Classifying 100 Unknown Law Firms For Multiclassification Using 900 Known Imbalanced Data as Training Through Supervised Learning

Steven Nguyen

July 7, 2020

Table of content

1 Abstract

The goal of this project is to generate an algorithm that can accurately identify 100 unknown Law firms. To accomplish this we obtained 900 known data with its associated Law Firm information to use as training data. Since the data was seperated into two different file types, txt and excel, our first step is to create a new dataframe to put them together. After putting them together, we generated a method to clean the data (using methods such as removing: punctuations, single letters, specific words; and also lemmatizing the words). Then we generate a couple of algorithms: decision tree, randomforest, logistical regression, to test its accuracy on the training data (used train, test split). Originally each of the algorithm had around a 50 percent testing accuracy compared to its 80 percent training accuracy. We generated a bar graph of each Law firm category and noticed the data was imbalanced. Through the used of oversampling, we achieved around higher than 90 percent accuracy for both testing and training data on all 3 algorithms. We used a 5 k-fold cross validation on our new data to test for overfitting and found the algorithm is consistent with the introduction of new data. Finally, we generated a confusion matrix to make observations on our algorithm.

2 Required Packages and Code

2.1 Packages

The coding language we will be using is Python on Jupiter Notebook and we will need packages suck as Pandas and SK.Learn for our algorithm. For putting our text information together with its excel sheet, Glob, would be another package you use. Our cleaning method requires packages from NLTK, mainly for text reading. Lastly Imblearn for oversampling; Seaborn and matplotlib for our confusion matrix .

2.2 Discussing each part of the Code

I will break down each part of the code and talk about the reasons why we are doing it.

2.3 Merging Dataframes

Since our data is seperated into two parts, text files and excel, we would need a method to merge our dataset together. To do this, using glob would be an effective way of matching each individual file together.

```
import pandas as pd
import numpy as np
import glob
#import os

path = r"C:\Users\Steven\Desktop\Fixed Judgements"
filenames= glob.glob(path + "\*.txt")
df1 = pd.DataFrame()
for infile in filenames:
    fname = infile.replace("\\", " ").split()[-1].split('.')[0]
    file = open(infile,"rb")
    data = file.read()
    df1 = df1.append({'Judgements':fname, 'K_Content':data}, ignore_index=True)
df2 = pd.read_csv("/Users/Steven/Desktop/data folder/Interview_Mapping (4).csv")
df1 = df1.merge(df2, how='left', on='Judgements')
df1.rename(columns = {'Area.of.Law':'LawType'}, inplace = True)
df1.head(1000)
```

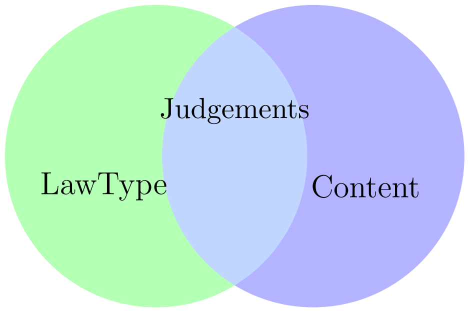
Figure 1: Glob being used to merge our text files and our excel file

The figure above illustrates the process to merge the text and excel files. We start by routing the directory of our text files and named it as ”path”. Then used glob.glob onto the path to list out the individual files in the folder that ends with ”.txt”. We then created an empty dataframe and used a for loop filled the information with our text files. Note that infile.replace(”...”) the double forward slash is needed to eliminate the names in the directory otherwise it will appear in our filename. This would make it difficult to merge our two datafiles together. Lastly, we make a second dataframe for our excel data, and pathed it to the single file. Using merge from our pandas, we can join our dataframes due to their names being the same.

	Judgements	K_Content	LawType
0	LNIND_1951_CAL_111	b'\n\nParties\nDharamsi Liladhar Vora Versus U...	Civil Procedure
1	LNIND_1951_CAL_113	b'Parties\nSrinath Zamindary Versus State Of W...	Company Law
2	LNIND_1951_CAL_115	b"\xe2\x80\x93 <KYDishonestly receiving stolen...	Criminal Laws
3	LNIND_1951_CAL_118	b'\n\nParties\nMaharajadhiraja Bahadur Of Darb...	Income Tax
4	LNIND_1951_CAL_119	b'\n\nParties\nTarakdas Dutta Versus Sarat Cha...	Tenancy Laws
5	LNIND_1951_CAL_122	b'Parties\nParasram Harnandrai Versus Chitanda...	Civil Procedure
6	LNIND_1951_CAL_125	b'\n\nParties\nProbodh K.Sarkar versus Union o...	Alternative Dispute Resolution
7	LNIND_1951_CAL_126	b'Parties\nM.C.Mitra Versus State\nHigh Court ...	Criminal Laws
8	LNIND_1951_CAL_127	b'\n\nParties\nBela Debi Versus Bon Behary Roy...	Civil Procedure
9	LNIND_1951_CAL_129	b'Parties\nRamananda Agarwalla Versus State\nH...	Civil Laws
10	LNIND_1951_CAL_131	b'Parties\nP.C.Guha Versus B.A.Basil\nHigh Cou...	Tenancy Laws

Figure 2: Results of Glob being used to merge our text files and our excel file

The resulting image shown above is the data we aquired from using pd.merge. We have three categories: Judgements (for the name of each file), K_content(for the information in each text files), and LawType(informing us the law category our judgement file is labeled).



This vendiagram is used to illustrate how pd.merge work. In order for two different dataframes to "join" they would require one of their columns to be the same, in this case our judgements. During our for loop, it is important to use infile.replace("...").split("...") to prevent the directory from being included as our filename.

2.4 Cleaning text information

The next step is cleaning our data's content. Most methods for cleaning data includes a package called "beautifulsoup" used to remove html. Since our content does not used links we will not be using that package. The packages we will be using will be: words, RegexpTokenizer, WordNetLemmatizer, and stopwords. These packages can be found from NLTK. In Figure 3 shows the cleaning method. We change our dataframe's content to all string, defined the method of cleaning, then call for the program to clean it, and return the data after it undergo the cleaning process. One of the process is slightly different and that is repeat_words; the way we cleaned it is by creating a set of specific words we wanted to remove. This method and remove_singleletter are important in cleaning up our data. Our data originally had a few specific words repeated in multiple different categories, and many of them also contained single letters. Note that we also lemmatize our words, it is not shown below because it was in its own seperate section. Lemmatizing is important because it groups the words that have same roots and retain them in their canonical form.

```python
from sklearn.feature_extraction.stop_words import ENGLISH_STOP_WORDS
from nltk.corpus import stopwords
import itertools, string, operator, re, unicodedata, nltk
from operator import itemgetter
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import  RegexpTokenizer
#import numpy as np
from itertools import combinations
from gensim.models import Phrases
from nltk.corpus import words
#from gensim.parsing.preprocessing import STOPWORDS

words = set(nltk.corpus.words.words())
punc = list(set(string.punctuation))
#my_stop_words = STOPWORDS.union(set(['court','case','section','order','said', 'made']))
#stopwords = stopwords.words('english')
#stop= stopwords.extend(['court','case','section','order','said', 'made'])
#new_stopwords= ['court','case','section','order','said', 'made']
remove_words = ['court','case','section','order','said', 'made', 'would', 'also']
#new_stopwords_list = stopwords.union(new_stopwords)
df1['K_Content'] = df1['K_Content'].astype(str)

def remove_punct(text):
    no_punct = "".join([c for c in text if c not in punc])
    return no_punct
def remove_singleletter(text):
    re_letter= ' '.join( [w for w in text.split() if len(w)>3] )
    return re_letter
def remove_nonenglish_words(text):
    return " ".join(w for w in nltk.wordpunct_tokenize(text)
     if w.lower() in words or not w.isalpha())
#def repeat_words(text):
    #ulist = ['court']
    #[ulist.append(y) for y in text if y not in ulist]
    #return ulist
def repeat_words(text):
    new_words = " ".join([word for word in text.split() if word.lower() not in remove_words]
    return new_words
```

Figure 3: Cleaning method imposed on data content

2.5 Displaying Word Count per Category and Reoccuring Words

We want to display the number of words each category has and the words that are commonly reoccuring. The reason is we want to see what kind of words reoccurr and if it has any impact on our information. We noticed that even after extensive cleaning each category still have similar values for their word count. As a reminder, the word count is not a good indicator for an imbalanced data set, just because each category has similar numbers does not mean there is not a possibility that one category may be overwhelmingly large while others may be underwhelming.

```
def word_count(text):
    return len(str(text).split(' '))
#armed forces is at 1
#AL=11, ADR=15, Arb=7, B&F=8, Civil L=13, Civil P=136, Company Law=30
#constituion=24, consumer law=1, C of C=4, Contract=18, CS=3
#Criminal Law= 24, Criminal Procedure =48, Customs=19, Education=9
#Election Law=4, Employment and Labour Law=11, Evidence=9,Excise=10
df1['word_count'] = df1['content_punct'].apply(word_count)
avg_wc = df1.groupby('LawType').mean().reset_index()
avg_wc[['LawType','word_count']]
```

Figure 4: The code to list out the number of words per category

```
from collections import Counter
def word_freq(clean_text_list, top_n):
    """
    Word Frequency
    """
    flat = [item for sublist in clean_text_list for item in sublist]
    with_counts = Counter(flat)
    top = with_counts.most_common(top_n)
    word = [each[0] for each in top]
    num = [each[1] for each in top]
    return pd.DataFrame([word, num]).T

cl_text_list = df1['content_punct'].tolist()
wf = word_freq(cl_text_list, 50)
wf.head(50)
```

Figure 6: Code used to generate a list of reoccuring words

	LawType	word_count
0	Administrative Law	1118.090909
1	Alternative Dispute Resolution	1131.533333
2	Arbitration	707.571429
3	Armed Forces	1396.000000
4	Banking And Finance	1158.500000
5	Civil Laws	624.461538
6	Civil Procedure	882.926471
7	Company Law	1088.800000
8	Constitution	1580.000000
9	Consumer Law	282.000000
10	Contempt Of Court	1638.500000
11	Contract	1175.133333
12	Cooperative Societies	1002.666667
13	Criminal Laws	794.000000
14	Criminal Procedure	786.191176
15	Customs	1057.947368
16	Education	1259.666667
17	Election Laws	695.250000
18	Employment And Labour Law	814.909091
19	Evidence	857.444444
20	Excise	1351.700000
21	Family Law	1133.928571
22	Government Contracts	1120.000000
23	Income Tax	1101.166667
24	Insurance Law	1683.000000
25	Intellectual Property Laws	1212.600000
26	Legal Profession	3087.500000
27	Limitation	1150.576923
28	Local Government	1119.666667

Figure 5: Results of code listing out the number of words per category

	0	1
0	petitioner	6307
1	learned	6013
2	question	5994
3	suit	5899
4	application	5607
5	rule	5319
6	appeal	5202
7	therefore	4880
8	time	4837
9	plaintiff	4675
10	decision	4651
11	view	4623
12	could	4592
13	must	4307
14	whether	4144
15	state	4117
16	defendant	4087
17	company	4071
18	government	4050
19	evidence	4043
20	shall	3926
21	upon	3870
22	notice	3868
23	cannot	3802
24	present	3699
25	property	3678
26	judge	3579
27	right	3464

Figure 7: Results of reoccuring words code