Lab3_Problem3

February 22, 2020

0.1 Lab 3:

- Jacob Stokes - jts3867
- Andrew Annestrand - ata758
- Musa Rafik - mar6827

0.2 Preprocessing

[1]:
```
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

from sklearn.model_selection import KFold
kfolds = KFold(n_splits=10, shuffle=True, random_state=42)


%config InlineBackend.figure_format = 'retina' #set 'png' here when working on
 ↪notebook
%matplotlib inline
```

[2]:
```
train = pd.read_csv("../input/train.csv")
test = pd.read_csv("../input/test.csv")
```

[3]:
```
train.head()
```

[3]:
```
   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]
```

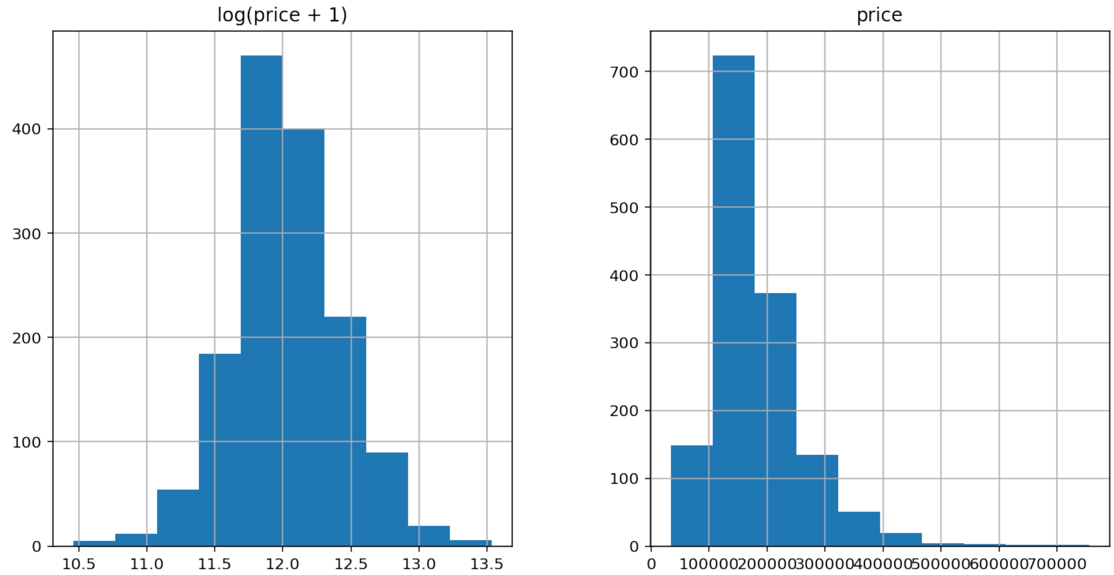
[4]:
```
all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],
                      test.loc[:,'MSSubClass':'SaleCondition']))
```

###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 the features more normal
- Create Dummy variables for the categorical features
- Replace the numeric missing values (NaN's) with the mean of their respective columns

[5]:
```
matplotlib.rcParams['figure.figsize'] = (12.0, 6.0)
prices = pd.DataFrame({"price":train["SalePrice"], "log(price + 1)":np.
 ↪log1p(train["SalePrice"])})
prices.hist()
```

[5]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fbd9edaf160>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbd9cd22128>]],
      dtype=object)
```



[6]:
```
#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])
```

[7]:
```
all_data = pd.get_dummies(all_data)
```

[8]:
```
#filling NA's with the mean of the column:
all_data = all_data.fillna(all_data.mean())
```

[9]:
```
#creating matrices for sklearn:
X_train = all_data[:train.shape[0]]
X_test = all_data[train.shape[0]:]
y = train.SalePrice
```

[10]:
```
from sklearn.linear_model import Ridge, RidgeCV, ElasticNet, Lasso, LassoCV,␣
↪LassoLarsCV
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
```

```
import seaborn as sns

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

0.3 Problem 2: Simple Ridge Regression

Here we run a ridge regression with alpha set to 0.1.

[11]:
```
ridge_model = Ridge(alpha=0.1)
ridge_model.fit(X_train, y)
print(rmse_cv(ridge_model))


predictions = np.expm1(ridge_model.predict(X_test))
```

```
[0.1088122  0.14368417 0.12308564 0.12385737 0.15028028 0.20652642
 0.14414075 0.11264046 0.13031523 0.08629446]
```

[12]:
```
turn_in = pd.DataFrame(test['Id'])
turn_in['SalePrice'] = predictions

turn_in.to_csv('FirstSubmission.csv',index=False)
# We got a RMSE of 0.13029
```

0.4 Problem 3: Comparing Ridge and Lasso

In this problem we run a CV ridge and lasso model and compare the best scores we can get from each model. From the results we see that Ridge performs better than Lasso generally for this data set.

[13]:
```
# Problem 3

ridgecv_model = RidgeCV()
lassocv_model = LassoCV()

ridgecv_model.fit(X_train, y)
lassocv_model.fit(X_train, y)

print(rmse_cv(ridgecv_model))
print(rmse_cv(lassocv_model))
```

```
[0.10765397 0.14410231 0.11323435 0.12893532 0.14272373 0.18212833
 0.12869769 0.10978299 0.12955362 0.08610172]
[0.18814293 0.21021651 0.17392941 0.17781418 0.17887062 0.20438118
 0.20848141 0.17581564 0.18470512 0.15196888]
```

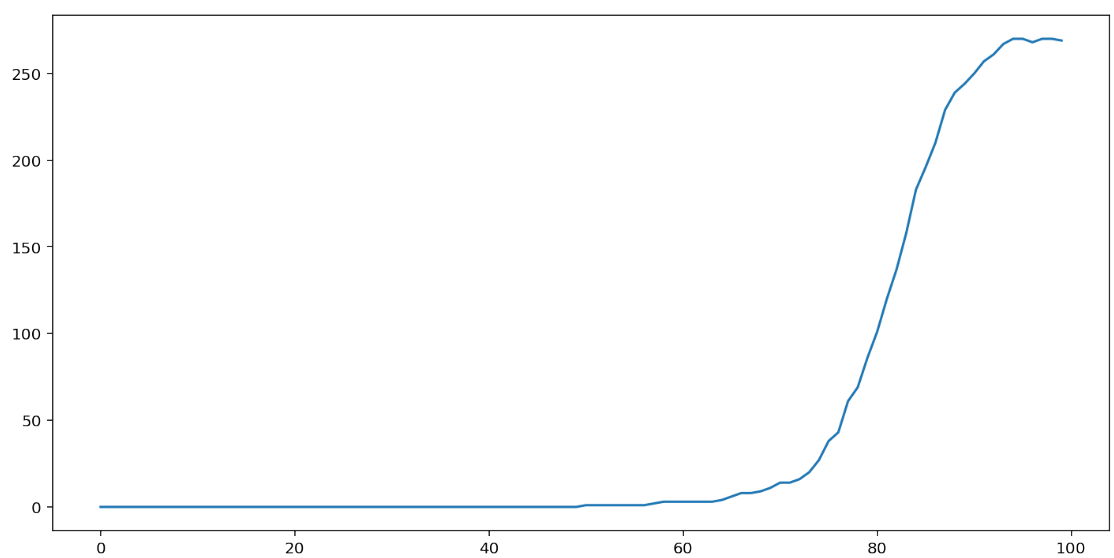
```
[14]: # Best score for Ridge: .1104
      # Best score for Lasso: .1736
```

0.5 Problem 4

For this problem we are training many lasso models and graphing the effect of the magnitude of alphas on coefficients. In our graph, the number of non-zero coefficients is shown to increase as the value of alpha decreases.

```
[15]: # Problem 4

alphas = 10**np.linspace(10,-6,100)*.5
non_zero_coefs = []

for a in alphas:
    lasso_model = Lasso(alpha=a,max_iter=100000)
    lasso_model.fit(X_train, y)

    num_non_zero = 0
    for coef in lasso_model.coef_:
        if (abs(coef) !=0):
            num_non_zero += 1
    non_zero_coefs.append(num_non_zero)


sns.lineplot(list(range(0,100)),non_zero_coefs)
# Shown below, we are plotting the number of non-zero coefficients as our alpha␣
↪size decreases. This graph validates our intuition
# that as our regularization weight increases, more coefficients will zero out.
```

```
[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7fbd93c0dbe0>
```



0.6 Problem 5

For this problem, we train both a lassocv and ridgecv and use their predictions on test data as features for another ridgecv model. This is the essence of "stacking." We stack the results of the models we train so that we can get the best from each and reduce our bias.

[16]:
```
# Problem 5
# Add the outputs of models as features and train a ridge regression on all␣
↪features plus model outputs
stacking_data_train = X_train.copy(deep=True)
stacking_data_train['ridge_predictions'] = pd.Series(ridgecv_model.
↪predict(X_train))
stacking_data_train['lasso_predictions'] = pd.Series(lassocv_model.
↪predict(X_train))

stacking_data_test = X_test.copy(deep=True)
stacking_data_test['ridge_predictions'] = pd.Series(ridgecv_model.
↪predict(X_test))
stacking_data_test['lasso_predictions'] = pd.Series(lassocv_model.
↪predict(X_test))


stacked_ridge_model = RidgeCV()
stacked_ridge_model.fit(stacking_data_train, y)

predictions = np.expm1(stacked_ridge_model.predict(stacking_data_test))
turn_in = pd.DataFrame(test['Id'])
turn_in['SalePrice'] = predictions

turn_in.to_csv('StackedRidgeSubmission.csv',index=False)
```

[17]:
```
# The above model returned a RMSE of 0.12241 which is an improvement over the␣
↪previous one!
```

0.7 Problem 6

For the first part of this problem we just run a default XGBoost and get a score. It's not too great so we find some optimal hyperparametes per other notebooks and train another XGBoost.

[18]:
```
# Problem 6
from xgboost import XGBRegressor

my_model = XGBRegressor(silent=True)
my_model.fit(X_train, y, verbose=False)
print(rmse_cv(my_model))
```

```
predictions = np.expm1(my_model.predict(X_test))
turn_in = pd.DataFrame(test['Id'])
turn_in['SalePrice'] = predictions

turn_in.to_csv('XGBoost_no_tuning.csv',index=False)
```

```
/opt/conda/lib/python3.6/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/conda/lib/python3.6/site-packages/xgboost/core.py:588: FutureWarning:
Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

```
[0.13043751 0.15175771 0.10319462 0.13488255 0.15912402 0.13230677
 0.14952718 0.11621785 0.13778941 0.09205627]
```

[19]:
```
xgb_model = XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=.4603, gamma=.0468, learning_rate=0.05,␣
↪max_delta_step=0,
       max_depth=3, min_child_weight=1.7817, missing=None, n_estimators=2200,
       n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
       reg_alpha=0.4640, reg_lambda=0.8571, scale_pos_weight=1, seed=42,
       silent=True, subsample=0.5213)
print(rmse_cv(xgb_model))

xgb_model.fit(X_train, y)
predictions = np.expm1(xgb_model.predict(X_test))
turn_in = pd.DataFrame(test['Id'])
turn_in['SalePrice'] = predictions

turn_in.to_csv('XGBoost_with_tuning.csv',index=False)
```

```
[0.10940141 0.14424263 0.09646917 0.12177479 0.14218592 0.14066332
 0.13124592 0.11642555 0.12350134 0.08003638]
```

```
/opt/conda/lib/python3.6/site-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/conda/lib/python3.6/site-packages/xgboost/core.py:588: FutureWarning:
Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

- XGBoost MSE no tuning: 0.13904
- XGBoost MSE with tuning: 0.13136

0.8 Problem 7

For this part we are going to try and get the best score we can. First we are going to try and blend 3 different MLR models and see what we get. After that, we will attempt to stack and then blend many models to improve accuracy and prevent overfitting.

[26]:
```
from sklearn.preprocessing import RobustScaler
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import ElasticNetCV

alphas_ridge = list(np.linspace(14.5,15.6,11))
alphas_lasso = [5e-05,.0001,.0002,.0003,.0004,.0005,.0006,.0007,.0008]
alphas_e = [.0001,.0002,.0003,.0004,.0005,.0006,.0007]
l1ratio_e = [.8,.85,.9,.95,.99,1]

ridge = make_pipeline(RobustScaler(), RidgeCV(alphas=alphas_ridge, cv=kfolds))
lasso = make_pipeline(RobustScaler(), LassoCV(alphas=alphas_lasso,␣
↪max_iter=1e7, cv=kfolds, random_state=42))
elasticnet = make_pipeline(RobustScaler(), ElasticNetCV(max_iter=1e7,␣
↪alphas=alphas_e, cv=kfolds, l1_ratio=l1ratio_e))

models = {'Ridge': ridge,
          'Lasso': lasso,
          'ElasticNet': elasticnet}

predictions = {}
scores = {}

for name, model in models.items():
    model.fit(X_train, y)
    predictions[name] = model.predict(X_train)

#     score = rmse_cv(model)
#     scores[name] = (score.mean(), score.std())

print(scores)
```

{}

[27]:
```
from sklearn.metrics import mean_squared_error

blended_predictions = (predictions['ElasticNet'] + predictions['Lasso'] +␣
↪predictions['Ridge'])/3
print(np.sqrt(mean_squared_error(y, blended_predictions)))
```

0.10326322904435709

[28]:
```
test_predictions = {}

for name, model in models.items():
    model.fit(X_train, y)
    test_predictions[name] = np.expm1(model.predict(X_test))

final_predictions = (test_predictions['ElasticNet'] + test_predictions['Lasso']␣
 ↪+ test_predictions['Ridge'])/3
turn_in = pd.DataFrame(test['Id'])
turn_in['SalePrice'] = final_predictions

turn_in.to_csv('BlendingMLRModels.csv',index=False)
```

[29]:
```
# Ok so now we have tried blending and stacking. Let's see if we can put it all␣
 ↪together and climb the leaderboards.

from mlxtend.regressor import StackingCVRegressor

stacking_model = StackingCVRegressor(regressors=(ridge, lasso, elasticnet,␣
 ↪xgb_model), meta_regressor=ridge, use_features_in_secondary=True)
stacking_model.fit(X_train, y)
```

[29]:
```
StackingCVRegressor(cv=5,
                    meta_regressor=XGBRegressor(base_score=0.5,
                                                booster='gbtree',
                                                colsample_bylevel=1,
                                                colsample_bynode=1,
                                                colsample_bytree=0.4603,
                                                gamma=0.0468,
                                                importance_type='gain',
                                                learning_rate=0.05,
                                                max_delta_step=0, max_depth=3,
                                                min_child_weight=1.7817,
                                                missing=None, n_estimators=2200,
                                                n_jobs=1, nthread=None,
                                                objective='reg:linear',
                                                random_state=0, reg…
                                             learning_rate=0.05,
                                             max_delta_step=0, max_depth=3,
                                             min_child_weight=1.7817,
                                             missing=None, n_estimators=2200,
                                             n_jobs=1, nthread=None,
                                             objective='reg:linear',
                                             random_state=0, reg_alpha=0.464,
                                             reg_lambda=0.8571,
                                             scale_pos_weight=1, seed=42,
                                             silent=True, subsample=0.5213,
```

```
                                                  verbosity=1)),
                                   shuffle=True, store_train_meta_features=False,
                                   use_features_in_secondary=True, verbose=0)
```

[31]:
```
# Stacking has worked, now we should get our blended results
blended_stacked_results = (np.expm1(xgb_model.predict(X_test)) + np.
 ↪expm1(stacking_model.predict(np.array(X_test))) + np.expm1(lasso.
 ↪predict(X_test)) + np.expm1(ridge.predict(X_test)) + np.expm1(elasticnet.
 ↪predict(X_test)))/5
print(blended_stacked_results)

turn_in = pd.DataFrame(test['Id'])
turn_in['SalePrice'] = blended_stacked_results
turn_in.to_csv('StackedBlended.csv',index=False)
```

```
[120247.4986126  155354.47027494 180965.91241369 … 168370.46769335
 118019.7948116  222502.40273571]
```

Our final and best MSE from a stacked and blended model was 0.1898 which puts us in the top 18% of competitors. I believe that in order to achieve a better MSE, we would need to do a bit more work in preprocessing our data. We kind of overlooked this part but could tell its importance by looking at other popular notebooks. Other things we learned is that ensembling is an effective strategy that should be used in regression.

Lab3_Problem2

February 22, 2020

[ ]:

```
# Extract pdfs from website

import os
import requests
from urllib.parse import urljoin
from bs4 import BeautifulSoup

url = "http://proceedings.mlr.press/v70/"

# If the folder does not exist, create one automatically
folder_location = './webscraping/'
if not os.path.exists(folder_location):
    os.mkdir(folder_location)

response = requests.get(url)
soup = BeautifulSoup(response.text, "html.parser")
for link in soup.select("a[href$='.pdf']"):
    #Name the pdf files using the last portion of each link
    filename = os.path.join(folder_location, link['href'].split('/')[-1])
    with open(filename, 'wb') as f:
        f.write(requests.get(urljoin(url, link['href'])).content)
```

[145]:

```
from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from pdfminer.pdfpage import PDFPage
from io import StringIO
import os

# Convert a given pdf to text and return the text
def convert_pdf_to_txt(pathname):
    rsrcmgr = PDFResourceManager()
    retstr = StringIO()
    codec = 'utf-8'
    laparams = LAParams()
    device = TextConverter(rsrcmgr, retstr, codec=codec, laparams=laparams)
```

```
    fptr = open(pathname, 'rb')
    interpreter = PDFPageInterpreter(rsrcmgr, device)

    try:
        for page in PDFPage.get_pages(fptr, set(), maxpages=0,␣
↪password="",caching=True, check_extractable=True):
            interpreter.process_page(page)
    except: # Need this for an exception that gets thrown for pdfs that can't␣
↪be converted
        return ""

    text = retstr.getvalue()

    fptr.close()
    device.close()
    retstr.close()
    return text

# Iterate through files in a given directory
# Convert the file to text and then store in a list
def store_text_to_list(path_name, file_extension):
    converted_text_list = []
    count = 0 # Keep track of the number of files converted

    for filename in os.listdir(path_name):
        if filename.endswith(file_extension):
            converted_text = convert_pdf_to_txt(path_name + filename)
            if(converted_text != ""):
                converted_text_list.append(converted_text)
                count += 1

    print("Number of files converted:",count)
    return converted_text_list
```

[146]:
```
# Convert all pdfs in the directory to text
converted_text_list = store_text_to_list('./pdfs/', '.pdf')
```

```
Number of files converted: 720
```

[ ]:
```
import string
from nltk.corpus import words
import nltk

nltk.download('words')

# Iterate through list and create a dictionary with (key, value) pairs being␣
↪(word, frequency)
```

```
# Add a word to dictionary if it's not already there, else update the entry by 1
freq_dict = {}


for document in converted_text_list:
    if(document != ""):
        split_document = document.split()
        for word in split_document:
            if word in words.words():
                if word in freq_dict:
                    freq_dict[word] = freq_dict[word] + 1
                else:
                    freq_dict[word] = 1
```

```
[nltk_data] Downloading package words to /Users/musarafik/nltk_data…
[nltk_data]   Unzipping corpora/words.zip.
```

[ ]:
```
# Sort dictionary in reverse order and create a list of tuples so we can easily␣
↪create a dataframe
sorted_d = sorted(((value, key) for (key,value) in freq_dict.items()),␣
↪reverse=True)
```

[1]:
```
import pickle
# Use pickle to save dictionary so we don't have to keep converting pdfs to␣
↪text everytime we restart notebook

# WARNING: do not uncomment and run this or else b will be loaded with junk
# with open('filename.pickle', 'wb') as handle:
#     pickle.dump(sorted_d, handle, protocol=pickle.HIGHEST_PROTOCOL)

# with open('freq_dict.pickle', 'wb') as handle:
#     pickle.dump(freq_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)

with open('freq_dict.pickle', 'rb') as handle:
    saved_dict = pickle.load(handle)

# b holds all the list of words and their frequencies
with open('filename.pickle', 'rb') as handle:
    b = pickle.load(handle)
```

[104]:
```
import pandas as pd

clean_dict = sorted(((value, key) for (key, value) in saved_dict.items()),␣
↪reverse=True)

# Convert list of tuples to dataframe and display top 15 words/symbols
df = pd.DataFrame(b, columns = ['Frequency', 'Word'])
```

```
df.head(15)
```

[104]:

	Frequency	Word
0	184869	the
1	101466	of
2	86891	and
3	61703	to
4	57986	a
5	55017	is
6	50851	in
7	47783	=
8	44202	for
9	36039	that
10	34678	we
11	29098	with
12	27959	1
13	26949	-
14	24756	+

As we can see from the dataframe, the top ten most frequent words are: 1. 'the' - 184869 occurrences 2. 'of' - 101466 occurrences 3. 'and' - 86891 occurrences 4. 'to' - 61703 occurrences 5. 'a' - 57986 occurrences 6. 'is' - 55017 occurrences 7. 'in' - 50851 occurrences 8. 'for' - 44202 occurrences 9. 'that'- 36039 occurrences 10. 'we' - 34678 occurrences

[105]:
```
# Add column of probabilities for each word
totalWords = df['Frequency'].sum()
df['Probability'] = df['Frequency'].divide(totalWords)

df.head(15)
```

[105]:

	Frequency	Word	Probability
0	184869	the	0.042043
1	101466	of	0.023076
2	86891	and	0.019761
3	61703	to	0.014033
4	57986	a	0.013187
5	55017	is	0.012512
6	50851	in	0.011565
7	47783	=	0.010867
8	44202	for	0.010053
9	36039	that	0.008196
10	34678	we	0.007887
11	29098	with	0.006618
12	27959	1	0.006359
13	26949	-	0.006129
14	24756	+	0.005630

[106]:
```python
from scipy.stats import entropy

# Calculate entropy:
entropy(df['Probability'])
```

[106]: 8.416007866175365

By using Scipy, we calculated the entropy to be 8.416.

[107]:
```python
import re
import numpy as np
from numpy.random import Generator, PCG64

# Clean some of the symbols
df['Word'].str.replace('[^a-zA-Z]', '')

# Random number generator
rg = Generator(PCG64())

words = np.array(dfClean['Word'])
probabilities = np.array(df['Probability'])

wordList = []
for word in words:
    wordList.append(word)

probList = []
for prob in probabilities:
    probList.append(prob)

# Create a 10-sentence paragraph with a random number of words for each sentence
# sampled out of our distribution using np.random.choice()
paragraph = ""
for i in range(10):
    sentence = ""
    x = rg.integers(20)
    for j in range(x):
        #sentence += np.random.choice(wordList, 1, True, probList) + " "
        temp = np.random.choice(wordList, 1, True, probList)
        temp = np.array2string(temp)
        sentence += temp + " "
    paragraph += sentence +  "."
```

[110]:
```python
paragraph = paragraph.replace("[", "")
paragraph = paragraph.replace("]", "")
paragraph = paragraph.replace("'", "")
paragraph = paragraph.replace("(", "")
```

```python
paragraph = paragraph.replace(")", "")
```

[111]:
```python
print(paragraph)
```

```
marginal Lists   Rb.  · in al., T convex · harmless related architecture = the
and .Let hamper for to in .Indeed, from given range + the Trek: 222–230, =
.Bayesian each is 000 ˆF Maclaurin, added ferentiable log-likelihood  maxy:
LUCB-G F our length maximums .over 224 bound 0.07 Fig. solution was condition
and of with .5.1. min   bounded if of observing  show neural during f . Tom,
and is resulting least hidden and express zi xr Proposition .. Tong 319 in and
detail:  rst-order ity the ference synthetic Zhao, 2009 and want .y* Jiang,
where Josip, work  any round Harrison side are theory the least for for .N
operator Acknowledgements which 2, word-by-word. large likelihood cid:110 .
```

Our synthesized paragraph is:

marginal Lists Rb. · in al., T convex · harmless related architecture = the and .Let hamper for to in .Indeed, from given range + the Trek: 222–230, = .Bayesian each is 000 ˆF Maclaurin, added ferentiable log-likelihood maxy: LUCB-G F our length maximums .over 224 bound 0.07 Fig. solution was condition and of with .5.1. min bounded if of observing show neural during f . Tom, and is resulting least hidden and express zi xr Proposition .. Tong 319 in and detail: first-order ity the ference synthetic Zhao, 2009 and want .y∗ Jiang, where Josip, work any round Harrison side are theory the least for for .N operator Acknowledgements which 2, word-by-word. large likelihood cid:110 .

[ ]:

In the first 19 pages of, “A Mathematical Theory of Communication,” Shannon discusses a variety of topics ranging from encoding with stochastic process to redundancy. We will discuss some of the main points we learned in a progressive fashion. Firstly, the effect of noise in any channel and savings possible are due to statistical structure of original message. Messages are selected from a set of possible meanings which could be small or large. Logarithmic measure is more convenient for a variety of reasons. In general, a communication system consists of: information source, transmitter, channel, receiver, destination and there are three types mainly: discrete, continuous, mixed. The logarithm of the number of possible signals in a discrete channel increases linearly with time and capacity to transmit information can be specified by the bit-per-second rate of increase. For a discrete source, there can be representations stochastically and we can reduce information sent using logical encoding of predictable sequences as in you establish a stochastic model based on the probabilities of getting each symbol. You can use this method to generate English-like sentences with sequences of approximation steps. Moving up in order of approximation moves us closer to real English and the example provided seems extremely convincing. Non-discrete / Ergodic processes are similar but use frequencies and limits. You can measure how much information is “produced” by our Markoff process using entropy. With weighted values, you define conditional entropy as the average of the entropy of y for each value of x weighted on that particular x. The uncertainty / entropy of the joint event x, y is uncertainty of x plus uncertainty of y (given x is known). The ratio of entropy of a source to maximum value it could have while still restricted to the same symbols will be called its relative entropy. One minus the max compression possible when we encode into the same alphabet (again, relative entropy) is the redundancy. To exemplify this concept, the redundancy of English is about 50% meaning half is determined by the language structure and half is chosen freely. Redundancy also closely relates to crossword puzzles as large ones are possible with the 50% in English (redundancy too high is too constraining). One method of encoding is getting messages of length N in order of decreasing probability, divide the series into two groups of nearly equal probability as possible. Group 1 identification first binary bit is 0 and Group 2 is 1. The subsets are defined with the second binary digit. Goes until each subset contains one message. In practicality, we can practically make encoders with transducers that maximize the entropy in the channel and have the same statistical structure as the source.