sum(Counter(dictionary_BABY).values())

pmf = dictionary_BABY
for x in pmf:
    pmf[x] = float(pmf[x])/float(total_words)

entropy = 0

for x in pmf:
    entropy += pmf[x]*math.log(pmf[x],2)*-1

print "Entropy: " + str(entropy)
```

```
Entropy: 9.1839856431
```

In [85]:

```
# Calculate PMF of all the words
dictionary_BABY = {}
words = []
probs = []

vec = CountVectorizer(input='filename', token_pattern = "[a-z]{4,}")
dtm = vec.fit_transform(text_names)
vocab = vec.get_feature_names()
dtm_a = dtm.toarray().tolist()
for index,word in enumerate(vocab):
    dictionary_BABY[word] = dtm[:,index].sum()

total_words = sum(Counter(dictionary_BABY).values())

pmf = dictionary_BABY
for x in pmf:
    pmf[x] = float(pmf[x])/float(total_words)
    words.append(x)
    probs.append(pmf[x])
```

In [148]:

```
# Part 3 - synthesize a random paragraph using the marginal distribution ov
er words

paragraph = ""

paragraph_index = []

for x in range(200):
    index = np.random.choice(np.arange(len(probs)), p = probs)
    paragraph_index.append(index)
    paragraph += words[index] + " "

print paragraph
```

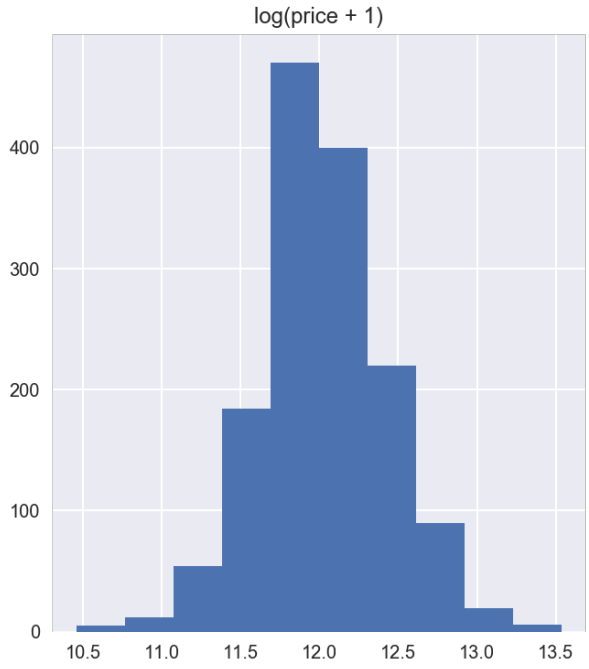
```
full pushed ablations tering associative latest speci bayesian have from se
ction work replaces kernel dimensional proposed introduced conference algor
ithmic large reviewers that injects injected true conditions satisfy regula
riser quantity side network intelligence linear networks
representationneighborhood this feature face neuron naive wise points unsup
ervised similarity remain preprint explicit framework small constraint mist
ake equivalence constants reconstruction clustering each shallow action cas
e trials above output needed applied close distribution innovation learn st
rongly method people uncertainty recallthattr probability rather that model
research problem chen results reported similarities gradient cient mnist ma
rkov probably last this vated gent agent than dataset tive artzi goals deno
te exposition word descent discussions than terms garber strongly examined
until theano inference running susceptible instances bergstra model term se
ntation synaptic variance sydney techniques rfou best model exact bear dist
ri theoretic utilize lemma bengio stochastic fast neural harley high lter g
radient without networks contextual dynamical structure network recurrent p
arametrized gaussianity sigkdd possible stochastic times imalmapping study
```

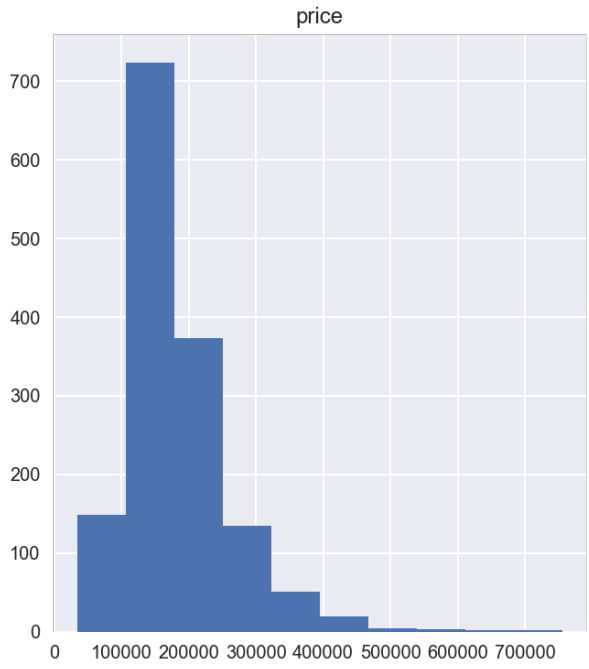
online upper state arnold arora problem depending special matrix nite variance environments just proceedings called literature issue gradient preferred learning sponding constant experienced regression method quantitative complement random summa based consistency least roger recovery models tasks freitas notes schrodt hence hachiya center densely method based this

In [119]:

```
# Question 3

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 notebook
%matplotlib inline

train = pd.read_csv("D:/Kevin
Liang/Documents/1_UT_SENIOR/UT_AUSTIN_FALL_2017/EE_379K/Lab3/train.csv")
test = pd.read_csv("D:/Kevin
Liang/Documents/1_UT_SENIOR/UT_AUSTIN_FALL_2017/EE_379K/Lab3/test.csv")

train.head()

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

matplotlib.rcParams['figure.figsize'] = (12.0, 6.0)
prices = pd.DataFrame({"price":train["SalePrice"], "log(price + 1)":np.log1
p(train["SalePrice"])})
prices.hist()

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

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

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

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

all_data = pd.get_dummies(all_data)

#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
```



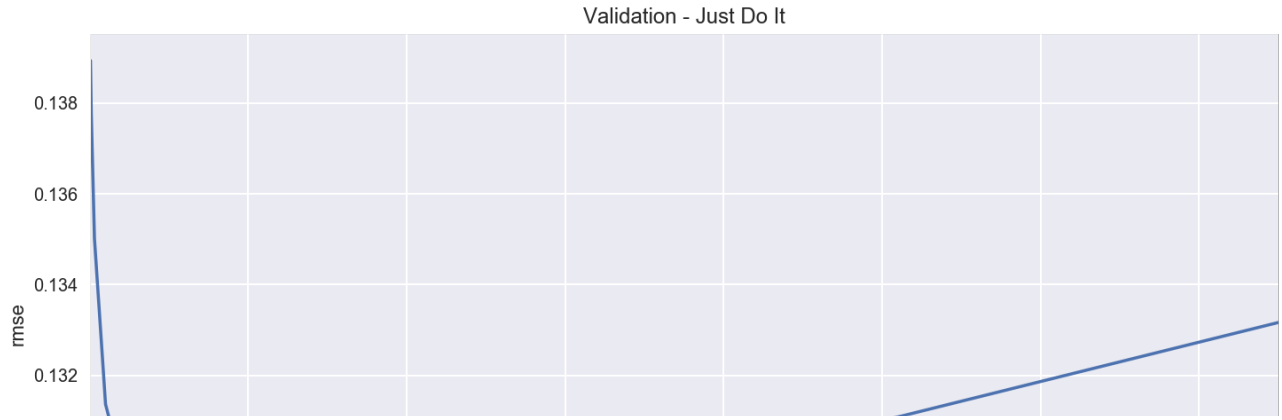


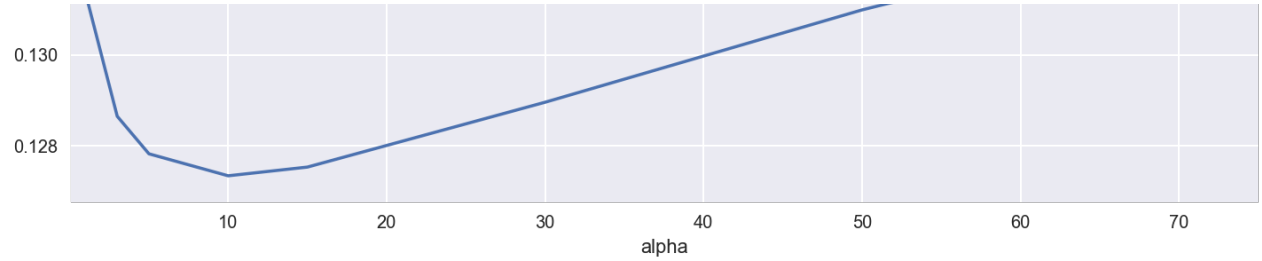
In [131]:

```
from sklearn.linear_model import Ridge, RidgeCV, ElasticNet, LassoCV, Lasso
LarsCV
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", cv = 5))
    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]

cv_ridge = pd.Series(cv_ridge, index = alphas)
cv_ridge.plot(title = "Validation - Just Do It")
plt.xlabel("alpha")
plt.ylabel("rmse")
```

Out[131]:

```
<matplotlib.text.Text at 0x107f4b70>
```





In [133]:

```
# Question 3 - Part 2

modelRidge = Ridge(alpha = 0.1)
modelRidge.fit(X_train,y)
predictions = np.expm1(modelRidge.predict(X_test))
df = pd.DataFrame(predictions)
df.to_csv("predictions.csv")

print "The RSME I get based on a Ridge model with an alpha value of 0.1: "
+ str(0.13029)
```

```
The RSME I get based on a Ridge model with an alpha value of 0.1: 0.13029
```

In [140]:

```
# Question 3 - Part 3

model_lasso = LassoCV(alphas = [1, 0.1, 0.001, 0.0005]).fit(X_train, y)
rmse_cv(model_lasso).mean()
pred = np.expm1(model_lasso.predict(X_test))
df1 = pd.DataFrame(pred)
df1.to_csv("predictions.csv")

print "The RSME I get based on a lasso model with 4 alpha values of [1, 0.1
, 0.001, 0.0005]: .12096"
```

```
The RSME I get based on a lasso model with 4 alpha values of [1, 0.1, 0.001
, 0.0005]: .12096
```