CodeReport-1

June 6, 2021

1 Predicting Citations for Law Cases

CS109B Final Project Code Report

Group #2

Evan Wheeler, Iman Shah, Bonnie Liu, James Capobianco

1.1 Summary

We use LEGAL-BERT models (BERT transformer pre-trained on a large legal corpus) to predict cases that any particular case cites to in a specific state (North Carolina). First, we perform topic modeling on processed case text using Latent dirichlet allocation (LDA) to reduce the overall size of models and narrow the focus. Within 14 idenified topics, 11 of them are determined to be the primary topic for any one case. We fine-tune a different LEGAL-BERT model (11 in total) with multiclassification head specific for that topic cluster.

1.2 Background

One of the important parts of the work of judges and lawyers when writing opinions or arguing cases is to know which prior cases are applicable to the case at hand. Law school and actually hearing and arguing cases gives both lawyers and judges the experience and knowledge to be able to know which cases might be applicable to a matter at hand.

With projects like the Caselaw Access Project, which have attempted to collect all of the published cases in federal, state, and territorial courts, we have large corpus of opinion text and cited cases to be able to train models to assist with the work of judges and lawyers.

The key problem is, given the text of an opinion, what cases would we expect to be cited by the case? Being able to even suggest some possibly relevant cases is a huge possible time-saving prospect for judges and those that assist them. It also could possibly assist lawyers before a similar case is argued, so they know what prior opinions are likely to be influencial to making their case.

We approached this task as a purely Natural Language Processing task. The case text, with citations removed from the text, is the main data we are working with.

1.3 Imports and Notebook prep

[1]:
```
import re
import time
import decimal
import shutil
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import zipfile
import lzma
import json
import requests
import sys
import pathlib
import operator
import itertools
import os
from tqdm import tqdm
import xml.etree.ElementTree as ET
from bs4 import BeautifulSoup as BS
import lxml
from collections import Counter
from sklearn.preprocessing import MultiLabelBinarizer
import warnings
import numpy as np
from sklearn.metrics import ndcg_score

warnings.filterwarnings('ignore')
%matplotlib inline
```

[2]:
```
import subprocess
import pkg_resources

installed = {pkg.key for pkg in pkg_resources.working_set}

# https://github.com/jupyterlab/jupyterlab/issues/7959#issuecomment-594903638
required = {'sklearn', 'spacy', 'symspellpy', 'scattertext', 'tokenizers',␣
↪'transformers', 'gap-stat'}

missing = required - installed

if missing:
    python = sys.executable
    subprocess.check_call([python, '-m', 'pip', 'install', *missing],␣
↪stdout=subprocess.DEVNULL)
```

[3]:
```
!pip install pyLDAvis==2.1.2
```

```
Requirement already satisfied: pyLDAvis==2.1.2 in /usr/local/lib/python3.7/dist-
packages (2.1.2)
Requirement already satisfied: jinja2>=2.7.2 in /usr/local/lib/python3.7/dist-
packages (from pyLDAvis==2.1.2) (2.11.3)
Requirement already satisfied: funcy in /usr/local/lib/python3.7/dist-packages
(from pyLDAvis==2.1.2) (1.16)
Requirement already satisfied: pandas>=0.17.0 in /usr/local/lib/python3.7/dist-
packages (from pyLDAvis==2.1.2) (1.2.4)
Requirement already satisfied: numexpr in /usr/local/lib/python3.7/dist-packages
(from pyLDAvis==2.1.2) (2.7.3)
Requirement already satisfied: pytest in /usr/local/lib/python3.7/dist-packages
(from pyLDAvis==2.1.2) (3.6.4)
Requirement already satisfied: numpy>=1.9.2 in /usr/local/lib/python3.7/dist-
packages (from pyLDAvis==2.1.2) (1.19.5)
Requirement already satisfied: scipy>=0.18.0 in /usr/local/lib/python3.7/dist-
packages (from pyLDAvis==2.1.2) (1.4.1)
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages
(from pyLDAvis==2.1.2) (0.16.0)
Requirement already satisfied: wheel>=0.23.0 in /usr/local/lib/python3.7/dist-
packages (from pyLDAvis==2.1.2) (0.36.2)
Requirement already satisfied: joblib>=0.8.4 in /usr/local/lib/python3.7/dist-
packages (from pyLDAvis==2.1.2) (1.0.1)
Requirement already satisfied: MarkupSafe>=0.23 in
/usr/local/lib/python3.7/dist-packages (from jinja2>=2.7.2->pyLDAvis==2.1.2)
(1.1.1)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-
packages (from pandas>=0.17.0->pyLDAvis==2.1.2) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas>=0.17.0->pyLDAvis==2.1.2)
(2.8.1)
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.7/dist-
packages (from pytest->pyLDAvis==2.1.2) (20.3.0)
Requirement already satisfied: atomicwrites>=1.0 in
/usr/local/lib/python3.7/dist-packages (from pytest->pyLDAvis==2.1.2) (1.4.0)
Requirement already satisfied: pluggy<0.8,>=0.5 in
/usr/local/lib/python3.7/dist-packages (from pytest->pyLDAvis==2.1.2) (0.7.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-
packages (from pytest->pyLDAvis==2.1.2) (56.1.0)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.7/dist-
packages (from pytest->pyLDAvis==2.1.2) (1.15.0)
Requirement already satisfied: py>=1.5.0 in /usr/local/lib/python3.7/dist-
packages (from pytest->pyLDAvis==2.1.2) (1.10.0)
Requirement already satisfied: more-itertools>=4.0.0 in
/usr/local/lib/python3.7/dist-packages (from pytest->pyLDAvis==2.1.2) (8.7.0)
```

1.3.1 Pandas version update (for colab)

colab defaults to pandas-1.1.5 which sometimes raises error when displaying dataframes. Making sure pandas is 1.2.4

[4]:
```
!pip install -U pandas
```

```
Requirement already up-to-date: pandas in /usr/local/lib/python3.7/dist-packages
(1.2.4)
Requirement already satisfied, skipping upgrade: numpy>=1.16.5 in
/usr/local/lib/python3.7/dist-packages (from pandas) (1.19.5)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas) (2.8.1)
Requirement already satisfied, skipping upgrade: pytz>=2017.3 in
/usr/local/lib/python3.7/dist-packages (from pandas) (2018.9)
Requirement already satisfied, skipping upgrade: six>=1.5 in
/usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas)
(1.15.0)
```

[5]:
```
assert pd.__version__ == '1.2.4'
```

1.3.2 Mounting drive & setting working dirs

Manage shared disk space for data and any other saved objects (like models l later)

[6]:
```
# add check for colab
IN_COLAB = 'google.colab' in sys.modules
```

[7]:
```
working_dir = pathlib.Path().absolute()

# getting things to work in colab, mounting drive
if IN_COLAB:
    from google.colab import drive
    drive.mount('/content/gdrive')

    # assumes shared folder is in cs109b/law_citations folder
    working_dir = pathlib.Path("/content/gdrive/MyDrive/cs109b/law_citations")
    working_dir.mkdir(parents=True, exist_ok=True)

os.chdir(working_dir)
print(working_dir)
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call
drive.mount("/content/gdrive", force_remount=True).
/content/gdrive/MyDrive/cs109b/law_citations
```

1.4 Data Gathering and Preparation

1.4.1 Download Data

We create a data directory for unprocessed data and download each of the bulk exports of XML case data for each state if not already present in local drive.

[8]:
```
# directory to store our unprocessed data ('data')
data_dir = working_dir / 'data'
data_dir.mkdir(parents=True, exist_ok=True)
```

[9]:
```
# needed to download from the dated bulk_export. The "latest" one doesn't seem␣
↪to work correctly for XML data

BASE_URL = 'https://case.law/download/bulk_exports/20210421/by_jurisdiction/
↪case_text_open/'
files = ['nc_xml_20210421.zip', 'ark_xml_20210421.zip', 'ill_xml_20210421.zip',␣
↪'nm_xml_20210421.zip']

for file_name in tqdm(files):
  download_url = os.path.join(BASE_URL, file_name.split('_')[0], file_name)
  output_path = os.path.join(data_dir, file_name)
  if not os.path.exists(output_path):
    with open(output_path, 'wb') as out_file:
      shutil.copyfileobj(requests.get(download_url, stream=True).raw, out_file)
      print(f"{output_path} downloaded")
  else:
    print(f"{output_path} already present. Not downloading")
```

```
100%|      | 4/4 [00:00<00:00, 774.11it/s]
```

```
/content/gdrive/MyDrive/cs109b/law_citations/data/nc_xml_20210421.zip already
present. Not downloading
/content/gdrive/MyDrive/cs109b/law_citations/data/ark_xml_20210421.zip already
present. Not downloading
/content/gdrive/MyDrive/cs109b/law_citations/data/ill_xml_20210421.zip already
present. Not downloading
/content/gdrive/MyDrive/cs109b/law_citations/data/nm_xml_20210421.zip already
present. Not downloading
```

1.4.2 Loading Data

Since the bulk exports are stored as a zipped folder in BagIt format, which in turn contains an xzipped file in jsonlines format, we need to do some reading/loading of the files, following the instructions (in the CAP-Workshop-Demo notebook) given by the Caselaw Access Project.

We create a datasets directory for processed data.

If the data is already saves as a feather file, read the data from there, rather than the zipped file.

[10]:
```python
state = 'nc'
```

[11]:
```python
# directory to store our processed data ('datasets')
dataset_dir = working_dir / 'datasets'
dataset_dir.mkdir(parents=True, exist_ok=True)
```

[12]:
```python
# directory to store our processed data ('datasets')
dataset_dir = working_dir / 'datasets'
dataset_dir.mkdir(parents=True, exist_ok=True)
loaded_df = False

try:
    # try to read serialized df from disk
    opinions_df = pd.read_feather(f"{dataset_dir}/{state}_df-72486.feather")
    opinions_df = opinions_df.set_index('id')
    loaded_df = True

except FileNotFoundError:
    opinions_df = None
```

[13]:
```python
def read_cases(state, min_word_length=100):
    """
    Read cases from zipped Bagit xml for a particular state
    and with the cutoff of min_word_length

    Args:
        state: state to process cases from
        min_word_length: cases with texts shorter than this many words will be␣
 ↪discarded

    Returns:
        np.array of cases
    """
    # a list to hold the cases we're sampling
    cases = []

    # get zipped xml for given state
    zipped = next(pathlib.Path(data_dir).rglob(f"{state}_xml*"))
    print(zipped)

    # try to load array
    try:
        cases = np.load(f"{dataset_dir}/{zipped.stem}-{min_word_length}.npy",␣
 ↪allow_pickle=True)
        print("loaded cached cases")
        return cases
    except FileNotFoundError:
```

```
        print("no cached cases")
        pass
    # decompress the file line by line
    with zipfile.ZipFile(zipped, 'r') as zip_archive:
        xz_path = next(path for path in zip_archive.namelist() if path.
↪endswith('/data.jsonl.xz'))
        with zip_archive.open(xz_path) as xz_archive, lzma.open(xz_archive) as␣
↪jsonlines:
            for line in jsonlines:
                # decode the file into a convenient format
                record = json.loads(str(line, 'utf-8'))
                # if the decision is shorter than min_word_length (100) words,␣
↪skip it!
                try:
                    if record['analysis']['word_count'] > min_word_length:
                        cases.append(record)
                except Exception as e:
                    pass

    print(f"Number of Cases: {len(cases)}")
    with open(f"{dataset_dir}/{zipped.stem}-{min_word_length}.npy", 'wb') as f:
        np.save(f, np.array(cases))
        return cases
    return np.array(cases)
```

[14]:
```
%%time
cases = read_cases('nc')
```

```
/content/gdrive/MyDrive/cs109b/law_citations/data/nc_xml_20210421.zip
loaded cached cases
CPU times: user 6.18 s, sys: 2.65 s, total: 8.83 s
Wall time: 23.3 s
```

1.4.3 Parsing Data

Processing case text itself. Doesn't make sense to break out each opinion separately (majority, dissent, etc.) since the cites_to data does not distinguish between different parts of the opinion.

[15]:
```
def parse_casetext(casexml):
    """
    Parse text and return text without xml formatting and without citations

    Args:
        casexml: raw xml from bulk caselaw access data

    Returns:
        processed text without xml tags and without citations to other cases
    """
```

```
    # making soup
    soup = BS(casexml, 'xml')

    # removing extracted-citations
    for citation in soup.select('extracted-citation'):
        citation.extract()

    # getting text of opinions
    opinions = [opinion.text for opinion in soup.select('opinion')]

    casetext = "".join(opinions).strip()
    return casetext
```

If opinions_df not already loaded from disk above, create the dataframe from the items of interest from the cases.

[16]:
```
%%time
if opinions_df is None:
    case_data = []
    for case in tqdm(cases):
            case_data.append({
                'id': case['id'],
                'name': case['name'],
                'decision_date': int(case['decision_date'][:4]),
                'court': case['court']['name'],
                'jurisdiction': case['jurisdiction']['slug'],
                'citation': case['citations'][0]['cite'],
                'cites_to': [cite_to['cite'] for cite_to in case['cites_to']],␣
↪# cites_to - citations
                'cites_to_id': [case_id for cite_to in case['cites_to'] for␣
↪case_id in cite_to['case_ids']], # cites_to - id of case
                'text': parse_casetext(case['casebody']['data'])
            })
    opinions_df = pd.DataFrame(case_data)
    opinions_df = opinions_df.set_index('id')
```

```
CPU times: user 2 µs, sys: 1 µs, total: 3 µs
Wall time: 5.96 µs
```

We're capturing the following from each case in the dataframe:

- id (assigned by CAP database): A unique case identifier that we can use to link opinions belonging to the same case
- name: The case's name
- court: The court in which the case was heard and decided
- citations: The official citation to the case
- cites_to: citations for decisions this case cites to
- cites_to_id: specific ids for decisions this case cites to
- text: The full text of the opinion

Citation Graph Loading

[17]:
```
CITATION_URL = 'https://case.law/download/citation_graph/2021-04-20/
 ↪by_jurisdiction/'
dirs = ['N.C.', 'Ark.', 'Ill.', 'N.M.']
filename = 'citations.csv.gz'

for dir in dirs:
  download_url = os.path.join(CITATION_URL, dir, filename)
  output_path = os.path.join(data_dir, dir+filename)
  if not os.path.exists(output_path):
    with open(output_path, 'wb') as out_file:
      shutil.copyfileobj(requests.get(download_url, stream=True).raw, out_file)
      print(f"{output_path} downloaded")
  else:
    print(f"{output_path} already present. Not downloading")
```

```
/content/gdrive/MyDrive/cs109b/law_citations/data/N.C.citations.csv.gz already
present. Not downloading
/content/gdrive/MyDrive/cs109b/law_citations/data/Ark.citations.csv.gz already
present. Not downloading
/content/gdrive/MyDrive/cs109b/law_citations/data/Ill.citations.csv.gz already
present. Not downloading
/content/gdrive/MyDrive/cs109b/law_citations/data/N.M.citations.csv.gz already
present. Not downloading
```

[18]:
```
# unzip the citations, using -k to keep archive (so we don't re-download it)
!yes n | gunzip -k data/N.C.citations.csv.gz
```

```
gzip: data/N.C.citations.csv already exists;    not overwritten
```

Create Dictionary of Source Citations and Destinations We can't use pd.read_csv() to read the citation graph cites datat, since each citation is a different length. We need to read line-by-line into a different data format. Using a dictionary here.

[19]:
```
source_file = os.path.join(data_dir, dirs[0]+filename[:-3])
citations = {}

with open(source_file) as f:
  for line in f:
    cites = line.strip().split(',')
    citations[cites[0]] = cites[1:]
```

[20]:
```
citations['8521088']
```

[20]:
```
['8551647', '8554594', '8561041', '8563251', '8564807']
```

We're adding a column in the opinions_df with the cites from the citations data - ("cites_to_from_graph") since it is different from the cites_to in the XML metadata. We'll use this more selective data for targets going forward. If it already exists in the opinions_df, no need to reload.

[21]:
```
def get_citation(row):
    try:
        return citations[str(row.name)]
    except KeyError:
        return []

if not 'cites_to_from_graph' in opinions_df.columns:
    opinions_df['cites_to_from_graph'] = opinions_df.apply(get_citation, axis=1)
```

[22]:
```
opinions_df.head()
```

[22]:
```
            index                        name  …  topic-13 top_topic
id                                             …
11274718        0  A. B. LONG v. G. W. LOGAN  …  0.006565         6
8657131         1          SHANKLE v. INGRAM  …  0.006593         0
11275047        2        STATE v. R. L. CROUSE  …  0.006847        10
11275447        3        STATE v. MATT. BRAGG  …  0.006643         0
8656063         4             HART v. CANNON  …  0.008902         0

[5 rows x 29 columns]
```

Checking to see if there are any cases that aren't cited at all.

[23]:
```
print(f"Total of uncited cases: {sum(opinions_df['cites_to_from_graph'].
 ↪isna())}")
```

```
Total of uncited cases: 0
```

1.5 Exploratory Data Analysis (EDA)

To explore overarching trends, we quantitatively and visually analyze various aspects including the court, decision date, text length, number of citations, sentence length, etc.

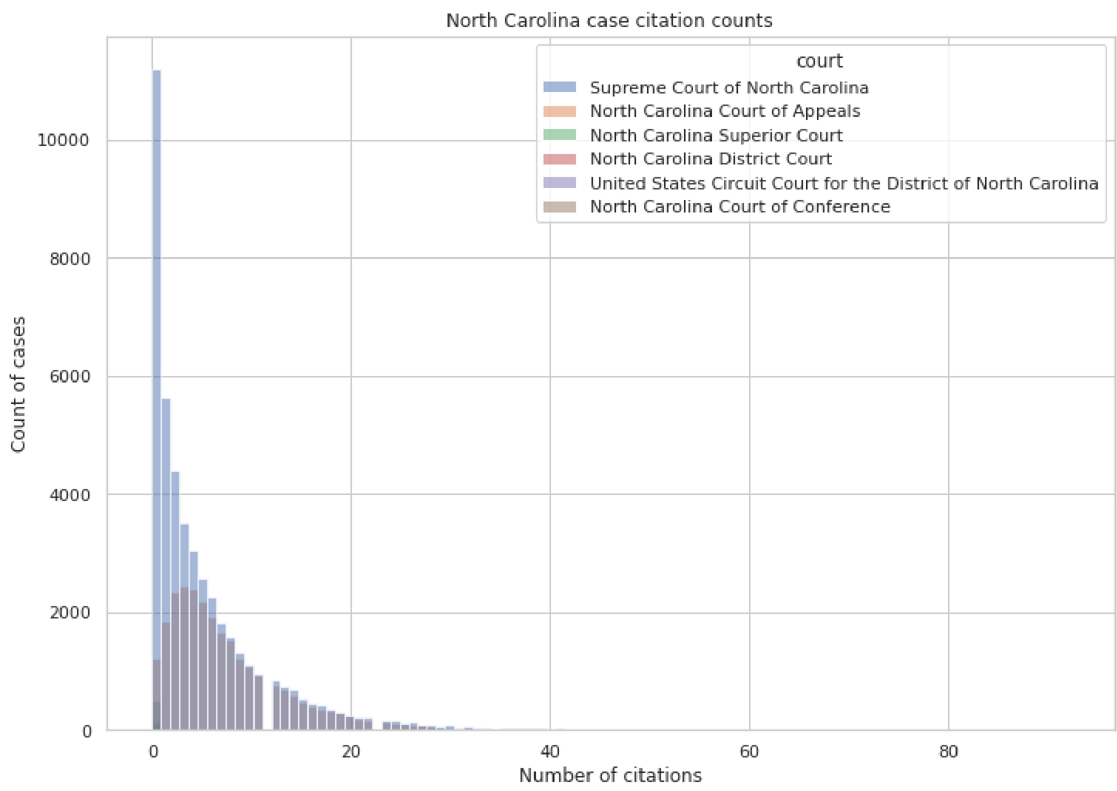
[24]:
```
pd.DataFrame(opinions_df.court.value_counts())
```

[24]:
```
                                                    court
Supreme Court of North Carolina                     45697
North Carolina Court of Appeals                     26082
North Carolina Superior Court                         498
United States Circuit Court for the District of…    132
North Carolina Court of Conference                     76
North Carolina District Court                           1
```

[25]:
```
if not 'num_cites_to_from_graph' in opinions_df.columns:
    opinions_df['num_cites_to_from_graph'] = opinions_df.cites_to_from_graph.
 ↪apply(len)
```

[26]:
```
sns.set(rc={'figure.figsize': (11.7, 8.27)})
sns.set(style="whitegrid")
```

[27]:
```
g = sns.histplot(data=opinions_df, x='num_cites_to_from_graph', hue='court',␣
↪bins=100)
g.set_xlabel('Number of citations')
g.set_ylabel('Count of cases')
g.set_title('North Carolina case citation counts');
```

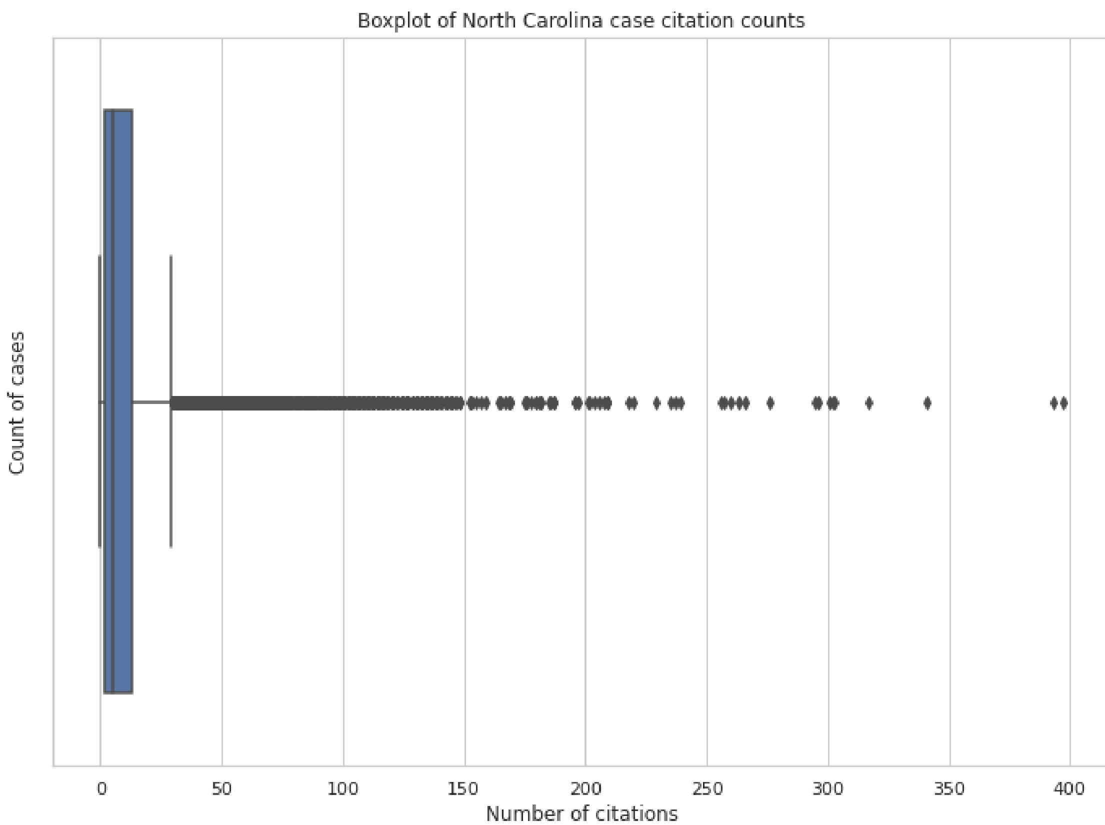


[28]:
```
np.mean(opinions_df.num_cites_to_from_graph)
```

[28]: 6.083685125403526

[29]:
```
g = sns.boxplot(opinions_df.num_cites_to)
g.set_xlabel('Number of citations')
g.set_ylabel('Count of cases')
g.set_title('Boxplot of North Carolina case citation counts')
```

[29]: Text(0.5, 1.0, 'Boxplot of North Carolina case citation counts')

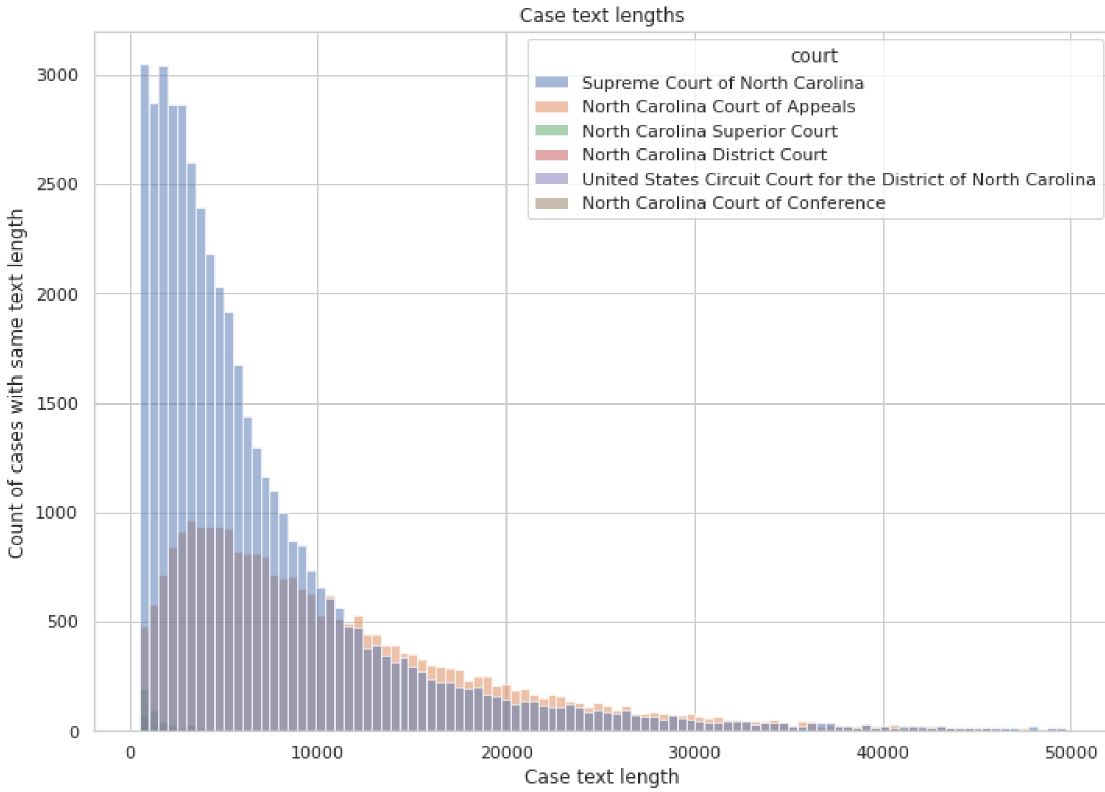


[30]:
```
if not 'text_len' in opinions_df.columns:
    opinions_df['text_len'] = opinions_df.text.apply(len)
```

[31]:
```
opinions_df.text_len.value_counts()[:10]
```

[31]:
```
1308    20
790     19
784     19
2220    19
2910    18
1528    18
4137    17
3119    17
2438    16
1460    16
Name: text_len, dtype: int64
```

[32]:
```
g = sns.histplot(data=opinions_df, x='text_len', hue='court', bins=100)
g.set_xlabel('Case text length')
g.set_ylabel('Count of cases with same text length')
g.set_title('Case text lengths');
```



[33]:
```python
np.mean(opinions_df.text_len)
```

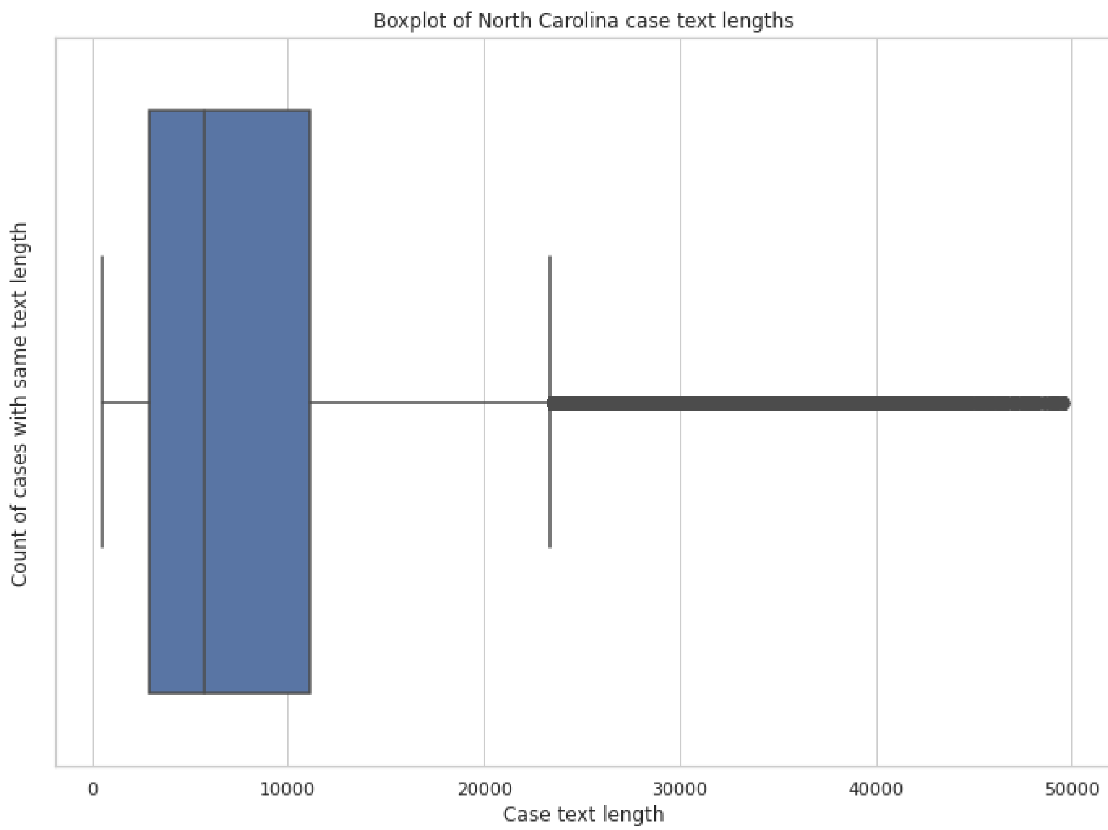
[33]: 8444.109676351296

[34]:
```python
np.mean(opinions_df[opinions_df.text_len > np.percentile(opinions_df.
    ↪text_len,5)].text_len)
```

[34]: 8847.430119082195

[35]:
```python
g = sns.boxplot(opinions_df.text_len)
g.set_xlabel('Case text length')
g.set_ylabel('Count of cases with same text length')
g.set_title('Boxplot of North Carolina case text lengths')
```

[35]: Text(0.5, 1.0, 'Boxplot of North Carolina case text lengths')



[36]:
```
opinions_df.shape
```

[36]: (72486, 30)

[37]:
```
# mean text_len of bottom 5th percentile is short
np.mean(opinions_df[opinions_df.text_len < np.percentile(opinions_df.
 ↪text_len,5)].text_len)
```

[37]: 784.141592920354

[38]:
```
# mean text_len of top 1 percentile is long
np.mean(opinions_df[opinions_df.text_len > np.percentile(opinions_df.
 ↪text_len,99)].text_len)
```

[38]: 43597.31172413793

[39]:
```
# drop short ones and super long ones based on text_len percentile
if not loaded_df:
    opinions_df = opinions_df[opinions_df.text_len > np.percentile(opinions_df.
 ↪text_len,5)]
```

```
    opinions_df = opinions_df[opinions_df.text_len < np.percentile(opinions_df.
↪text_len,99)]
```

[40]:
```
opinions_df.shape
```

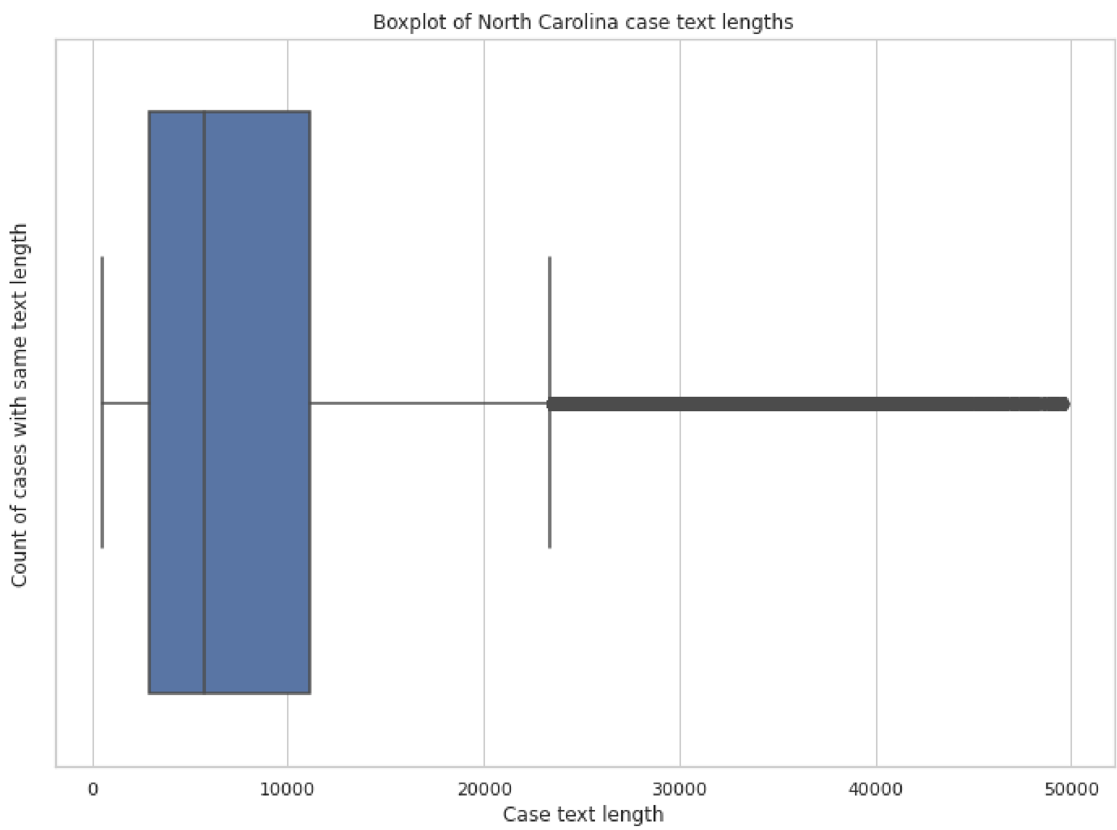
[40]: (72486, 30)

[41]:
```
np.mean(opinions_df.text_len)
```

[41]: 8444.109676351296

[42]:
```
g = sns.boxplot(opinions_df.text_len)
g.set_xlabel('Case text length')
g.set_ylabel('Count of cases with same text length')
g.set_title('Boxplot of North Carolina case text lengths')
```

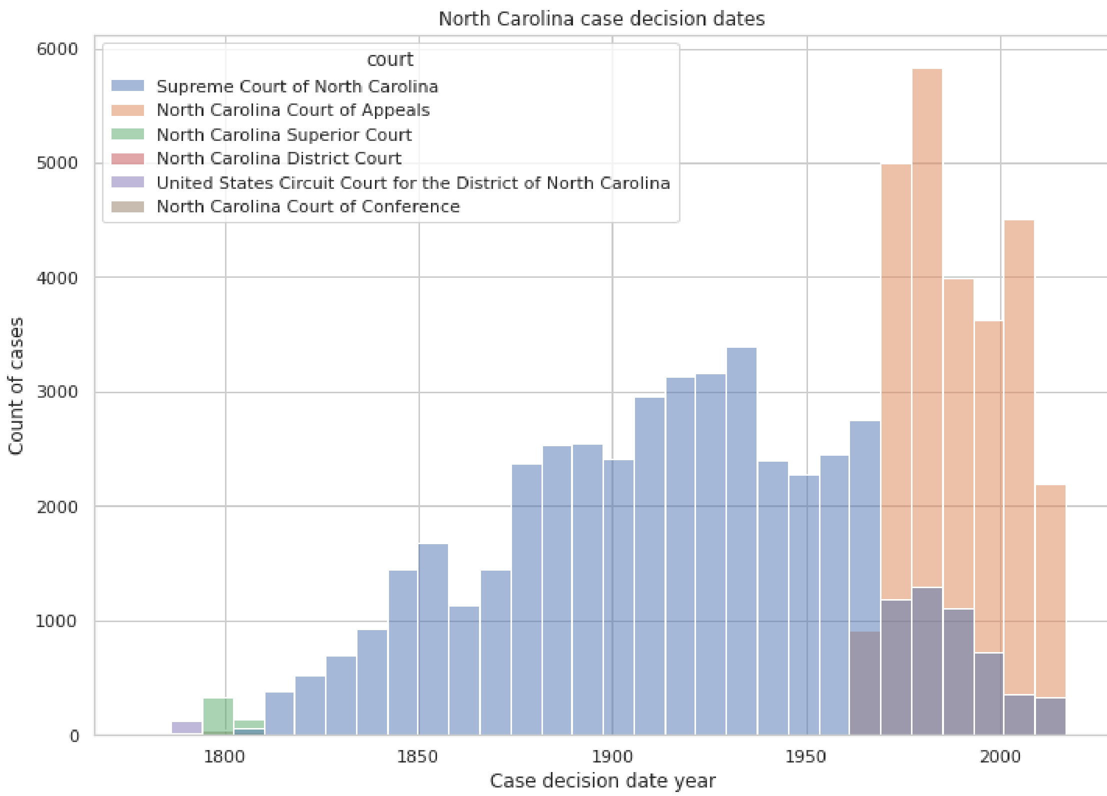
[42]: Text(0.5, 1.0, 'Boxplot of North Carolina case text lengths')



[43]:
```
opinions_df.decision_date.value_counts()[:10]
```

[43]:
```
1985    1034
1984    1010
1974     929
1980     917
1983     903
1972     879
1979     874
1975     861
1982     834
1986     805
Name: decision_date, dtype: int64
```

[44]:
```python
g = sns.histplot(data=opinions_df, x='decision_date', hue='court', bins=30)
g.set_xlabel('Case decision date year')
g.set_ylabel('Count of cases')
g.set_title('North Carolina case decision dates');
```



[45]:
```python
if not 'year' in opinions_df.columns:
    opinions_df['year'] = opinions_df['decision_date']
```

[46]:
```
%%time
# split case text into array of sentences and add to df as new column
# regex pattern thx to https://towardsdatascience.com/
↪tokenize-text-columns-into-sentences-in-pandas-2c08bc1ca790
pattern = re.compile(r"(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?|!)\s")
if not 'sents' in opinions_df.columns:
    opinions_df['sents'] = opinions_df.text.apply(lambda x: pattern.split(x))
```

```
CPU times: user 203 µs, sys: 75 µs, total: 278 µs
Wall time: 282 µs
```

[47]:
```
if not loaded_df:
    # serialize df to disk if we didn't load it from disk
    opinions_df.reset_index().to_feather(f"{dataset_dir}/{state}_opinions.
↪feather")
```

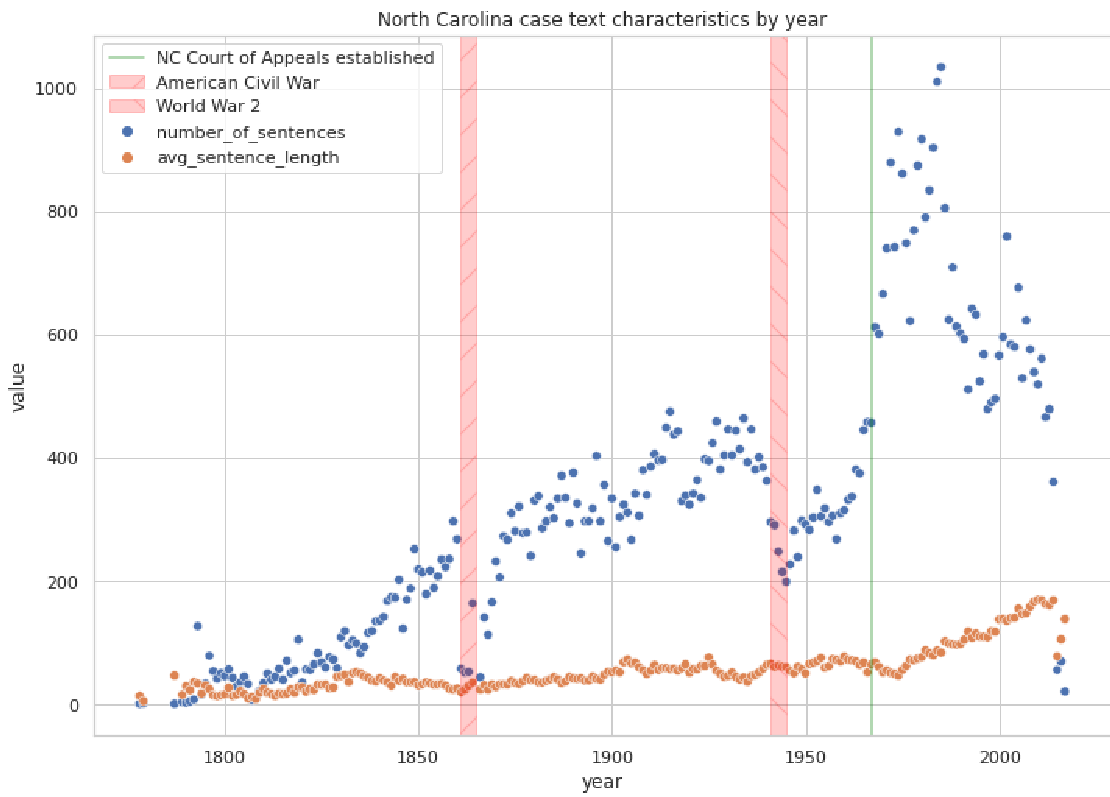
[48]:
```
number_of_sentences = (opinions_df.groupby(['year']).sents.count()
                       .reset_index(name='number_of_sentences'))

avg_sentence_length = (opinions_df.groupby('year').sents
                       .apply(lambda x: np.round(np.mean(x.str.len()), 1))
                       .reset_index(name='avg_sentence_length'))

stats = number_of_sentences.merge(avg_sentence_length, how="outer")
display(stats)
```

```
     year  number_of_sentences  avg_sentence_length
0    1778                    1                 14.0
1    1779                    2                  5.5
2    1787                    1                 47.0
3    1789                    4                 15.8
4    1790                    3                 29.7
..    …                    …                    …
226  2013                  479                161.4
227  2014                  361                169.2
228  2015                   56                 78.1
229  2016                   70                105.7
230  2017                   21                138.5

[231 rows x 3 columns]
```

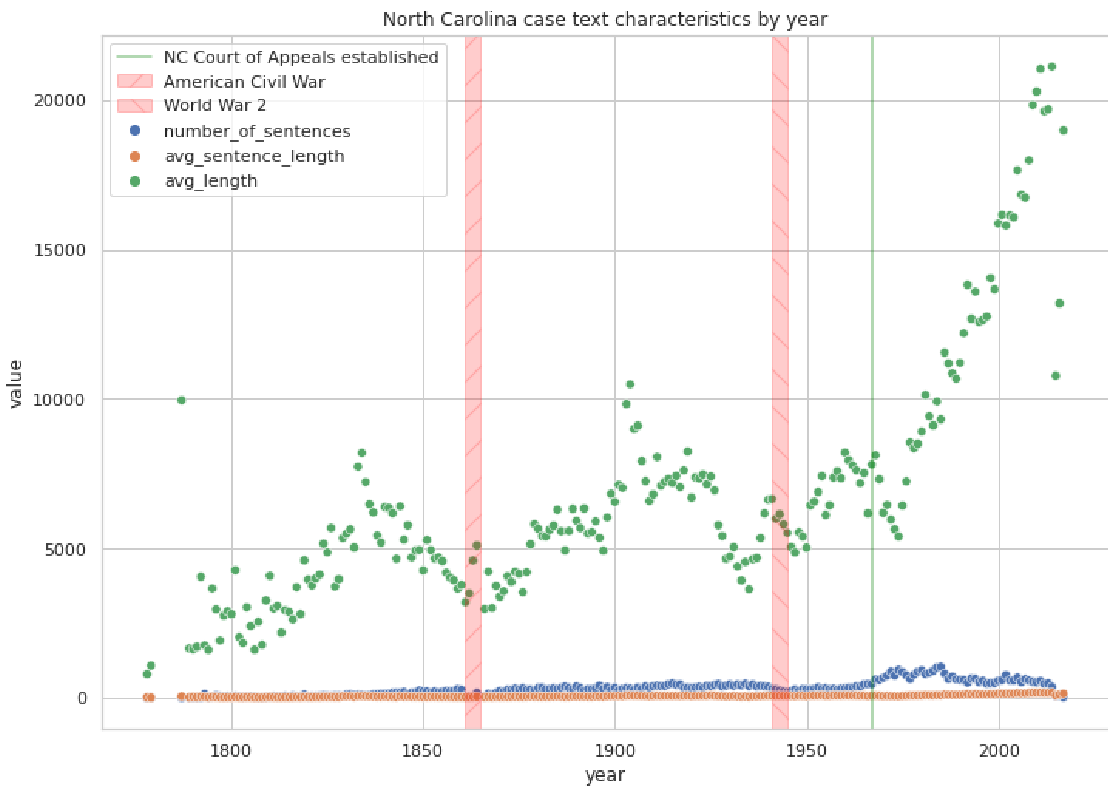
Since 1778, the number of sentences per case has steadily increased. The two prominent dips in sentence length correspond to the Civil War and World War II. Somewhat puzzling is the drastic decline in number of sentences after 1980. While some cases consisted of as many as 1000 sentences in 1970, this has fallen to 300 by 2010 (after disregarding the outlier). The reason underlying this phenomenon would be an interseting direction for future research. By comparison, average sentence length remains relatively constant, as English syntax and language structures are relatively stable.

[49]:
```
def plot_stats(stats):
    g = sns.scatterplot(x="year", y="value", hue="variable",
                        data=pd.melt(stats, ['year']))
    g.axvspan(1861, 1865, alpha=0.2, color='red', hatch='/', label='American␣
↪Civil War')
    g.axvspan(1941, 1945, alpha=0.2, color='red', hatch='\\', label='World War␣
↪2')
    g.axvline(1967, alpha=0.4, color='green', label='NC Court of Appeals␣
↪established')
    g.legend()
    g.set_title(f"North Carolina case text characteristics by year");
```

[50]:
```
plot_stats(stats)
```



[51]:
```
avg_length = (opinions_df.groupby(['year']).text_len.mean()
              .reset_index(name='avg_length'))

plot_stats(stats.merge(avg_length, how="outer"))
```



[52]:
```
from collections import Counter

citations_counts = Counter()
opinions_df.cites_to_id.apply(lambda x: citations_counts.update(x));
```

[53]:
```
top_cited = citations_counts.most_common(1000)
top_cited[:10]
```

[53]:
```
[(12046400, 1108),
 (8565416, 516),
 (6167798, 498),
 (6204802, 409),
 (6157001, 398),
 (8573434, 372),
 (8559773, 360),
 (8561041, 336),
 (8629835, 333),
 (6168882, 318)]
```

[54]:
```
from operator import itemgetter
top_cited_ids = list(map(itemgetter(0), top_cited))
```

[55]:
```
unknown_cases = list()
def get_year_for_case(id):
    try:
        case = opinions_df.loc[id]
        return case['year']
    except KeyError:
        unknown_cases.append(id)

top_cited_years = Counter(filter(None, (map(get_year_for_case, top_cited_ids))))
print(f"excluding {len(unknown_cases)} unknown cases\n")

print("years with most top cited cases:")
top_cited_years.most_common(25)
```

```
excluding 479 unknown cases

years with most top cited cases:
```

[55]:
```
[(1972, 23),
 (1977, 18),
 (1980, 17),
 (1971, 17),
 (1970, 16),
 (1983, 15),
 (1985, 15),
 (1979, 15),
 (1978, 15),
 (1974, 15),
 (1986, 15),
 (1982, 14),
 (1975, 14),
 (1976, 13),
 (1967, 12),
 (1969, 12),
 (1981, 11),
 (1968, 10),
 (1988, 9),
 (1984, 8),
 (1991, 8),
 (1973, 8),
 (1965, 7),
 (2000, 7),
 (1987, 7)]
```

1.6 Topics

We download and utilize the English Spacy model, which components three components, including named entity recognition, part-of-speech tagging, and dependency parsing, as printed below. For spell-checking, we import the SymSpell and Verbosity modules.

[56]:
```
# download English spacy model
# sometimes this fails for mysterious colab reasons
# so retry if necessary...
subprocess.check_call([sys.executable, '-m', 'spacy', 'download',␣
 ↪'en_core_web_md'], stdout=subprocess.DEVNULL)
```

[56]: 0

[57]:
```
import spacy
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation

import en_core_web_md
# race condition for colab with google drive:
# above cell's downloaded model may not
# be available to read immediately, so
# `en_core_web_md.load()` may fail.
# reruning this cell a moment later should resolve
nlp = en_core_web_md.load()
print(nlp.pipe_names)

import pyLDAvis
import pyLDAvis.sklearn
pyLDAvis.enable_notebook()
```

```
['tagger', 'parser', 'ner']
```

```
/usr/local/lib/python3.7/dist-packages/past/types/oldstr.py:5:
DeprecationWarning: Using or importing the ABCs from 'collections' instead of
from 'collections.abc' is deprecated since Python 3.3,and in 3.9 it will stop
working
  from collections import Iterable
```

[58]:
```
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

[59]:
```
USE_SPELLCHECK = False
```

[60]:
```
if USE_SPELLCHECK:
    dictionary_path = pkg_resources.resource_filename("symspellpy",␣
 ↪"frequency_dictionary_en_82_765.txt")
    bigram_path = pkg_resources.resource_filename("symspellpy",␣
 ↪"frequency_bigramdictionary_en_243_342.txt")
```

[61]:
```
if USE_SPELLCHECK:
    from symspellpy import SymSpell
    from symspellpy import Verbosity
```

```
    sym_spell = SymSpell(max_dictionary_edit_distance=2, prefix_length=7)
    # term_index is the column of the term and count_index is the
    # column of the term frequency
    sym_spell.load_dictionary(dictionary_path, term_index=0, count_index=1)
```

[62]:
```
if USE_SPELLCHECK:
    # https://symspellpy.readthedocs.io/en/latest/examples/lookup.
 ↪html#return-original-word-if-no-correction-within-edit-distance-is-found
    # lookup suggestions for single-word input strings
    input_term = "tbe"  # misspelling of "the"
    print(f"symspellpy example input:\n\t{input_term}\n")

    # max edit distance per lookup
    # (max_edit_distance_lookup <= max_dictionary_edit_distance)
    suggestions = sym_spell.lookup(input_term, Verbosity.CLOSEST,
                                   max_edit_distance=2, include_unknown=True)
    # display suggestion term, term frequency, and edit distance
    print("symspellpy suggestions (term, edit distance, term frequency)")
    for n, suggestion in enumerate(suggestions):
        print(f"\t{n}: {suggestion}")

    feeling_lucky = suggestions[0]._term
    print(f"\n'I\'m feeling lucky' (first result):\n\t{feeling_lucky}")
```

[63]:
```
# common OCR errors
nlp.Defaults.stop_words |= {"tbe","tbis","tbat","tben","tne"}
# common uninformative legal words
nlp.Defaults.stop_words |=␣
 ↪{"case","court","defendent","state","trial","evidence","charge","judge","counsel","testimon
```

[64]:
```
assert 'tbe' in nlp.Defaults.stop_words
```

[65]:
```
# workaround for spacy bug https://github.com/explosion/spaCy/issues/
 ↪922#issuecomment-360135141
nlp.vocab.add_flag(lambda s: s.lower() in spacy.lang.en.stop_words.STOP_WORDS,␣
 ↪spacy.attrs.IS_STOP)
```

[65]: 12

[66]:
```
import string

def is_nice_token(token):
    return ((not token.is_stop) and (not token.is_punct) and
            (not token.is_digit) and (len(token) > 2))

def modify_token(token):
```

```
    # returns string

    # if token has an entity type, return entity type
    # https://spacy.io/api/annotation#named-entities
    if token.ent_type_:
        return token.ent_type_

    if token.is_oov:
        text = token.text.translate(str.maketrans('', '', string.punctuation))

        if USE_SPELLCHECK:
            # if token is not in the Spacy English vocabulary,
            # blindly try to correct spelling. with a small max_edit_distance,
            # this should take care of many minor OCR errors
            suggestions = sym_spell.lookup(text, Verbosity.CLOSEST,
                                        max_edit_distance=2,␣
↪include_unknown=True)
            # return top suggestion (or original if no close suggestions are␣
↪found)
            return suggestions[0]._term
        return text

    # otherwise, return lemmatized and lowercased
    return token.lemma_.lower()

def tokenizer_spacy(text):
    doc = nlp(text)
    filtered = list(filter(is_nice_token, doc))
    return list(map(modify_token, filtered))
```

[67]:

```
print(tokenizer_spacy("tbe ##34 nb nbb convicted 23)sd lkj)we:hn ended walking␣
↪wtf , lol Charlotte Matthew $50 dollars the"))
```

```
['nbb', 'convict', '23sd', 'lkjwe', 'end', 'walk', 'PERSON', 'lol', 'PERSON',
'PERSON', 'MONEY']
```

[68]:

```
print(tokenizer_spacy("There must be both allegation and proof to entitle a␣
↪party to the relief he seeks. McKee v. Lineberger,"))
```

```
['allegation', 'proof', 'entitle', 'relief', 'seek', 'PERSON', 'PERSON']
```

[69]:

```
print(tokenizer_spacy("the deceased being on board a steamer received a shock␣
↪from the bursting of the boiler, and that boiling water, coal, &c., were␣
↪thereby thrown against deceased, of which shock, &c., the deceased instantly␣
↪died;"))
```

```
['deceased', 'board', 'steamer', 'receive', 'shock', 'bursting', 'boiler',
'boil', 'water', 'coal', 'throw', 'deceased', 'shock', 'deceased', 'instantly',
'die']
```

[70]:
```
print(tokenizer_spacy(" And this decision is approved in University v.␣
↪Lassiter, . See also Johnson v. Rowland, ; Boddie v. Woodard, ; Reese v.␣
↪Jones, ; Henry v. Cannon, ante, 24; Gilchrist v. Kitchen, ante, 20; Hinton v.
↪ Deems, ; State v. Laman, .\n*538The Code of Civil Procedure, says Bynum, J.
↪, in the case of Austin v. Clarke,"))
```

```
['decision', 'approve', 'ORG', 'ORG', 'PERSON', 'rowland', 'PERSON', 'PERSON',
'LAW', 'LAW', 'PERSON', 'PERSON', 'ante', 'PERSON', 'kitchen', 'ante', 'ORG',
'deems', 'PERSON', 'LAW', 'LAW', 'LAW', 'LAW', 'say', 'PERSON', 'PERSON',
'PERSON']
```

[71]:
```
print(opinions_df.shape)
docs = opinions_df.text.values
dataset_key = str(docs.shape[0])
print(dataset_key)
docs.shape
```

```
(72486, 30)
72486
```

[71]: (72486,)

[72]:
```
import pickle

# stuff a 'version' into tokenizer name to keep track of substantial
# changes to tokenization/preprocessing
tokenizers = {'spacy02': tokenizer_spacy}#, 'bert': tokenizer_bert}

def get_or_make_vectors(docs, tokenizer_name='spacy02'):
    tfidf_name = f"{state}_tfidf-{docs.shape[0]}{tokenizer_name}"
    vectorizer_name = f"{state}_vectorizer-{docs.shape[0]}{tokenizer_name}"
    try:
        tfidf = pickle.load(open(f"{dataset_dir}/{tfidf_name}.pkl", 'rb'))
        vectorizer = pickle.load(open(f"{dataset_dir}/{vectorizer_name}.pkl",␣
↪'rb'))
    except:
        # make vectors based on term frequency- inverse document frequency.
        # discard tokens that appear in fewer than 10 docs,
        # as well as those appearing in over 95% of docs
        vectorizer = TfidfVectorizer(tokenizer=tokenizers[tokenizer_name],
                                     min_df=10, max_df=0.95)
        tfidf = vectorizer.fit_transform(docs)
        pickle.dump(tfidf, open(f"{dataset_dir}/{tfidf_name}.pkl", "wb"))
        pickle.dump(vectorizer, open(f"{dataset_dir}/{vectorizer_name}.pkl",␣
↪"wb"))
    return vectorizer, tfidf
```

[73]:
```
%%time
vectorizer, tfidf = get_or_make_vectors(docs, 'spacy02')
```

```
CPU times: user 178 ms, sys: 269 ms, total: 446 ms
Wall time: 2.92 s
```

[74]:
```
%%time
from sklearn.model_selection import GridSearchCV

def gridsearch_lda(params):
    lda = LatentDirichletAllocation()
    model = GridSearchCV(lda, param_grid=search_params)
    model.fit(tfidf)

    best_lda_model = model.best_estimator_

    print("Best Model's Params: ", model.best_params_)
    print("Best Log Likelihood Score: ", model.best_score_)
    print("Model Perplexity: ", best_lda_model.perplexity(tfidf))
    return best_lda_model

search_params = {'n_components': [10, 15, 20, 25, 30, 35, 40, 45, 50],
                 'learning_decay': [.5, .7, .9]}
# while gridsearch's best model (10 components, 0.9 learning decay)
# has the best log likelihood score, it's first topic includes
# over 70% of tokens so isn't really useful as a topic model
#lda_model = gridsearch_lda(search_params)
```

```
CPU times: user 10 µs, sys: 0 ns, total: 10 µs
Wall time: 12.4 µs
```

[75]:
```
import joblib

def get_or_make_lda(tfidf, n_components=14):
    try:
        lda_model = joblib.load(f"{dataset_dir}/{state}_lda{n_components}-72486.
↪jl")
    except FileNotFoundError:
        lda = LatentDirichletAllocation(n_components=n_components,
                                        learning_decay=0.5,
                                        learning_offset=30.0)
        lda_model = lda.fit(tfidf)
        # sklearn recommends joblib over pickle for trained models
        # https://scikit-learn.org/stable/modules/model_persistence.html
        joblib.dump(lda_model, f"{dataset_dir}/{state}_lda{n_components}-72486.
↪jl")
        return lda_model
```

[76]:
```
%%time
n_topics = 14
#lda_model = get_or_make_lda(tfidf, n_components=n_topics)
lda_model = joblib.load(f"{dataset_dir}/nc_lda14-72486.jl")
```

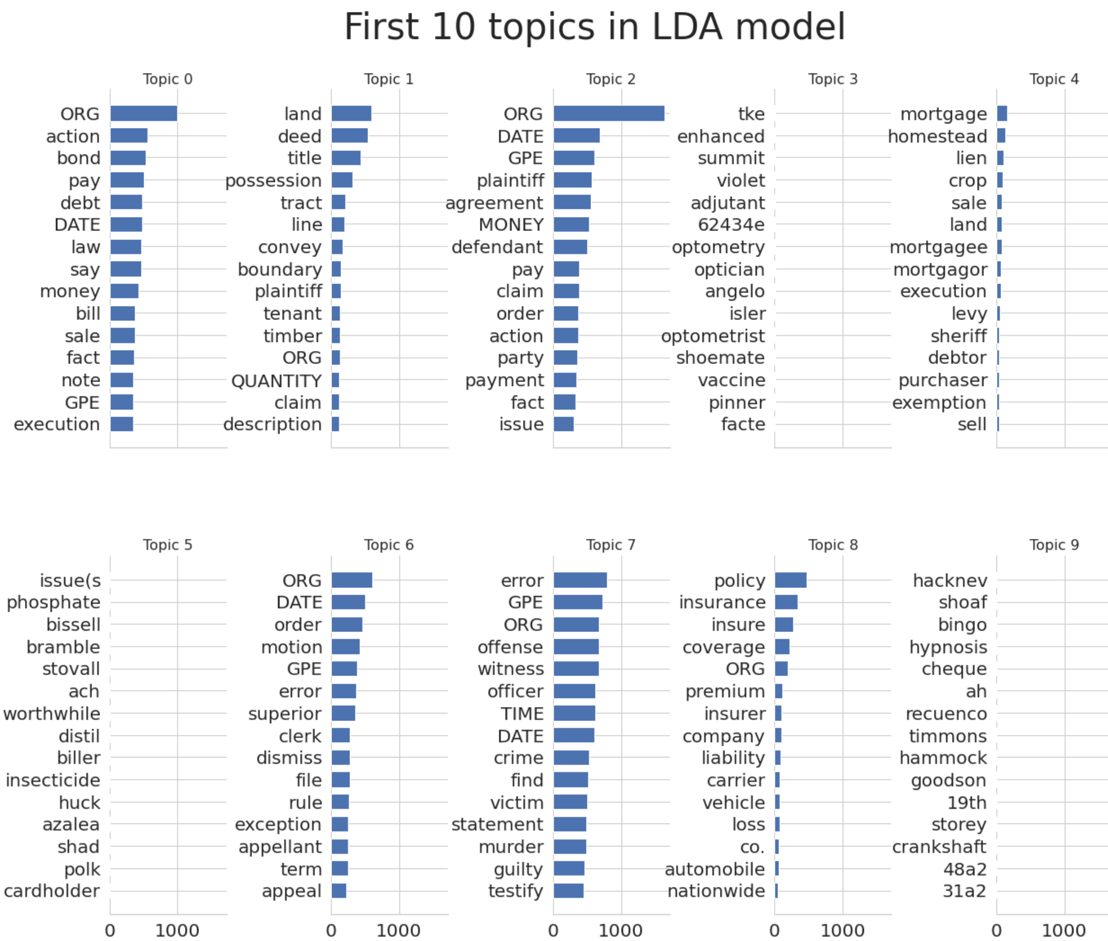
```
CPU times: user 5.15 ms, sys: 2.68 ms, total: 7.83 ms
Wall time: 17 ms
```

[77]:
```
print(f"Log Likelihood: {lda_model.score(tfidf):.2f}")
print(f"Perplexity: {lda_model.perplexity(tfidf):.2f}")
```

```
Log Likelihood: -5885664.35
Perplexity: 6600.77
```

[78]:
```
def plot_top_words(model, feature_names, n_top_words, title):
    # adapted from https://scikit-learn.org/stable/auto_examples/applications/
 ↪plot_topics_extraction_with_nmf_lda.
 ↪html#sphx-glr-auto-examples-applications-plot-topics-extraction-with-nmf-lda-py
    fig, axes = plt.subplots(2, 5, figsize=(20, 15), sharex=True)
    axes = axes.flatten()
    for topic_idx, topic in enumerate(model.components_[:10]):
        top_features_ind = topic.argsort()[:-n_top_words - 1:-1]
        top_features = [feature_names[i] for i in top_features_ind]
        weights = topic[top_features_ind]

        ax = axes[topic_idx]
        ax.barh(top_features, weights, height=0.7)
        ax.set_title(f'Topic {topic_idx}',
                     fontdict={'fontsize': 16})
        ax.invert_yaxis()
        ax.tick_params(axis='both', which='major', labelsize=20)
        for i in 'top right left'.split():
            ax.spines[i].set_visible(False)
        fig.suptitle(title, fontsize=40)

    plt.subplots_adjust(top=0.90, bottom=0.05, wspace=0.90, hspace=0.3)
    plt.show()
```

[79]:
```
plot_top_words(lda_model, vectorizer.get_feature_names(), 15, 'First 10 topics␣
 ↪in LDA model')
```



The following interactive interface displays the 14 clusters produced by the LDA model, as well as the intertopic distance map and most relevant terms by topic. The different clusters can be thought of to generally describe the following topics:

- Cluster 1: Criminal Offense
- Cluster 2: Financial Transactions
- Cluster 3: Contractual Relationships
- Cluster 4: Employment and Workplace Injuries
- Cluster 5: Municipal Public Administration (Education, Taxation, and other Infrastructure)
- Cluster 6: Legal Procedures
- Cluster 7: Inheritance
- Cluster 8: Miscellaneous
- Cluster 9: Real Estate and Property
- Cluster 10: Insurance
- Cluster 11: Mortgages
- Cluster 12: Miscellaneous
- Cluster 13: Miscellaneous
- Cluster 14: Industrial and Agricultural Production

Saliency is displayed in blue, while relevance is shown in red. We observe that overall, the most salient terms were terms categorized as `ORG` organizations, which include a vast array of private and public entities, as well as local and state jurisdictions. `DATE` and `GPE` (geogolitical entity) are the next two most salient term categories. Disregarding these proper nouns, we observe that "land", "defendant," "plaintiff," "action," "deed," and "claim" are the most salient terms.

We additionally note the distinction between salience/relevance and frequency. Saliency and relevance are computed as

$$\text{Saliency}(w) = \text{Frequency}(w) \times [\sum_t p(t|w) \times \log(\frac{p(t|w))}{p(t)}]$$

$$\text{Relevance}(w|t) = \lambda \times p(t|w) + (1-\lambda) \times \frac{p(t|w))}{p(w)}]$$

for topic t and term w.

Since relevance is a convex combination of the marginal probability of observing a topic-term combination and the conditional probability of observing a topic-term combination conditional upon observing the term, adjusting the parameter λ to smaller values allows us to put greater emphasis on the second term and observe which terms are more salient for the particular topic in question.

Furthermore, from the Intertopic Distance Map (constructed with multidimensional scaling), we find that the clusters appear to feature little no overlap along the top two principal component dimensions, indicating that our multicolinearity is unlikely to be an issue.

[80]:
```
%%time
pyLDAvis.sklearn.prepare(lda_model, tfidf, vectorizer, mds='tsne')
```

```
CPU times: user 34.1 s, sys: 59 s, total: 1min 33s
Wall time: 28.9 s
```

[80]:
```
PreparedData(topic_coordinates=                x           y  topics  cluster
Freq
topic
7        1.425140 -203.131439       1        1  20.387109
0       93.696953  -55.018154       2        1  17.966058
2      141.694016   25.320429       3        1  13.933368
11    -126.233505    5.739788       4        1  12.080774
13     -59.868134  101.519302       5        1  11.955243
6      151.966629  133.119141       6        1   6.898430
10      12.542690 -101.441658       7        1   6.160299
12      49.372585   57.773476       8        1   3.259136
1      203.505997  -84.541840       9        1   2.748666
8      119.923477 -166.068970      10        1   1.508381
4      -19.315592   -8.882428      11        1   0.999671
9      -95.841057 -109.373474      12        1   0.702308
3       37.554749  165.057098      13        1   0.700445
```

```
5      243.648911   32.733547      14         1   0.700111, topic_info=
Term          Freq         Total Category  logprob  loglift
2741             ORG  6953.000000  6953.000000  Default  30.0000  30.0000
14336           land  1640.000000  1640.000000  Default  29.0000  29.0000
7956            deed  1328.000000  1328.000000  Default  28.0000  28.0000
18141         policy   808.000000   808.000000  Default  27.0000  27.0000
20536           sale   989.000000   989.000000  Default  26.0000  26.0000
…               …           …          …       …       …       …
19100  racketeering     1.940955     4.683256  Topic14  -7.7890   4.0809
15345      mcculloch     2.282354     6.293624  Topic14  -7.6270   3.9474
12489         humble     2.161134     9.323834  Topic14  -7.6815   3.4997
18893     publishing     2.223556    11.241255  Topic14  -7.6531   3.3412
16373           news     2.160925    20.374855  Topic14  -7.6816   2.7179

[937 rows x 6 columns], token_table=       Topic      Freq    Term
term
358         1  0.055905     1.1
358         3  0.838568     1.1
358         5  0.055905     1.1
396        12  0.520265  108a57
482         3  0.238009     12b
…      …       …      …
25887       3  0.009148    zone
25887       4  0.091484    zone
25887       5  0.850805    zone
25889       3  0.019892  zoning
25889       5  0.954796  zoning

[3886 rows x 3 columns], R=30, lambda_step=0.01, plot_opts={'xlab': 'PC1',
'ylab': 'PC2'}, topic_order=[8, 1, 3, 12, 14, 7, 11, 13, 2, 9, 5, 10, 4, 6])
```

The silhouette plot and PCA decomposition visualizes the quality of separation between clusters. From the silhouette plot, the first 9 clusters have silhouette coefficients that are exclusively positive, suggesting that intracluster similarity is high, characteristic of high quality clustering. Negative silhouette scores become more prevalent for later/higher-numbered clusters, suggesting that inter-group similarity is higher than intragrouop similarity, which aligns with our previous observation that later clusters mostly capture residual, miscellaneous topics that are not incorporated in previous umbrella topics.

[81]:
```python
import matplotlib.cm as cm
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
from sklearn.metrics import silhouette_samples

# function from HW2
#modified code from http://scikit-learn.org/stable/auto_examples/cluster/
 ↪plot_kmeans_silhouette_analysis.html
```

```python
def silplot(X, cluster_labels, clusterer, pointlabels=None):
    n_clusters = clusterer.n_clusters

    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(11,8.5)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example we
    # will set a limit
    ax1.set_xlim([-0.2, 1])

    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters = ", n_clusters,
          ", the average silhouette_score is ", silhouette_avg,".",sep="")

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(0,n_clusters+1):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)
        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples
```

```
    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")
    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.2, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)

    # axes will be first 2 PCA components
    pca = PCA(n_components=2).fit(X)
    X_pca = pca.transform(X)
    ax2.scatter(X_pca[:, 0], X_pca[:, 1], marker='.', s=200, lw=0, alpha=0.7,
                c=colors, edgecolor='k')
    xs = X_pca[:, 0]
    ys = X_pca[:, 1]

    if pointlabels is not None:
        for i in range(len(xs)):
            plt.text(xs[i],ys[i],pointlabels[i])

    # Labeling the clusters (transform to PCA space for plotting)
    centers = pca.transform(clusterer.cluster_centers_)
    # Draw white circles at cluster centers
    ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
                c="white", alpha=1, s=200, edgecolor='k')

    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % int(i), alpha=1,
                    s=50, edgecolor='k')

    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("PC1")
    ax2.set_ylabel("PC2")

    plt.suptitle(("Silhouette analysis for KMeans clustering \n on␣
↪document-topic probabilities"
                 " \n from 14 component LDA model with n_clusters = %d" %␣
↪n_clusters),
                 fontsize=14, fontweight='bold')
    return silhouette_avg
```

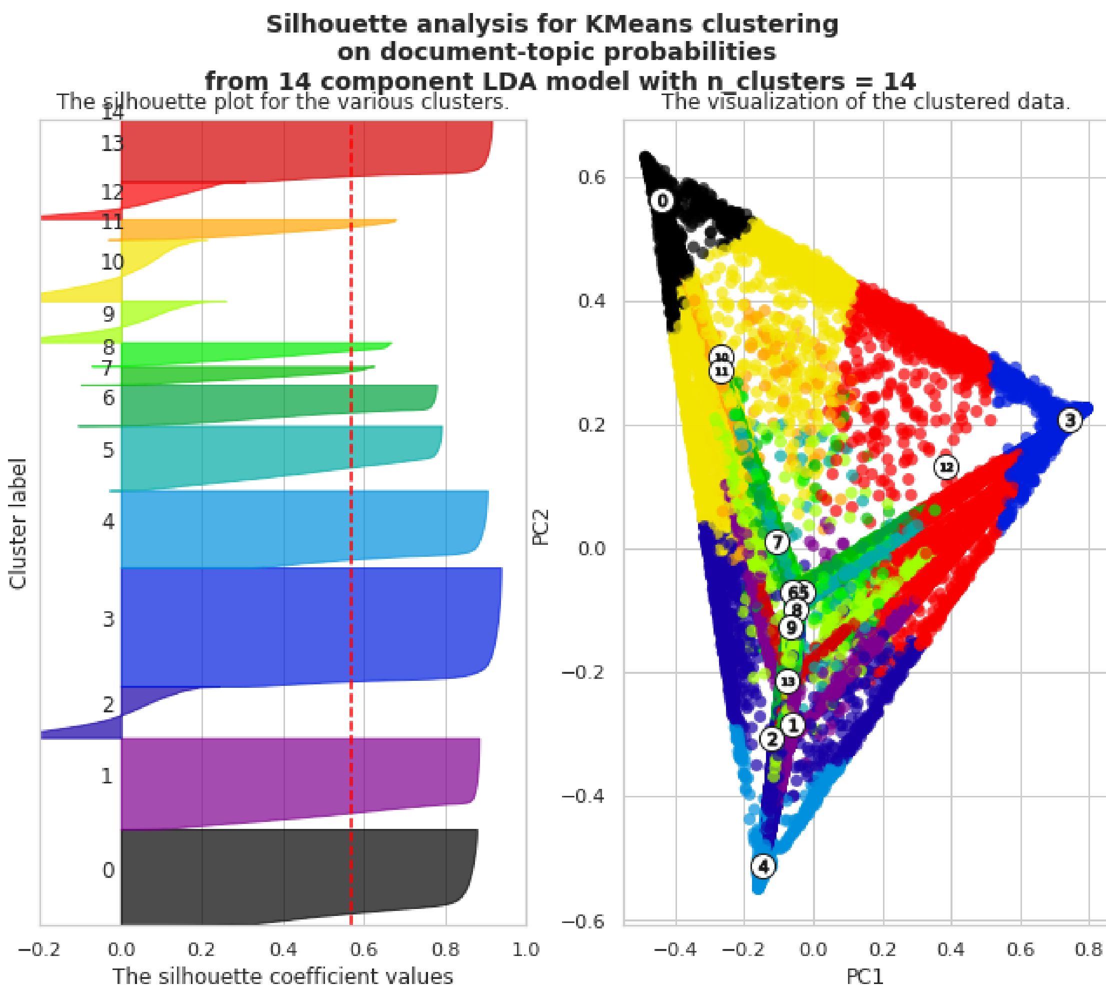
Next, we implement KMeans clustering. We use the Elbow method to infer that the optimal number of clusters is 14, based on the evolution of silhouette scores across a range of possible cluster numbers.

[82]:
```
%%time
from sklearn.cluster import KMeans

# get document-topic probabilities
lda_output = lda_model.transform(tfidf)
```

```
CPU times: user 33.6 s, sys: 58.5 s, total: 1min 32s
Wall time: 24 s
```

[83]:
```
# see what the topics look like using n_clusters=n_components
# (e.g., use same number of clusters as LDA components)
clusters_topics = KMeans(n_clusters=n_topics).fit(lda_output)
sil_avg = silplot(lda_output, clusters_topics.labels_, clusters_topics)
```

```
For n_clusters = 14, the average silhouette_score is 0.5709619863045728.
```



1.7 Feature engineering

[84]:
```
doc_topic_probs_df = pd.DataFrame(lda_output)
doc_topic_probs_df.columns = ['topic-' + str(c) for c in doc_topic_probs_df.
 ↪columns]
doc_topic_probs_df = doc_topic_probs_df.assign(top_topic=np.argmax(lda_output,␣
 ↪axis=1))
```

[85]:
```
opinions_df = opinions_df.reset_index()
assert doc_topic_probs_df.shape[0] == opinions_df.shape[0]
```

[86]:
```
opinions_df = pd.merge(opinions_df, doc_topic_probs_df, left_index=True,␣
 ↪right_index=True)
```

The below dataframe shows the predictor and response variables of interest, and is constructed as a subset of the `opinions_df` dataframe. The predictor is the text column, the outcome is the citations column (`cites_to_from_graph`), and 11 separate models are separately fitted based on the categorical variable `top_topic`.

[87]:
```
opinions_df.shape
```

[87]:
```
(72486, 46)
```

[88]:
```
# if needed, uncomment to re-serialize df
#opinions_df.reset_index().to_feather(f"{dataset_dir}/{state}_df-{docs.
 ↪shape[0]}.feather")
opinions_df = pd.read_feather(f"{dataset_dir}/{state}_df-72486.feather")
```

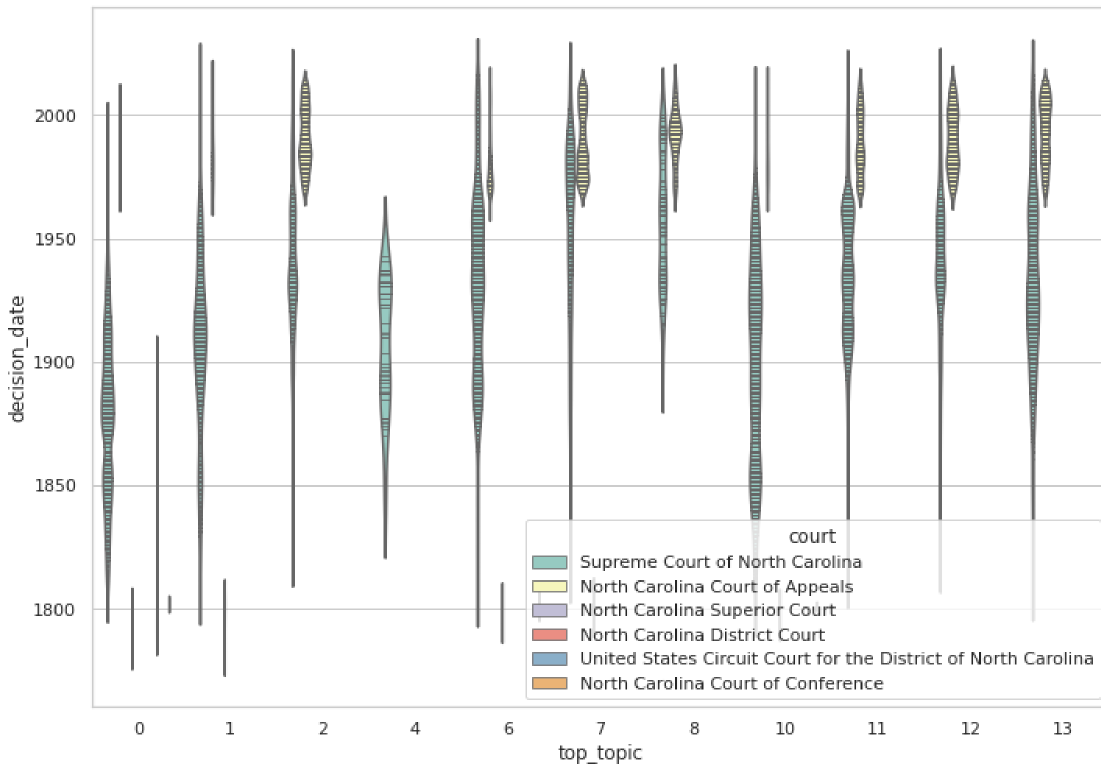
[89]:
```
clusters_col = 'top_topic'
```

[90]:
```
for group, rows in opinions_df.groupby(by=clusters_col):
    cited_ids = list(itertools.chain.from_iterable(rows.cites_to_from_graph.
 ↪values))
    print(f'cluster {group}: {len(rows)} cases with {len(cited_ids)} citations␣
 ↪to {len(set(cited_ids))} cases')
```

```
cluster 0: 15392 cases with 40554 citations to 14812 cases
cluster 1: 1490 cases with 8056 citations to 3338 cases
cluster 2: 11014 cases with 80500 citations to 23031 cases
cluster 4: 53 cases with 171 citations to 132 cases
cluster 6: 6219 cases with 25149 citations to 11144 cases
cluster 7: 13675 cases with 126834 citations to 18868 cases
cluster 8: 423 cases with 2446 citations to 1075 cases
cluster 10: 4491 cases with 20607 citations to 7405 cases
cluster 11: 8763 cases with 63040 citations to 14724 cases
cluster 12: 2041 cases with 14117 citations to 6131 cases
cluster 13: 8925 cases with 59508 citations to 16961 cases
```

Below we visualize a violinplot for the decision date across topic clusters and sorted by court. We observe that a majority of case opinions are penned by the NC Supreme Court, as it was the State's only appellate court until the creation of the Court of Appeals. After the NC General Assembly created the Court of Appeals in 1967 following a constitutional ammendment "to relieve pressure on the North Carolina Supreme Court," we observe a substantial decline proportion of NC Supreme Court cases and a corresponding increase cases decided by the Court of Appeals. Meanwhile, cases decided by the Court of Conference were exclusively in earlier periods, as the Court of Conference is the former name of the Supreme Court. This pattern is relatively consistent across all topics.

[91]:
```
g = sns.violinplot(data=opinions_df, x=clusters_col, y='decision_date',
                   scale="count", inner="stick", hue='court', palette="Set3")
```



1.8 Modeling

[91]:
```

```

1.8.1 Baseline Model

For a baseline model, we are looking at the top 3000 cases cited, and predicting for any case, the average number of citations for all cases, and the top cited for that many. We determine a baseline Normalized Discounted Cumulative Gain (NDCG) score for this model and its naive predictions. Raw accuracy isn't suitable because of the sheer number of possible predictions. Our naive model yielded a NDCG score of 0.0767, a low score that is to be expected.

```
[92]: # Accumulate all citations that have occurred
all_citations = []
for citations in opinions_df['cites_to_from_graph']:
    all_citations += list(citations)

# Count the number of times a case is cited
all_citations = Counter(all_citations)
```

```
[93]: import operator
# Our target variable will be an array of length 3000, where the ith element is␣
 ↪1 if the
# i-th most cited case is cited by the observed case, 0 otherwise
num_cases = 3000
mlb = MultiLabelBinarizer(classes=list(map(operator.itemgetter(0),␣
 ↪all_citations.most_common(num_cases))))
targets = mlb.fit_transform(opinions_df['cites_to_from_graph'])
```

```
[94]: if not 'num_cites_to_from_graph' in opinions_df.columns:
    opinions_df['num_cites_to_from_graph'] = opinions_df.cites_to_from_graph.
 ↪apply(len)
```

```
[95]: # Find the average amount of cases cited
avg_citations = np.mean(opinions_df.num_cites_to_from_graph)
avg_citations
```

```
[95]: 6.083685125403526
```

```
[96]: # Our naive model will predict that each case cites "avg_citations" cases, all␣
 ↪being
# the "avg_citations" most frequently cited cases
naive_predictions = [[1]*int(avg_citations) +␣
 ↪[0]*(num_cases-int(avg_citations))]*len(opinions_df)
```

```
[97]: # Calculate the NDCG score using our naive predictions
ndcg_score(targets, naive_predictions)
```

```
[97]: 0.07674996678084936
```

1.8.2 Preliminary Setup

The below sections of code import relevant Python libraries, set working directory, and mount drive for Google Collab

```
[98]: import nltk
import tensorflow as tf
import matplotlib.pyplot as plt
import matplotlib.cm as cm
```

```
# useful structures and functions for experiments
from time import sleep
from collections import Counter
from collections import defaultdict
from glob import glob

# specific machine learning functionality
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from tensorflow import keras
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization
from tensorflow.keras.utils import to_categorical
from tensorflow.python.keras import backend as K
from tensorflow.python.keras.utils.layer_utils import count_params
from sklearn.model_selection import train_test_split
from sklearn import manifold
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import f1_score, confusion_matrix
from transformers import BertTokenizer, TFBertForSequenceClassification,␣
↪BertConfig
from transformers import GPT2Tokenizer, TFGPT2LMHeadModel
```

[99]:
```
# Enable/Disable Eager Execution
# Reference: https://www.tensorflow.org/guide/eager
# TensorFlow's eager execution is an imperative programming environment that␣
↪evaluates operations immediately,
# without building graphs

#tf.compat.v1.disable_eager_execution()
#tf.compat.v1.enable_eager_execution()

print("tensorflow version", tf.__version__)
print("keras version", tf.keras.__version__)
print("Eager Execution Enabled:", tf.executing_eagerly())

# Get the number of replicas
strategy = tf.distribute.MirroredStrategy()
print("Number of replicas:", strategy.num_replicas_in_sync)

devices = tf.config.experimental.get_visible_devices()
print("Devices:", devices)
print(tf.config.experimental.list_logical_devices('GPU'))

print("GPU Available: ", tf.config.list_physical_devices('GPU'))
print("All Physical Devices", tf.config.list_physical_devices())
```

```
# Better performance with the tf.data API
# Reference: https://www.tensorflow.org/guide/data_performance
AUTOTUNE = tf.data.experimental.AUTOTUNE
```

```
tensorflow version 2.4.1
keras version 2.4.0
Eager Execution Enabled: True
INFO:tensorflow:Using MirroredStrategy with devices
('/job:localhost/replica:0/task:0/device:GPU:0',)
Number of replicas: 1
Devices: [PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
[LogicalDevice(name='/device:GPU:0', device_type='GPU')]
GPU Available:  [PhysicalDevice(name='/physical_device:GPU:0',
device_type='GPU')]
All Physical Devices [PhysicalDevice(name='/physical_device:CPU:0',
device_type='CPU'), PhysicalDevice(name='/physical_device:GPU:0',
device_type='GPU')]
```

[100]:
```
#USE_TENSORBOARD = IN_COLAB and True
USE_TENSORBOARD = IN_COLAB and False
```

[101]:
```
log_dir = working_dir / 'logs'
log_dir.mkdir(parents=True, exist_ok=True)

if USE_TENSORBOARD:
    %load_ext tensorboard
```

1.8.3 Data Preparation

Our modelling approach consists of dividing the North Carolina data into clusters based on the topic that was predicted to best fit each individual case. We then build a neural network multi-label classifier for each cluster, using an extension of the BERT transformer pre-trained on millions of legal language texts from the US and UK, LEGAL-BERT, to encode case text that will be used as our predictor. Our response variable is an array of binary indicators that reflect whether or not any particular case was cited for all cases that at any point were cited in the cluster.

Here we read in our saved `opinion_df` dataframe produced in the pre-processing above, configure our setup, and initialize state to North Carolina and sample size to the length of the dataframe (72486 cases).

[102]:
```
# CONFIG choose state
state = 'nc'
```

[103]:
```
# CONFIG choose dataframe by sample size
#sample = 38540
#sample = 7708
sample = 72486
```

[104]:
```python
# directory to store our processed data ('datasets')
dataset_dir = working_dir / 'datasets'
dataset_dir.mkdir(parents=True, exist_ok=True)

if opinions_df is None:
    # try to read serialized df from disk
    opinions_df = pd.read_feather(f"{dataset_dir}/{state}_df-{sample}.feather")
    opinions_df = opinions_df.set_index('id')
```

We examine the "top_topic" column of the dataframe, which encodes the topic that is assigned the highest predicted probability for each case. Note that while our Latent Dirichlet Allocation model predicted 14 topic clusters, only 11 unique topics are present in the "top_topic" column, suggesting that 3 topics are never predicted to be the most likely for North Carolina cases. We will thus have 11 total models. For each cluster, its respective model's name is saved and the number of cases assigned to this top cluster, total number of citations, and total number of cited cases are saved and printed below (clusters are 0 indexed, so actual cluster number is shown cluster number plus 1). As shown, clusters 1 (Criminal Offense), 3 (Contracts), and 8 (Miscellaneous) contain the most number of cases (each exceeding 10,000 cases), while cluster 5 (Municipal Public Administration) and 9 (Real Estate and Property) contain the least number of cases, at respectively 53 and 423. As expected, number of cases are fairly correlated with total number of citations and total number of cited cases.

[105]:
```python
# CONFIG choose df column with cluster labels
clusters_col = 'top_topic'
n_clusters = len(opinions_df[clusters_col].unique())
clusters = np.sort(opinions_df[clusters_col].unique())
n_clusters
```

[105]:
```
11
```

[106]:
```python
model_name_base = "legalbert_pretrained_seqsig_full_spacy02-topic"
def get_model_name(cluster_num, num_labels, full=False):
    # CONFIG set template used for model names
    # alter this to save new sets of weights/metrics/history
    # for different architectures etc rather than overwriting
    # existing ones
    model_name = f"{model_name_base}{cluster_num}_{num_labels}labels"
    if full:
        start_time = str(int(time.time()))
        model_name = model_name + f"_{start_time}"
    return model_name
print(get_model_name(2, 123))
print(get_model_name(2, 123, True))
```

```
legalbert_pretrained_seqsig_full_spacy02-topic2_123labels
legalbert_pretrained_seqsig_full_spacy02-topic2_123labels_1620704488
```

[107]:
```
clusters_counts = dict()
for group, rows in opinions_df.groupby(by=clusters_col):
    cited_ids = list(itertools.chain.from_iterable(rows.cites_to_from_graph.
 ↪values))
    clusters_counts.update({group: (get_model_name(group,␣
 ↪len(set(cited_ids))+1),len(rows), len(cited_ids), len(set(cited_ids)))})
    print(f'cluster {group}: {len(rows)} cases with {len(cited_ids)} citations␣
 ↪to {len(set(cited_ids))} cases')
```

```
cluster 0: 15392 cases with 40554 citations to 14812 cases
cluster 1: 1490 cases with 8056 citations to 3338 cases
cluster 2: 11014 cases with 80500 citations to 23031 cases
cluster 4: 53 cases with 171 citations to 132 cases
cluster 6: 6219 cases with 25149 citations to 11144 cases
cluster 7: 13675 cases with 126834 citations to 18868 cases
cluster 8: 423 cases with 2446 citations to 1075 cases
cluster 10: 4491 cases with 20607 citations to 7405 cases
cluster 11: 8763 cases with 63040 citations to 14724 cases
cluster 12: 2041 cases with 14117 citations to 6131 cases
cluster 13: 8925 cases with 59508 citations to 16961 cases
```

The below dataframe shows the raw predictor and response variables of interest, and is constructed as a subset of the `opinions_df` dataframe. The predictor is the text column, which will be encoded into a representation suitable for our model, the outcome is the citations column (`cites_to_from_graph`), which will be vectorized, and 11 separate models are separately fitted based on the categorical variable `top_topic`.

[108]:
```
# CONFIG choose which df columns to use as predictor and response
x_col = 'text'
y_col = 'cites_to_from_graph'
df = opinions_df[[x_col, clusters_col, y_col]]
df
```

[108]:
```
                                                    text  …
cites_to_from_graph
0      Ashe, J.\nThere is no error. The refusal to al…  …  [2092693,
8683631, 8683825, 8694640, 8696216, …
1      Walker, J.\nThis is an action to recover damag…  …  [2085500,
8650685, 8651154, 8660752, 8689057, …
2      Ashe, J.\nThis proceeding was begun before a m…  …
[1955411, 8683312, 8696587]
3      Smith, C. J.,\nafter stating the above. We thi…  …  [1955433,
8688747, 8694413, 8696782, 8696979, …
4      Clark, C. J.\nThis is an action by a married w…  …     [8652969,
8653502, 8655948, 8659681, 11273471]
…                                                  …  …
…
72481  *418BIGGS, Judge.\nThis appeal arises from the…  …  [867658, 4760257,
```

```
8520628, 8522147, 8526954, 9…
72482  McGEE, Judge.\nThe record in this case shows t…  …  [132074, 1155764,
1155801, 4149183, 4719700, 8…
72483  McGEE, Judge.\nThe undisputed facts in this ca…  …
[3739573, 4167670, 8301065, 11656152]
72484  JOHNSON, Judge.\nIn the fall of 1992, plaintif…  …  [4764316,
8525480, 8525814, 8527518, 8559762, …
72485  Barnhill, J.\nPetitioner stood indicted, charg…  …
[8651961]

[72486 rows x 3 columns]
```

For each cluster, we split the dataset into train, test, and validation subsets, with relative proportion of 80% - 10% - 10%. We utilize the MultiLabelBinarizer function from sklearn to convert a list of sets to the supported multilabel format, i.e. a samples x classes binary matrix indicating the presence of a class label, with binary labels. We additionally store the test sets in the list `test_sets` and the splits in the dictionary `splits_for_cluster` so as to preserve the particular split and avoid mismatch between observed and predicted values due to randomness in splitting.

[109]:
```
test_sets = list()
mlbs = dict()
splits_for_cluster = dict()

def _get_splits_for_cluster(c, test_size=0.2):
    print(f">>> cluster {c}")
    cdf = df[df[clusters_col]==c][[x_col, y_col]].dropna()
    print(f"\t{cdf.shape[0]} cases")
    labels = cdf[y_col].explode().unique()
    print(f"\t{len(labels)} labels")
    mlb = MultiLabelBinarizer(classes=labels)
    targets = mlb.fit_transform(cdf[y_col])
    mlbs.update({c: mlb})

    X_tr, X_te, y_tr, y_te = train_test_split(cdf[x_col], targets,␣
 ↪test_size=test_size)
    # split test further into test and val
    X_te, X_va, y_te, y_va = train_test_split(X_te, y_te, test_size=.5)
    splits = (X_tr, X_va, X_te, y_tr, y_va, y_te)
    splits_for_cluster.update({c: splits})
    y_te_citations = mlb.inverse_transform(y_te)
    test_sets.append(pd.DataFrame({'case_text': X_te.values,
                                   'citations': y_te_citations}).
 ↪assign(cluster=c))
    print("\tsplits shapes:", list(map(operator.attrgetter('shape'), splits)))
```

[110]:
```
for c in clusters:
    _get_splits_for_cluster(c)
```

```
test_sets = pd.concat(test_sets, axis=0)
test_sets
```

```
>>> cluster 0
        15392 cases
        14813 labels
        splits shapes: [(12313,), (1540,), (1539,), (12313, 14813), (1540,
14813), (1539, 14813)]
>>> cluster 1
        1490 cases
        3339 labels
        splits shapes: [(1192,), (149,), (149,), (1192, 3339), (149, 3339),
(149, 3339)]
>>> cluster 2
        11014 cases
        23032 labels
        splits shapes: [(8811,), (1102,), (1101,), (8811, 23032), (1102, 23032),
(1101, 23032)]
>>> cluster 4
        53 cases
        133 labels
        splits shapes: [(42,), (6,), (5,), (42, 133), (6, 133), (5, 133)]
>>> cluster 6
        6219 cases
        11145 labels
        splits shapes: [(4975,), (622,), (622,), (4975, 11145), (622, 11145),
(622, 11145)]
>>> cluster 7
        13675 cases
        18869 labels
        splits shapes: [(10940,), (1368,), (1367,), (10940, 18869), (1368,
18869), (1367, 18869)]
>>> cluster 8
        423 cases
        1076 labels
        splits shapes: [(338,), (43,), (42,), (338, 1076), (43, 1076), (42,
1076)]
>>> cluster 10
        4491 cases
        7406 labels
        splits shapes: [(3592,), (450,), (449,), (3592, 7406), (450, 7406),
(449, 7406)]
>>> cluster 11
        8763 cases
        14725 labels
        splits shapes: [(7010,), (877,), (876,), (7010, 14725), (877, 14725),
(876, 14725)]
```

```
>>> cluster 12
        2041 cases
        6132 labels
        splits shapes: [(1632,), (205,), (204,), (1632, 6132), (205, 6132),
(204, 6132)]
>>> cluster 13
        8925 cases
        16962 labels
        splits shapes: [(7140,), (893,), (892,), (7140, 16962), (893, 16962),
(892, 16962)]
```

[110]:

```
                                             case_text  … cluster
0    DeNNy, J.\nThe question for determination on t…  …       0
1    Ruffin. Judge.\nThere seems to he no doubt, th…  …       0
2    Olakk, C. J.\nTbe complaint alleges that after…  …       0
3    Clark, J.:\nWhen this cause was here before ()…  …       0
4    Brown, J.\nIn deraigning ber title, tbe plaint…  …       0
..                                                 …  …     …
887  Campbell, J.\n\n[1] Ordinarily, if a suitable …  …      13
888  Davis, J.,\n(after stating the case). We think…  …      13
889  ARNOLD, Judge.\nAppellants' property qualifies…  …      13
890  Rodman, J.\nThe Machinery Act, G.S. 105-271, e…  …      13
891  Clark, O. J'.\nTbe original petition, sec. 5, …  …      13

[7246 rows x 3 columns]
```

1.8.4 Model Building & Training

Here, we define a function to construct a model for LEGAL-BERT representations for each cluster. Next, we build a dataset through tokenizing the case texts and shuffling, batching, and prefetching the data, setting train and validation shuffle buffer sizes to the respective lengths of the train and validation datasets.

[111]:

```
from transformers import TFAutoModelForSequenceClassification

def build_pretrained_bert_for_cluster(c, num_labels):
    # Set the model name as
    model_name = get_model_name(c, num_labels, full=True)
    model = TFAutoModelForSequenceClassification.from_pretrained("nlpaueb/
 ↪legal-bert-base-uncased",
                                                                          ␣
 ↪num_labels=num_labels, name=model_name)
    return model
```

[112]:

```
from transformers import AutoTokenizer

def encode_inputs(X, max_length):
    tokenizer = AutoTokenizer.from_pretrained("nlpaueb/legal-bert-base-uncased")
```

```
    inputs = tokenizer(X.to_list(), padding="max_length",␣
↪truncation="longest_first",
                        max_length=max_length, return_token_type_ids=True,
                        return_attention_mask=True, return_tensors="tf")
    return inputs

def make_datasets(splits, batch_size=32, max_length=256):
    X_tr, X_va, X_te, y_tr, y_va, y_te = splits

    TRAIN_SHUFFLE_BUFFER_SIZE = len(X_tr)
    VALIDATION_SHUFFLE_BUFFER_SIZE = len(X_te)

    X_tr_processed = encode_inputs(X_tr, max_length)
    train_data = tf.data.Dataset.
↪from_tensor_slices(((X_tr_processed["input_ids"],
                                                                  ␣
↪X_tr_processed["token_type_ids"],
                                                                  ␣
↪X_tr_processed["attention_mask"]), y_tr))
    train_data = train_data.shuffle(buffer_size=TRAIN_SHUFFLE_BUFFER_SIZE)
    train_data = train_data.batch(batch_size)
    train_data = train_data.prefetch(buffer_size=AUTOTUNE)

    X_te_processed = encode_inputs(X_te, max_length)
    test_data = tf.data.Dataset.
↪from_tensor_slices(((X_te_processed["input_ids"],
                                                                 ␣
↪X_te_processed["token_type_ids"],
                                                                 ␣
↪X_te_processed["attention_mask"]), y_te))
    test_data = test_data.batch(batch_size)
    test_data = test_data.prefetch(buffer_size=AUTOTUNE)

    X_va_processed = encode_inputs(X_va, max_length)
    val_data = tf.data.Dataset.from_tensor_slices(((X_va_processed["input_ids"],
                                                                ␣
↪X_va_processed["token_type_ids"],
                                                                ␣
↪X_va_processed["attention_mask"]), y_va))
    val_data = val_data.batch(batch_size)
    val_data = val_data.prefetch(buffer_size=AUTOTUNE)
    return train_data, val_data, test_data
```

We now finetune the pretrained LEGAL-BERT model and train on the dataset for each cluster. Specifically, we add pooling and dropout layers with dropout rate equal to 0.2, as well as a Dense layer with number of units equal to number of cases ever cited in that cluster. Categorial crossentropy is a suitable choice of loss function given the shape of our response variable.

We also save the metrics produced by the models and define functions for plotting our results.

[113]:
```
def finetune_pretrained_bert_for_cluster(cluster, train_ds, val_ds, num_labels,␣
↪learning_rate=2e-5, epochs=5, max_length=256, train=False):
    transformer_model = build_pretrained_bert_for_cluster(cluster, num_labels)

    # adapted from https://towardsdatascience.com/
↪multi-label-multi-class-text-classification-with-bert-transformer-and-keras-c6355eccb63a
    bert_main = transformer_model.layers[0]

    inputs = {'input_ids': tf.keras.layers.Input(shape=(max_length,),
                                                 name='input_ids',␣
↪dtype='int32'),
              'token_type_ids': tf.keras.layers.Input(shape=(max_length,),
                                                 name='token_type_ids',␣
↪dtype='int32'),
              'attention_mask': tf.keras.layers.Input(shape=(max_length,),
                                                 name='attention_mask',␣
↪dtype='int32')}

    bert_model = bert_main(inputs)[1]
    dropout = tf.keras.layers.Dropout(.4, name='pooled_output')
    pooled_output = dropout(bert_model, training=False)

    citations = tf.keras.layers.Dense(units=num_labels, name='citations',␣
↪activation='sigmoid')(pooled_output)
    outputs = {'citations': citations}

    model = tf.keras.Model(inputs=inputs, outputs=outputs,␣
↪name=transformer_model.name)
    if not train:
        return model, None

    model.summary()
    optimizer = keras.optimizers.Adam(lr=learning_rate, epsilon=1e-08)
    loss = keras.losses.CategoricalCrossentropy(from_logits=True)

    metrics = ['accuracy', tf.keras.metrics.TopKCategoricalAccuracy(k=100)]

    model.compile(loss=loss, optimizer=optimizer, metrics=metrics)
    start_time = time.time()

    #es = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=2)
    #callbacks = [es]
    callbacks = []

    if USE_TENSORBOARD:
```

```
        tb = tf.keras.callbacks.TensorBoard(log_dir=f"{log_dir}/{model.name}",
                                            profile_batch=20, histogram_freq=1)
        callbacks.append(tb)
    history = model.fit(train_ds, validation_data=val_ds, epochs=epochs,
                        verbose=1, callbacks=callbacks)
    execution_time = (time.time() - start_time)/60.0
    print(f"Training execution time: {execution_time:.2f} mins")
    return model, history
```

[114]:
```
model_dir = working_dir / 'models'
model_dir.mkdir(parents=True, exist_ok=True)
```

[115]:
```
class JsonEncoder(json.JSONEncoder):
    """
    custom JsonEncoder (from cs109b HW6 hwutils)
    used to convert model metrics when saving
    a `metrics.json` file for each trained model
    """
    def default(self, obj):
        if isinstance(obj, np.integer):
            return int(obj)
        elif isinstance(obj, np.floating):
            return float(obj)
        elif isinstance(obj, decimal.Decimal):
            return float(obj)
        elif isinstance(obj, np.ndarray):
            return obj.tolist()
        else:
            return super(JsonEncoder, self).default(obj)
```

[116]:
```
def plot_loss(model_history, out_file=None):
    """
    This helper function plots the NN model accuracy and loss.
    Arguments:
        model_history: the model history return from fit()
        out_file: the (optional) path to save the image file to.
    """
    fig, ax = plt.subplots(1, 2, figsize=(12, 4))

    history = model_history
    ax[0].plot(history.history['accuracy'])
    ax[0].plot(history.history['val_accuracy'])
    ax[0].set_title('model accuracy')
    ax[0].set_ylabel('accuracy')
    ax[0].set_xlabel('epoch')
    ax[0].legend(['train', 'validation'], loc='upper left')
```

```
    # summarize history for loss
    ax[1].plot(history.history['loss'])
    ax[1].plot(history.history['val_loss'])
    ax[1].set_title('model loss')
    ax[1].set_ylabel('loss')
    ax[1].set_xlabel('epoch')
    ax[1].legend(['train', 'validation'], loc='upper left')
    plt.show()

    if out_file:
        plt.savefig(out_file)
```

[117]:
```
def plot_recall_ratio(y_true, y_pred, out_file=None, ax=None, title=''):
    """
    This helper function plots a histogram of the ratio between at what rank
    all cited cases are accounted for (i.e., there is perfect recall) in the
    test predictions and the number of cited cases.
    Arguments:
        y_true: the true test response
        y_pred: the predicted test response
        out_file: the (optional) path to save the image file to.
    """
    show_plot = False
    if not ax:
        fig, ax = plt.subplots()
        show_plot = True

    recall_ratios = []
    for j in range(len(y_true)):
      cited = [i for i, case in enumerate(y_true[j]) if case == 1]
      num_cited = len(cited)
      predicted_citations_ranked = np.flip(np.argsort(y_pred[j]))
      try:
        # Find at what rank (using 1 indexing) all cited cases are accounted for
        index_perfect_recall = np.max([list(predicted_citations_ranked).
↪index(c) for c in cited]) + 1
        recall_ratios.append(index_perfect_recall / num_cited)
      except ValueError:
        pass

    ax.hist(recall_ratios, bins=30)
    ax.set_title(title)
    ax.set_ylabel('frequency')
    ax.set_xlabel('recall ratio')
    if show_plot:
        plt.show()
```

```
    if out_file:
        plt.savefig(out_file)
    return ax
```

[118]:
```
from tensorflow.python.framework import ops

def train_score_cluster(cluster, batch_size=32, max_length=256,␣
↪learning_rate=2e-5, epochs=5):
    K.clear_session()
    ops.reset_default_graph()
    splits = splits_for_cluster[cluster]
    num_labels = splits[3].shape[1]

    # model weights/metrics/histories are saved with full name including␣
↪timestamp
    # to distinguish each training run
    #legalbert_pretrained_cluster0_20489labels_1620347831

    # we use the name without timestamp to check for weights etc
    # and load the most recent if any exist
    #legalbert_pretrained_cluster0_20489labels*
    model_name = get_model_name(cluster, num_labels)
    print("\n", model_name)

    # check for and load saved weights
    saves = glob(f"{model_dir}/{model_name}*/model.h5")
    if len(saves) > 0:
        model = tf.keras.models.load_model(saves[0])
        print(f"Loaded saved model from {model_dir}.")
    else:
        print(f"No saved model in {model_dir}.")
        train_ds, val_ds, test_ds = make_datasets(splits, batch_size=batch_size,
                                                  max_length=max_length)
        model, history = finetune_pretrained_bert_for_cluster(cluster,␣
↪train_ds, val_ds,
                                                               num_labels,␣
↪max_length=max_length,
                                                               epochs=epochs,␣
↪train=True)
        model_save_dir = working_dir / 'models' / model.name
        model_save_dir.mkdir(parents=True, exist_ok=True)

        # save full name with timestamp
        model.save(os.path.join(f"{model_dir}/{model.name}", "model.h5"))
        plot_loss(model_history=history)
        if USE_TENSORBOARD:
            save_embeddings_for_projector(model)
```

```
        # save model history
        with open(os.path.join(f"{model_dir}/{model.name}", f"train_history.
↪json"), "w") as f:
            f.write(json.dumps(history.history, cls=JsonEncoder))

    saved_history = glob(f"{model_dir}/{model_name}*/train_history.json")
    if len(saved_history) > 0:
        history = json.load(open(saved_history[0]))

    saved_metrics = glob(f"{model_dir}/{model_name}*/metrics.json")
    if len(saved_metrics) > 0:
        metrics = json.load(open(saved_metrics[0]))
    else:
        train_ds, val_ds, test_ds = make_datasets(splits, batch_size=batch_size,
                                                  max_length=max_length)
        model_size = os.stat(glob(f"{model_dir}/{model_name}*/model.h5")[0]).
↪st_size

        # evaluate on test data
        evaluation_results = model.evaluate(test_ds)
        print(evaluation_results)
        y_pred = model.predict(test_ds)['citations']
        #plot_recall_ratio(splits[5], y_pred)

        trainable_parameters = count_params(model.trainable_weights)
        non_trainable_parameters = count_params(model.non_trainable_weights)

        # save model metrics
        metrics = {
            "trainable_parameters": trainable_parameters,
            "non_trainable_parameters": non_trainable_parameters,
            "loss": evaluation_results[0],
            "accuracy": evaluation_results[1],
            "topk_categorical_accuracy": evaluation_results[2],
            "ndcg_score": ndcg_score(splits[5], y_pred),
            "model_size": model_size,
            "learning_rate": learning_rate,
            "epochs": epochs,
            "name": model.name,
            "cluster": cluster
        }
        with open(os.path.join(f"{model_dir}/{model.name}", "metrics.json"),␣
↪"w") as f:
            f.write(json.dumps(metrics, cls=JsonEncoder))
    print(metrics)
    return model, history, metrics, splits
```

[119]:
```
if USE_TENSORBOARD:
    %tensorboard --logdir "/content/gdrive/MyDrive/cs109b/law_citations/logs/"
```

[120]:
```
assert sorted(splits_for_cluster.keys()) == sorted(clusters_counts.keys()) ==␣
↪sorted(clusters)
print("clusters:", clusters)
```

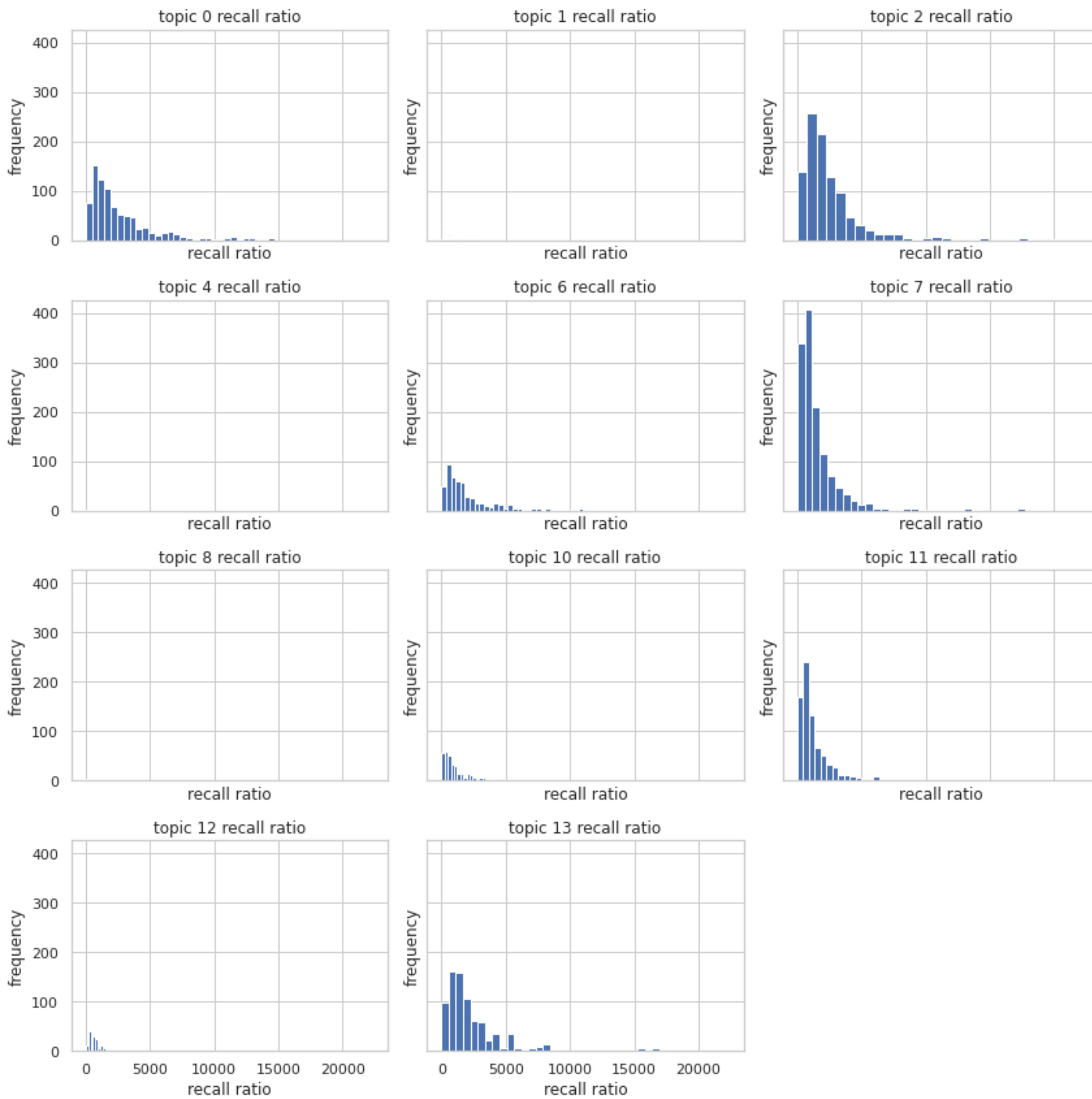
```
clusters: [ 0  1  2  4  6  7  8 10 11 12 13]
```

[121]:
```
%%time
def train_missing():
    for n in clusters:
        model_name = clusters_counts[n][0]
        saves = glob(f"{model_dir}/{model_name}*/model.h5")
        if len(saves) > 0:
            print(f"found saved model for '{model_name}'")
            continue
        else:
            training_result = train_score_cluster(n)
train_missing()
```

```
found saved model for
'legalbert_pretrained_seqsig_full_spacy02-topic0_14813labels'
found saved model for
'legalbert_pretrained_seqsig_full_spacy02-topic1_3339labels'
found saved model for
'legalbert_pretrained_seqsig_full_spacy02-topic2_23032labels'
found saved model for
'legalbert_pretrained_seqsig_full_spacy02-topic4_133labels'
found saved model for
'legalbert_pretrained_seqsig_full_spacy02-topic6_11145labels'
found saved model for
'legalbert_pretrained_seqsig_full_spacy02-topic7_18869labels'
found saved model for
'legalbert_pretrained_seqsig_full_spacy02-topic8_1076labels'
found saved model for
'legalbert_pretrained_seqsig_full_spacy02-topic10_7406labels'
found saved model for
'legalbert_pretrained_seqsig_full_spacy02-topic11_14725labels'
found saved model for
'legalbert_pretrained_seqsig_full_spacy02-topic12_6132labels'
found saved model for
'legalbert_pretrained_seqsig_full_spacy02-topic13_16962labels'
CPU times: user 8.16 ms, sys: 6.56 ms, total: 14.7 ms
Wall time: 53.2 ms
```

1.9 Results

[ ]:
```
%%time
def plot_recall_ratio_charts(print_evaluate=False):
    fig = plt.figure(figsize=(12, 12))
    axes = fig.subplots(nrows=4, ncols=3, sharex=True, sharey=True).ravel()
    for i, ax in enumerate(axes):
        if i < n_clusters:
            n = clusters[i]
            model_name = clusters_counts[n][0]
            saves = glob(f"{model_dir}/{model_name}*/model.h5")
            if len(saves) > 0:
                model = tf.keras.models.load_model(saves[0])
                splits = splits_for_cluster[n]
                train_ds, val_ds, test_ds = make_datasets(splits, batch_size=5,
                                                          max_length=256)
                if print_evaluate:
                    evaluation_results = model.evaluate(test_ds)
                    print(evaluation_results)
                y_pred = model.predict(test_ds)['citations']
                plot_recall_ratio(splits[5], y_pred, ax=ax, title=f"topic {n}␣
 ↪recall ratio")
        else:
            ax.set_axis_off()
    fig.tight_layout()
    plt.show()
plot_recall_ratio_charts()
```




```
CPU times: user 15min 57s, sys: 56.3 s, total: 16min 53s
Wall time: 15min 56s
```

[123]:
```python
#model, history, metrics, splits = train_score_cluster(0)
#training_results = [t[0].name for t in map(train_score_cluster, clusters)]
```

[124]:
```python
def models_metrics():
    models_metrics_list = glob(f"{model_dir}/{model_name_base}*/metrics.json")
    all_models_metrics = []
    for mm_file in models_metrics_list:
        with open(mm_file) as json_file:
            model_metrics = json.load(json_file)
            all_models_metrics.append(model_metrics)
```

```
    # Load metrics to dataframe
    view_metrics = pd.DataFrame(data=all_models_metrics)

    # Format columns
    view_metrics['accuracy'] = view_metrics['accuracy']*100
    view_metrics['accuracy'] = view_metrics['accuracy'].map('{:,.2f}%'.format)

    view_metrics['trainable_parameters'] = view_metrics['trainable_parameters'].
↪map('{:,.0f}'.format)
    view_metrics['loss'] = view_metrics['loss'].map('{:,.2f}'.format)
    view_metrics['ndcg_score'] = view_metrics['ndcg_score'].map('{:,.2f}'.
↪format).astype('float')
    view_metrics['topk_categorical_accuracy'] =␣
↪view_metrics['topk_categorical_accuracy'].map('{:,.2f}'.format).
↪astype('float')
    view_metrics['model_size'] = view_metrics['model_size']/1000000
    view_metrics['model_size'] = view_metrics['model_size'].map('{:,.0f} MB'.
↪format)

    # Filter columns
    view_metrics =␣
↪view_metrics[["cluster","trainable_parameters","loss","accuracy","topk_categorical_accuracy

    view_metrics = view_metrics.sort_values(by=["cluster"],ascending=False)
    return view_metrics
```

We display relevant metrics for each model, including Loss, Accuracy, TopK Categorical Accuracy, Normalized Discounted Cumulative Gain score. (Note that the index number is *not* the actual cluster number; cluster numbers are displayed in the `name` column as `topic[X]`.

In addition to accuracy, we chose to additionally monitor the TopK categorical accuracy as well as normalized discounted cumulative gain, because accuracy may not capture the entire narrative and may give a partial representation in this high-dimensional context, for the following reason. Since the universe of possible cited cases is large, obtaining a high accuracy requires an exact matching between predicted citation (Yes/No) and actual citation for all citation; therefore, accuracy is relatively stringent and inflexible metric. In comparison, NDCG score (similar to recall) allows greater flexibility in missing certain citations; instead of examining the precise correctness, it looks at the proportion of top n correct citations that are cited, and therefore is more lenient and fitting for our problem.

We observe that Topic 8 has the highest accuracy at 11.90%, far outstripping other topics. Furthermore, it also has the highest NDCG score at 0.28.

Despite the positive correlation between accuracy and NDCG score/TopK categorical, the correspondence is far from tight. For instance, accuracies for topics 4 and 12 are extremely low at 0.00%; however, TopK categorical accuracy is counterintuitively the highest for topic 4 (at a perfect 1.00) and similarly far from negligible for topic 12 (at 0.14). We hypothesize that this mismatch in accuracy metrics may be due to a “grouping” effect in citations; namely, judge opinions tend to cite a prominent group of landmark cases that are highly related. For instance, in the realm of racial discrimination, Plessy v. Ferguson, Dred Scott v. Sandford, and Brown v. Board of Education are landmark cases that are likely cited together when reviewing legal precedents.

Thus, in inferring all citations for a case, the prediction will likely be highly accurate for the group of landmark cases (as they are highly prevalent and co-occur together), but less precise for remaining cases that are less notable or historically significant. This disproportionality may explain the disparity between high accuracy scores and unimpressive NDCG scores, vice versa.

[125]:
```
display(models_metrics())
```

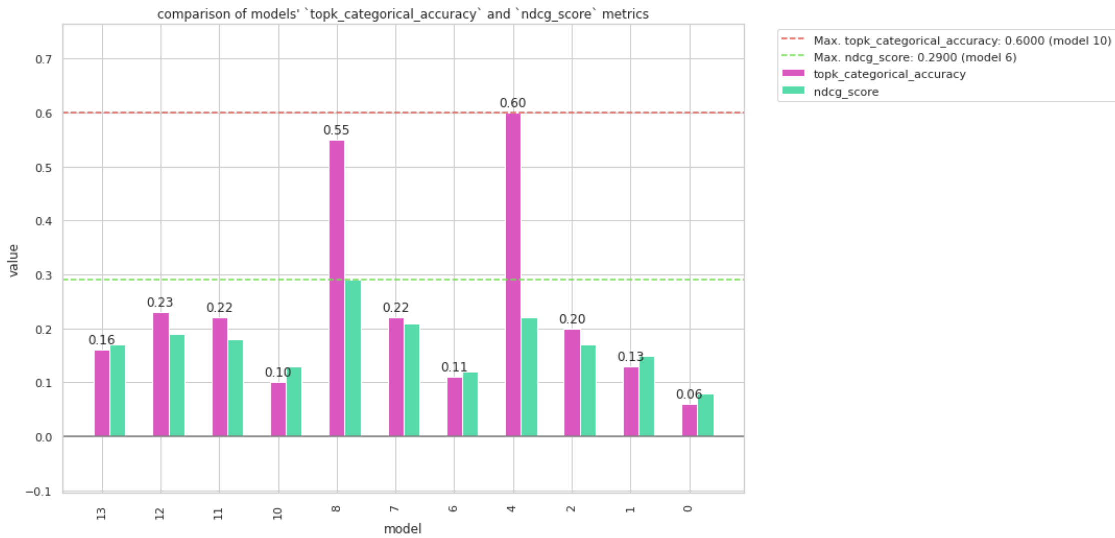
```
    cluster  …                                                name
10       13  …  legalbert_pretrained_seqsig_full_spacy02-topic…
9        12  …  legalbert_pretrained_seqsig_full_spacy02-topic…
8        11  …  legalbert_pretrained_seqsig_full_spacy02-topic…
7        10  …  legalbert_pretrained_seqsig_full_spacy02-topic…
6         8  …  legalbert_pretrained_seqsig_full_spacy02-topic…
5         7  …  legalbert_pretrained_seqsig_full_spacy02-topic…
4         6  …  legalbert_pretrained_seqsig_full_spacy02-topic…
3         4  …  legalbert_pretrained_seqsig_full_spacy02-topic…
2         2  …  legalbert_pretrained_seqsig_full_spacy02-topic…
1         1  …  legalbert_pretrained_seqsig_full_spacy02-topic…
0         0  …  legalbert_pretrained_seqsig_full_spacy02-topic…

[11 rows x 10 columns]
```

[126]:
```
colors = sns.color_palette("hls", 7)
def compare_models_metrics(scores, metrics=['topk_categorical_accuracy',␣
↪'ndcg_score']):
    """
    Plot summary table and bar chart of given model metrics

    :param scores: dataframe with a 'cluster' column and given metrics columns
    :returns: None
    """
    fig, ax = plt.subplots()

    # x locations for bars
    ind = np.arange(len(scores['cluster']))
    # width of bars
    width = 0.27

    ax.set_xlabel("model")
    ax.set_xticks(range(len(scores['cluster']))) # workaround to avoid warning␣
↪https://github.com/pandas-dev/pandas/issues/35684
    ax.set_xticklabels(scores['cluster'], rotation=90)
    ax.set_ylabel('value')
    score_range = min(scores[metrics[0]]) + max(scores[metrics[0]])
```

```
    print(score_range)
    ax.set_ylim(min(scores[metrics[0]])-np.abs(score_range/4),␣
↪max(scores[metrics[0]])+np.abs(score_range/4))
    ax.set_title(f"comparison of models' `{metrics[0]}` and `{metrics[1]}`␣
↪metrics")
    plt.axhline(0, c='grey')
    plt.axhline(max(scores[metrics[0]]), c=colors[0], ls='--',
                label=f"Max. {metrics[0]}: {max(scores[metrics[0]]):.4f} (model␣
↪{scores['cluster'][np.argmax(scores[metrics[0]])]})")
    bars_metric_0 = plt.bar(ind, scores[metrics[0]], width, color=colors[6],␣
↪label=metrics[0])
    plt.axhline(max(scores[metrics[1]]), c=colors[2], ls='--',
                label=f"Max. {metrics[1]}: {max(scores[metrics[1]]):.4f} (model␣
↪{scores['cluster'][np.argmax(scores[metrics[1]])]})")
    bars_metric_1 = plt.bar(ind+width, scores[metrics[1]], width,␣
↪color=colors[3], label=metrics[1])

    def autolabel(rects):
        """Attach a text label above each bar in *rects*, displaying its height.
↪"""
        # https://matplotlib.org/3.2.1/gallery/lines_bars_and_markers/barchart.
↪html
        for rect in rects:
            height = rect.get_height()
            ax.annotate(f'{height:.2f}',
                        xy=(rect.get_x() + rect.get_width() / 2, height),
                        xytext=(0, 3),  # 3 points vertical offset
                        textcoords="offset points",
                        ha='center', va='bottom')

    autolabel(bars_metric_0)
    # put legend to the right of the plot (thx to https://stackoverflow.com/a/
↪43439132)
    plt.legend(bbox_to_anchor=(1.04,1), loc="upper left")

    plt.show()
```

[127]:
```
compare_models_metrics(models_metrics())
```

```
0.6599999999999999
```



[128]:
```
tf.get_logger().setLevel('INFO')
```

[129]:
```
def encode_case(case):
    tokenizer = AutoTokenizer.from_pretrained("nlpaueb/legal-bert-base-uncased")
    inputs = tokenizer(case, padding="max_length", truncation="longest_first",
                       max_length=256, return_token_type_ids=True,
                       return_attention_mask=True, return_tensors="tf")
    return(inputs)

def predict_citations_for_case_text(case_texts, lda_model, vectorizer,␣
↪threshold=0.5):
    # tokenize and vectorize case_texts
    doc_vecs = vectorizer.transform(case_texts)
    # get document-topic probabilities
    lda_output = lda_model.transform(doc_vecs)
    # find topic with highest probability for each doc
    top_topics = np.argmax(lda_output, axis=1)

    # join case_texts with topic predictions
    docs_with_topics = pd.DataFrame({'case_text': case_texts,
                                     'topic_pred': top_topics})
    results = []
    # group by topic and iterate (so we can load each topic's corresponding
    # language model one and only one time)
    for topic, topic_case_texts in docs_with_topics.groupby('topic_pred'):
        # get model_name and look for saved model runs
        model_name = clusters_counts[topic][0]
        print("\n", model_name)
        saves = glob(f"{model_dir}/{model_name}*/model.h5")
```

```
        if len(saves) > 0:
            model = tf.keras.models.load_model(saves[0])
            print(f"Loaded saved model from {model_dir}.")
            for idx, case_text in topic_case_texts.case_text.iteritems():
                # tokenize and encode case_text using Legal BERT tokenizer
                encoded = encode_case(case_text)
                # inference
                predicted = model.predict((encoded["input_ids"],
                                           encoded["token_type_ids"],
                                           encoded["attention_mask"]),
                                          np.zeros(len(encoded["input_ids"])))
                # get predicted probabilities for each label
                # and round to 0 or 1 based on given threshold
                predicted_probs = predicted['citations']
                predictions = np.where(predicted_probs > threshold, 1, 0)

                # convert array of 0s and 1s back to citation ids
                # using multilabelbinarizer
                citation_ids = mlbs[topic].inverse_transform(predictions)
                result = {'id': idx, 'case_text': case_text, 'citations_pred':␣
↪citation_ids,
                          'num_citations_pred': np.sum(predictions),␣
↪'topic_pred': topic}
                results.append(result)
    results_df = pd.DataFrame.from_records(results).set_index('id')
    return results_df
```

[130]:
```
test_cases_sample = test_sets.sample(n=5)
test_cases_sample['num_citations'] = test_cases_sample.citations.apply(len)
display(test_cases_sample)

results = predict_citations_for_case_text(test_cases_sample.case_text,␣
↪vectorizer=vectorizer,
                                          lda_model=lda_model, threshold=.99)
pd.merge(test_cases_sample, results)
```

```
                                              case_text  … num_citations
816  PHILLIPS, Judge.\nIn entering judgment for the…  …              1
99   EAGLES, Judge.\nRespondents argue that the tri…  …              5
189  ClaeK, C. J.\nThe evidence, for the plaintiff,…  …              0
164  HUNTER, Judge.\nTerry P. Smith, individually a…  …             14
196  Clarkson, J.\nThis'case has been here twice be…  …             10

[5 rows x 4 columns]


 legalbert_pretrained_seqsig_full_spacy02-topic1_3339labels
Loaded saved model from /content/gdrive/MyDrive/cs109b/law_citations/models.
```

```
 legalbert_pretrained_seqsig_full_spacy02-topic2_23032labels
Loaded saved model from /content/gdrive/MyDrive/cs109b/law_citations/models.

 legalbert_pretrained_seqsig_full_spacy02-topic7_18869labels
Loaded saved model from /content/gdrive/MyDrive/cs109b/law_citations/models.
```

[130]:

```
                                          case_text  … topic_pred
0  PHILLIPS, Judge.\nIn entering judgment for the…  …          2
1  EAGLES, Judge.\nRespondents argue that the tri…  …          1
2  ClaeK, C. J.\nThe evidence, for the plaintiff,…  …          7
3  HUNTER, Judge.\nTerry P. Smith, individually a…  …          2
4  Clarkson, J.\nThis'case has been here twice be…  …          2

 [5 rows x 7 columns]
```

1.10 Model Strengths and Limitations

Each model showed more promising results than the naive approach, achieving a higher NDCG score. Further, our modelling approach worked quite well with our chosen formatting of the predictor and response variables. Before separating the data into multiple clusters, our attempts to construct an array for each observation indicating whether or not any given case is cited proved intractable. Creating an array with a length in the tens of thousands for each of the tens of thousands of observations required processing power and storage that we did not have access to. While we initially considered significantly reducing the scope of the study, dividing the task into multiple smaller ones made the modelling possible and took advantage of all of the available data.

Our most relevant limitations were a lack of time and processing ability. Each piece of this process took a significant amount of time to run. The most obvious example is the LDA modelling, which took 7 hours to produce its output. But this shortage of time also impacted how well-tuned each of our 11 classification models could be. We were unable to thoroughly experiment with the depth of the neural networks or the number of epochs they were run, for example. We were also unable to make use of other features in the given data, such as the year of the case, when making predictions.

1.11 Next Steps

The most natural next step for our study would be to expand beyond North Carolina's cases. A simple extension of our approach that would work for multiple states would be to designate each state as its own cluster and then subcluster within them. This would account for the fact that cases within any given state are most likely to cite cases from the same state.

Following the paper Inferring Mechanisms for Global Constitutional Progress by Rutherford et al. (2017), we could consider the hierarchical dependencies within the sequence of rulings to see if the adoption of a certain legal stance co-occurs, contributes, or likely results from the adoption of another legal stance. More concretely, judicial decisions on higher-level umbrella rights likely preceded and paved the way for judicial decisions about specific rights. For instance, under the umbrella of right to work laws, the broad legal principle was likely delineated in an earlier case, followed by later cases establishing specific rights like rights to safe working conditions, right to equal payment for work, limits to child employment, and right to form trade unions. We could investigate the hierarchical and sequential process of citing and refining previous rulings, producing a sequence of cases with increasingly granular specifications. The conceptual framework resembles a tree diagram, whereby higher level domains (in earlier cases) continuously branch off into intermediate/lower level topics (in later cases), and nodes represent cases.

Next, the geographical and temporal dimensions of the spread of case citations may also exhibit intriguing patterns. Regions that are more similar in political affiliation and demographic composition, and more proximate in terms of geographical and social distance are likely to witness higher rates of convergence in judicial attitudes as well as faster spread, as measured by the number and date of case citations. For instance, more liberal and progresive states adopted right-to-work laws for minority groups in faster succession than more conservative states, which should be manifest in the trail of case citations. We could track the trails of case citations, as was done for patent citations in Jaffe and Trajtenberg (2000), and investigate questions like: 1) What differentiates cases that are frequently cited/ have a longer citation trail from cases that are not? (We could potentially perform NLP to investigate whether the appearance of certain keywords within the opinion correlates with its degree of influence) 2) Are there any recurrent geographic, demographic, or social patterns to citation trails? In other words, do there appear to be spillover effects where one state's judicial decisions tend to influence those of its neighbors? What about states that are "socially" close, as measured by the Facebook Social Connectedness Index? Do the same spillover effects apply to states that are both physically and socially distant but politically similar? 3) How long on average does it take for the judicial system to "internalize" and "popularize" a landmark case, as measured by when rulings begin to cite it en masse.

We could further divide the universe of cases between citations of referenced cases or citations of overturned cases. From common intuition, we expect 1) higher level federal and appellate courts are more likely to cite rulings by lower level courts for overturning purposes, and 2) lower level district courts are more likely to cite higher level court rulings for reference purposes. Meanwhile, it's also quite likely for the Supreme Court to overturn its own historical decisions (especially since no higher court exists to overturn their decisions), due to evolution of the political landscape and social norms. For instance, in the domain of racial discrimination, the Supreme Court ruled in Dred Scott v. Sandford that African Americans, free or slave, were not American citizens and could not sue in federal courts, and established in Plessy v. Ferguson the notorious "separate but equal" principle, in the 19th century. Amidst the civil rights movements in the mid-20th century, the Supreme Court, ruling in Brown v. Board of Education, overturned the "separate but equal" precedent and initiated a wave of later court cases that enhanced racial equality.

Another possible next step would be to focus solely on citations made to federally decided cases. This is especially intriguing, since this would not require dividing the data set by state. Thus, we would be able to see how federal precedents are used across the country and how pairs of states compare in the specific ones their courts most frequently cite.

[130]:

```

```