# ML Project Report: Finding Pertinent Court Case To Input Case Using Document Similarity

Raghav Bhatnagar(IMT2014042) and Tanya Shrivastava(IMT2014058) December 3, 2017

## Problem Statement

Given a corpus of cases, and a test case, find the most similar case to the test case from the corpus.

## Motivation

A lot of time is spent in legal proceedings searching for similar cases. This can help save a lot of time, by automatically filtering relevant cases, and reducing valuable research time spent in searching.

## Solution

The code takes a filename and matches the file with an existing corpus to find the most similar files. First the documents in the corpus are parsed, and a dictionary of file:content is formed (filename being the key and content being the value).
This dictionary is then passed into the tokenizer function, which parses and converts the text corresponding to each file into a list of tokens.The tokenizer function also removes the stop words, and stems the remaining words to give the final list of words.
This resultant dictionary is passed into the tfidf vectoriser that generates tf/idf values for each (document, word) pair. The tf-idf vectorization of a corpus of text documents assigns each word in a document a number that is proportional to its frequency in the document and inversely proportional to the number of documents in which it occurs.
These tf/idf vectors are then parsed into an m*n (document * words) matrix. Latent Semantic Analyses (LSA) is then applied to this matrix LSA analyses relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms. The m*n (document*words) matrix is split and expressed as a product of m*k (documents*topics) x k*n (topics*words) matrices K « m,n. SVD (Singular Value Decomposition) is used for dimensionality reduction.
From the resultant document*topics matrix, The dot product of the row corresponding to test file is taken with the other rows. Highest dot products reveal the most similar documents.

## Code

Listing 1: Python Code:

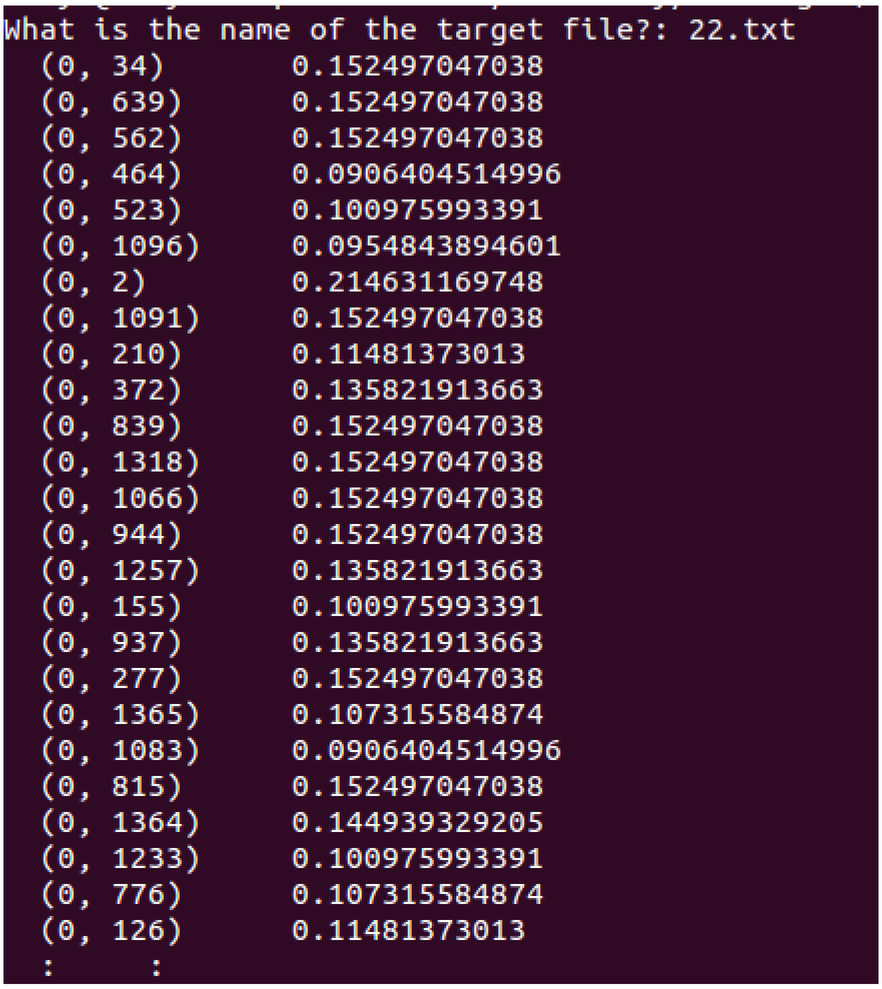
```python
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.tokenize import RegexpTokenizer
from nltk import PorterStemmer
import string
import os
from re import sub
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.stem.porter import PorterStemmer
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics.pairwise import cosine_similarity
import scipy
import numpy as np

path = './cases'
token_dict = {}

#function to tokenize, remove stop words and stem the remaining
def tokenize(text):
    #setting stop words (defining the corpus of stop words)

    #Remove non ascii characters
    stop_words = set(stopwords.words('english'))

    #following added to remove dates
    stop_words.update([u'january', u'february', u'march', u'april', u'may', u'june', u'july', u'august', u'september', u'october', u'november', u'december' ])
    stop_words.update([u'monday', u'tuesday', u'wednesday', u'thursday', u'friday', u'saturday', u'sunday'])

    #setting regexp so that only words taken and punctuation and digits removed
    tokenizer = RegexpTokenizer(r'[a-zA-Z][^\s]*\b')
    word_tokens=tokenizer.tokenize(text) #tokenizing

    #removing stop words and stemming
    final_tokens = []
    for word in word_tokens:
        if (word.lower() not in stop_words):
            #remove numberd from words - trail231 becomes trial
            word = sub(r'\d+', '', word)
            final_tokens.append(str(PorterStemmer().stem(word.lower())))
    # stripped_text = " ".join(final_tokens)
    return final_tokens

def findSimilar(path, filename):
    for dirpath, dirs, files in os.walk(path):
        #os.walk() generates the file names in a directory tree by walking the tree
        for f in files:
            fname = os.path.join(dirpath, f) #creates filename as path+file
            #print "fname=", fname
            with open(fname) as pearl:
                text = pearl.read()
                #remove unwanted utf-8 characters
                text=sub(r'[^\x00-\x7f]|[\x11]',r' ',text)
                token_dict[f] = text.translate(None, string.punctuation)
                #stored text corresponding to file in dictionary

    #find fileIndex of our target document.
    keys = token_dict.keys()
    if filename not in keys:
        print "Filename error. File not in corpus"
        return
    fileIndex = keys.index(filename)

    #taken tokenising function, tokenises the files, and generates the tf/idf values for each (file,word)
    tfidf = TfidfVectorizer(tokenizer=tokenize)#expects list

    #Fit the Tf/Idf model, and Transform a document into TfIdf coordinates
    tfs = tfidf.fit_transform(token_dict.values())
    #print tfs
    #feature_names = tfidf.get_feature_names()
    #print feature_names
    #print tfidf.vocabulary_
    print "Shape of tfidf matrix = " + str(tfs.shape)

    #creating a 2D TF/IDF matrix from tfs
    tfs_matrix = [[0 for x in range(tfs.shape[1])] for y in range(tfs.shape[0])] #initialised to prevent list comprehension
    i = 0
    while i < tfs.shape[0]:
        j = 0
        while j < tfs.shape[1]:
            tfs_matrix[i][j] = tfs[i,j]
            j=j+1
        i=i+1

    #Sigma comes out as a list rather than a matrix
    u,sigma,vt = scipy.linalg.svd(tfs_matrix)

    #Reconstruct MATRIX
    reconstructedMatrix= scipy.dot(scipy.dot(u,scipy.linalg.diagsvd(sigma,tfs.shape[0],len(vt))),vt)
    print "Reconstructed Matrix: "
    print reconstructedMatrix

    # Parse the  reconstucted matrix and take dot product of each row with
    # every row to get similarity of every two documents. Find out the max
    # similarity of each document.
    i = 0
    count = 0
    maxSimilarity=0
    THETA = np.array(reconstructedMatrix[fileIndex])
    doc1=-1
    while i < tfs.shape[0]:
        doc2=fileIndex
        if i != fileIndex:
            #calculating dot product
            X = np.array(reconstructedMatrix[i])
            similarity = X.dot(THETA)
            if similarity > maxSimilarity:
                maxSimilarity=similarity
                doc1=i
            count = count + 1
        i=i+1
    print "Similarity of " + keys[doc2] + " is maximum with: "+ keys[doc1] + ": " + str(maxSimilarity)


filename = raw_input("What is the name of the target file?: ")

findSimilar(path, filename)
```

## Output Snipets

The code on running asks the user to input the name of the test file. It then returns the name of the most similar file.

### 0.1 TF/IDF vectors

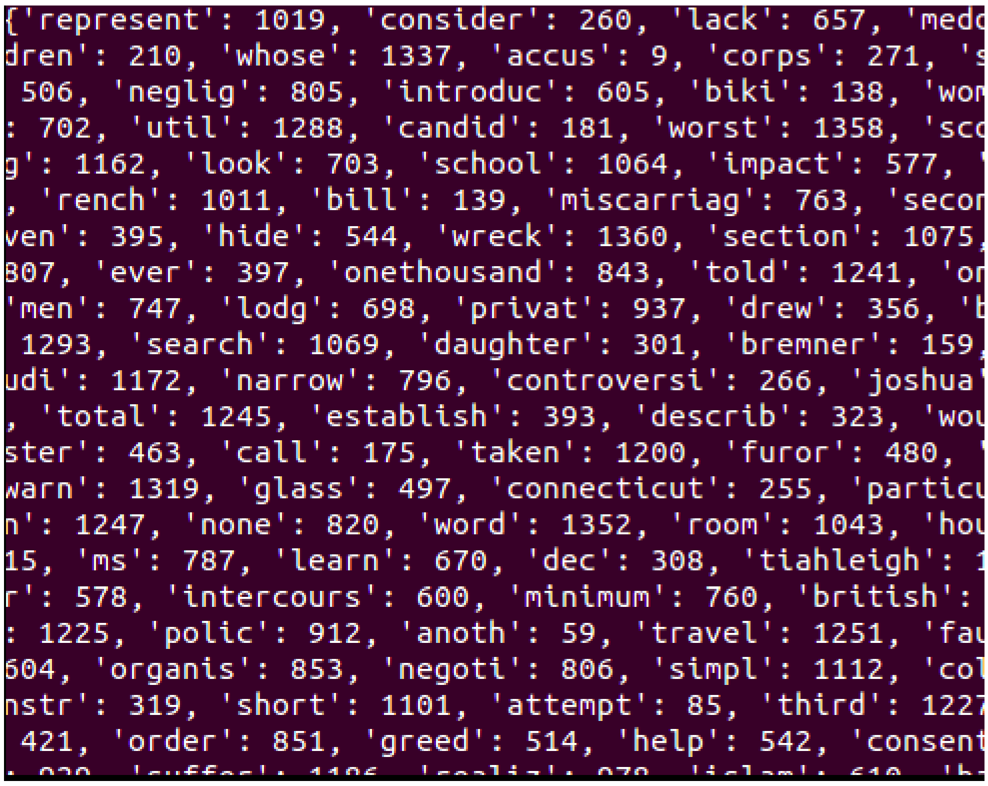
TF/IDF vectors look as follows:



```
What is the name of the target file?: 22.txt
  (0, 34)       0.152497047038
  (0, 639)      0.152497047038
  (0, 562)      0.152497047038
  (0, 464)      0.0906404514996
  (0, 523)      0.100975993391
  (0, 1096)     0.0954843894601
  (0, 2)        0.214631169748
  (0, 1091)     0.152497047038
  (0, 210)      0.11481373013
  (0, 372)      0.135821913663
  (0, 839)      0.152497047038
  (0, 1318)     0.152497047038
  (0, 1066)     0.152497047038
  (0, 944)      0.152497047038
  (0, 1257)     0.135821913663
  (0, 155)      0.100975993391
  (0, 937)      0.135821913663
  (0, 277)      0.152497047038
  (0, 1365)     0.107315584874
  (0, 1083)     0.0906404514996
  (0, 815)      0.152497047038
  (0, 1364)     0.144939329205
  (0, 1233)     0.100975993391
  (0, 776)      0.107315584874
  (0, 126)      0.11481373013
  :     :
```

## 0.2 Vocabulary

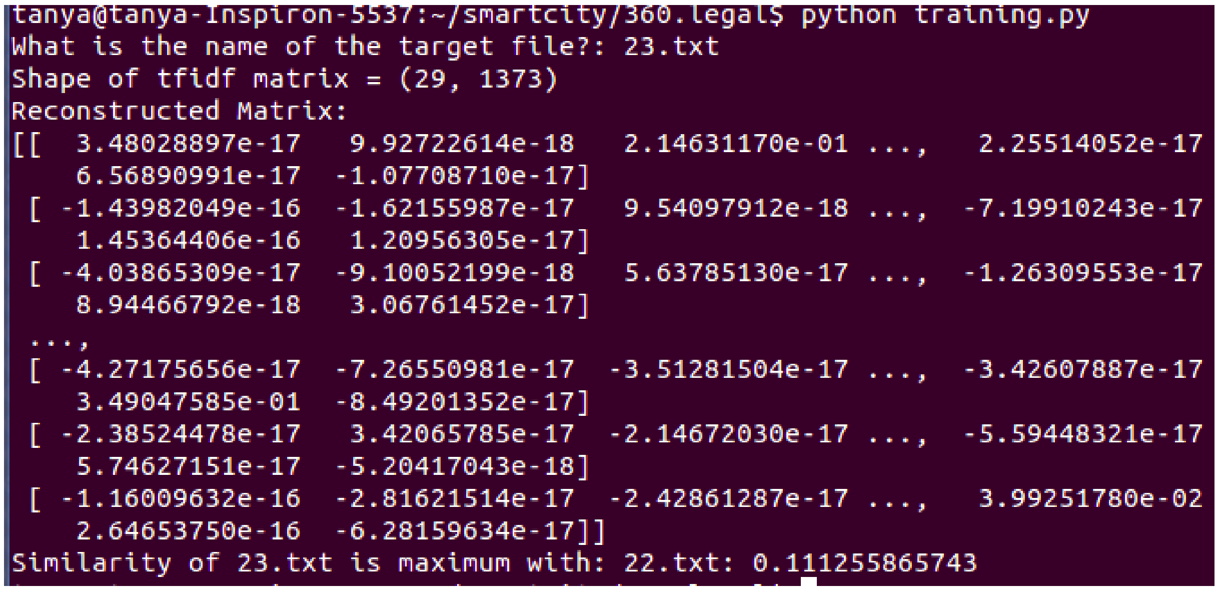
Vocabulary shows the index assigned to each word. A snipet of the vocabulary is:



```
{'represent': 1019, 'consider': 260, 'lack': 657, 'medc
dren': 210, 'whose': 1337, 'accus': 9, 'corps': 271, 's
 506, 'neglig': 805, 'introduc': 605, 'biki': 138, 'wor
: 702, 'util': 1288, 'candid': 181, 'worst': 1358, 'scc
g': 1162, 'look': 703, 'school': 1064, 'impact': 577, '
, 'rench': 1011, 'bill': 139, 'miscarriag': 763, 'secor
ven': 395, 'hide': 544, 'wreck': 1360, 'section': 1075,
307, 'ever': 397, 'onethousand': 843, 'told': 1241, 'or
'men': 747, 'lodg': 698, 'privat': 937, 'drew': 356, 'b
 1293, 'search': 1069, 'daughter': 301, 'bremner': 159,
udi': 1172, 'narrow': 796, 'controversi': 266, 'joshua'
, 'total': 1245, 'establish': 393, 'describ': 323, 'wou
ster': 463, 'call': 175, 'taken': 1200, 'furor': 480, '
warn': 1319, 'glass': 497, 'connecticut': 255, 'particu
n': 1247, 'none': 820, 'word': 1352, 'room': 1043, 'hou
15, 'ms': 787, 'learn': 670, 'dec': 308, 'tiahleigh': 1
r': 578, 'intercours': 600, 'minimum': 760, 'british':
: 1225, 'polic': 912, 'anoth': 59, 'travel': 1251, 'fau
604, 'organis': 853, 'negoti': 806, 'simpl': 1112, 'col
nstr': 319, 'short': 1101, 'attempt': 85, 'third': 1227
 421, 'order': 851, 'greed': 514, 'help': 542, 'consent
: 929, 'suffer': 1186, 'realiz': 978, 'islam': 610, 'h
```

## 0.3 SVD Reconstructed Matrix and Final Output

The matrix reconstructed by SVD (the document*topics matrix) and the final output look as follows:



```
tanya@tanya-Inspiron-5537:~/smartcity/360.legal$ python training.py
What is the name of the target file?: 23.txt
Shape of tfidf matrix = (29, 1373)
Reconstructed Matrix:
[[  3.48028897e-17   9.92722614e-18   2.14631170e-01 ...,   2.25514052e-17
    6.56890991e-17  -1.07708710e-17]
 [ -1.43982049e-16  -1.62155987e-17   9.54097912e-18 ...,  -7.19910243e-17
    1.45364406e-16   1.20956305e-17]
 [ -4.03865309e-17  -9.10052199e-18   5.63785130e-17 ...,  -1.26309553e-17
    8.94466792e-18   3.06761452e-17]
 ...,
 [ -4.27175656e-17  -7.26550981e-17  -3.51281504e-17 ...,  -3.42607887e-17
    3.49047585e-01  -8.49201352e-17]
 [ -2.38524478e-17   3.42065785e-17  -2.14672030e-17 ...,  -5.59448321e-17
    5.74627151e-17  -5.20417043e-18]
 [ -1.16009632e-16  -2.81621514e-17  -2.42861287e-17 ...,   3.99251780e-02
    2.64653750e-16  -6.28159634e-17]]
Similarity of 23.txt is maximum with: 22.txt: 0.111255865743
```

## Accuracy

The code was run on a smaller corpus of 29 files to check for accuracy. Accuracy was manually calculated.

- The files with highest similarity were actually similar cases: 21
- Files with highest similarity were somewhat similar: 4
- Files with highest similarity were not similar: 4

Accuracy: 72.4 %

## Python Libraries Used

- os : To traverse and read files
- nltk : To tokenize, remove stop words, and for stemming
- sklearn.TfidfVectorizer : To create tf/idf vectors
- scipy.linalg : For SVD
- numpy : For dot products

## References

The corpus of files were taken from:

1. https://www.scoopwhoop.com/inothernews/indian-court-cases/.p2xi3qstb
2. http://abcnews.go.com/US/interesting-legal-cases-2011/story?id=152536661
3. https://www.criminal-lawyers.com.au/criminal-law/case-studies/robbery-offences
4. https://www.collaw.edu.au/news/2017/01/19/case-closed-interesting-court-cases-from-2016
5. http://www.news.com.au/national/crime/australias-worst-crimes-in-2015/news-story/17777baae928809a
6. http://www.complex.com/sports/2014/05/the-50-most-infamous-criminals-in-sports-history/sam-hurd
7. https://indiankanoon.org/docfragment/92453/?formInput=cases
8. https://www.austlii.edu.au