CAPSTONE PROJECT: SIGAL TEAM

Website Crawler & Text Classification

Ash Chetty, Wei Zhu, Po-Chen Su, Harsh Shah, Haochen Wang | University of the Pacific

Table of Contents

1. Introduction

The primary objectives of this project are to develop a website crawler and to create a text classification model. The aim of the Capstone Project is, through the utilization of various python packages, to construct a website crawler that is capable of extracting pertinent pages from two distinct websites, as determined by QQ Tech. However, when faced with the task of extracting more than a thousand pages, discerning the relevance of each document can prove to be a difficult task. To overcome this obstacle, various text classification machine learning models will be implemented. The end goal is to decide on a model (or more) which efficiently distinguishes between relevant and non-relevant files. The models that were put to the test are: Logistic Regression, Random Forest, XGBoost, KNN, RNN-GRU, and LSTM, and their results will be explained towards the end of this report.

2. Website Crawler

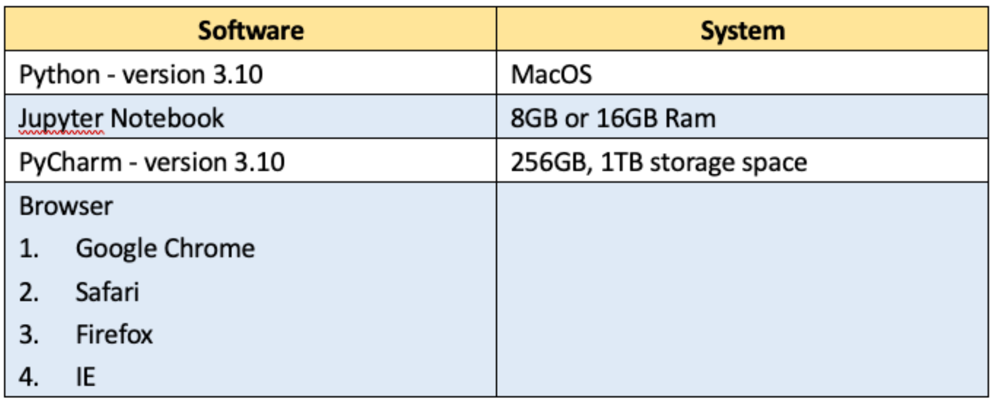
In this website crawler part, we have two websites to crawl: American Legal Publishing and Code of Federal Regulations. Our general approach is to obtain all the links and titles on the websites, and then use our keywords to check these titles. We will save the legal content in HTML format for the titles that contain the keyword and write this information into the metadata.

For the American Legal Publishing website, there is a large amount of data, with over 10,000 plus entries for different states and localities. Therefore, we will start with one state and gradually test to improve our program for other states.

For the Code of Federal Regulations website, there is much less data, and we can crawl entire pages directly without needing to set too many conditions.

3. Environment

Software & System



Software	System
Python - version 3.10	MacOS
Jupyter Notebook	8GB or 16GB Ram
PyCharm - version 3.10	256GB, 1TB storage space
Browser 1. Google Chrome 2. Safari 3. Firefox 4. IE	

The Python and PyCharm versions we are using are 3.10, which was chosen at the beginning. We will test some code on Jupyter Notebook to confirm our assumptions and results. Our main development environment is PyCharm. We use Google Chrome, Safari, Firefox, and IE as browsers. Our system is MacOS with 8GB or 16GB of RAM and 256GB or 1TB of storage capacity.

Python Package

These are the Python packages we are using, and we consider os, requests, Beautiful Soup, and Selenium to be the most important packages. This is because we mainly read content from web pages and access the necessary content.

copy	Provides functions for creating and manipulating copies of objects.
csv	Provides classes for reading and writing comma-separated values (CSV) files.
os	Provides a way of interacting with the operating system.
re	Provides support for regular expressions (regex).
shutil	Provides a higher-level interface for file operations.
requests	Allows sending HTTP/1.1 requests using Python.
BeautifulSoup	Parses HTML and XML documents.
selenium	Provides a way to automate web browsers.
pandas	Provides data structures and tools for data analysis.
time	Provides various time-related functions.
datetime	Provides classes for working with dates and times.
deque	Provides a data structure for a double-ended queue.
logging	Provides a way to log messages from your Python application.
RotatingFileHandler	A handler for logging to a file that automatically rotates the log file when it reaches a certain size.

4. Websites Structure Introduction

American Legal Publishing	**Differences**
● State(s) (home page)	Many States.
○ Locality(s) (cities)	Each state has one or many localities.
■ Titles / Chapters (overview page)	Crawl on the overview page.
● Subchapters / Articles / Parts	
○ **<u>Leaf page (contents)</u>**	
Code Of Federal Regulations	Many Titles.
● Title(s) (home page)	Crawl on each Title.
○ Chapters	
■ Subchapters / Parts	
● **<u>Leaf page (contents)</u>**	

On the American Legal Publishing website, the homepage has many states, and each state has one or more localities. The overview page on the locality page is where the chapters and content are stored.

On the Code of Federal Regulations website, the homepage has many titles, and each title contains chapters.

Both websites have a common feature: there are many small chapters within each large chapter, and the content we need is written on the last page of the large chapter.

5. Map out the Targeting Result

We have set many conditions in our program, and the process is quite complex, making it difficult to explain in detail. Therefore, this page is simply intended to describe the conditions we have set and how we extract the content. All contents are written in Leaf page. If a Chapter Title is detected with any keywords, all content within that Chapter will be saved, and the same method will be applied to the subchapter and leaf page.

All contents are written in Leaf page.

- If a **Chapter Title** is detected with any keywords, all content within that Chapter will be saved.
- If a **Subchapter Title** is detected with any keywords, all content within that Subchapter will be saved.
- If a **Leaf page Title** is detected with any keywords, the content of **that Leaf page** will be saved.

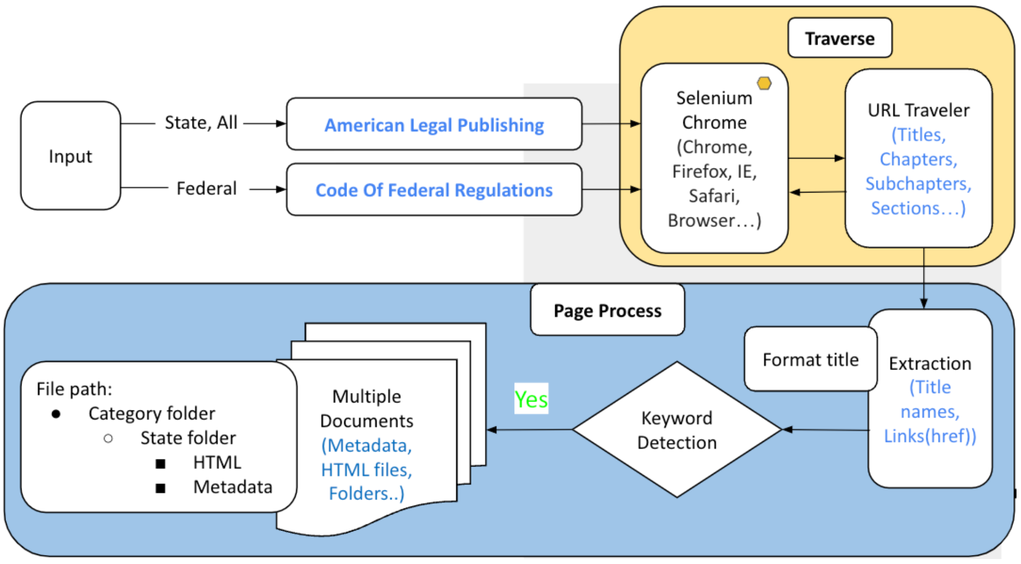
The information will be stored in different **Category Folders** based on the different **Keyword Categories**. (Category / State / HTMLs)

Keyword Category & Keywords

These are our keywords and keyword categories, which were provided to us by double Q Tech. We use this data to perform our detection.

Category	Keywords
Residential Lease & Rental	rent; lease; leasing; tenant; landlord; housing; real property; real properties; real estate; residence; residential; premises; occupancy; eviction
Employment Documents	employ; labor; worker; working; talent; wage; salary; compensation; payroll; benefits; at will; workplace; job; occupation; profession; non disclosure; non competition; vacation time; time off; resign; termination; layoff
Consumer Agreements	consumer; end user; business; merchant; seller; sale; goods; commodity; commodities; retail; commercial; commerce; trade; price; pricing; loan; debt; credit; obligation; liability; indemnity; release; waiver

6. Process Flow Overview


Input
State, All
Federal
American Legal Publishing
Code Of Federal Regulations
Traverse
Selenium Chrome (Chrome, Firefox, IE, Safari, Browser…)
URL Traveler (Titles, Chapters, Subchapters, Sections…)
Page Process
Format title
Extraction (Title names, Links(href))
Keyword Detection
Yes
Multiple Documents (Metadata, HTML files, Folders..)
File path:
Category folder
State folder
HTML
Metadata

When you enter a specific state or "all", you will enter American Legal Publishing. When you enter "Federal", you will enter Code of Federal Regulations.

In the "Traverse" section, we use Selenium and select these web drivers. These are the browsers we have tested and selected. We will explain why we conducted these tests later.

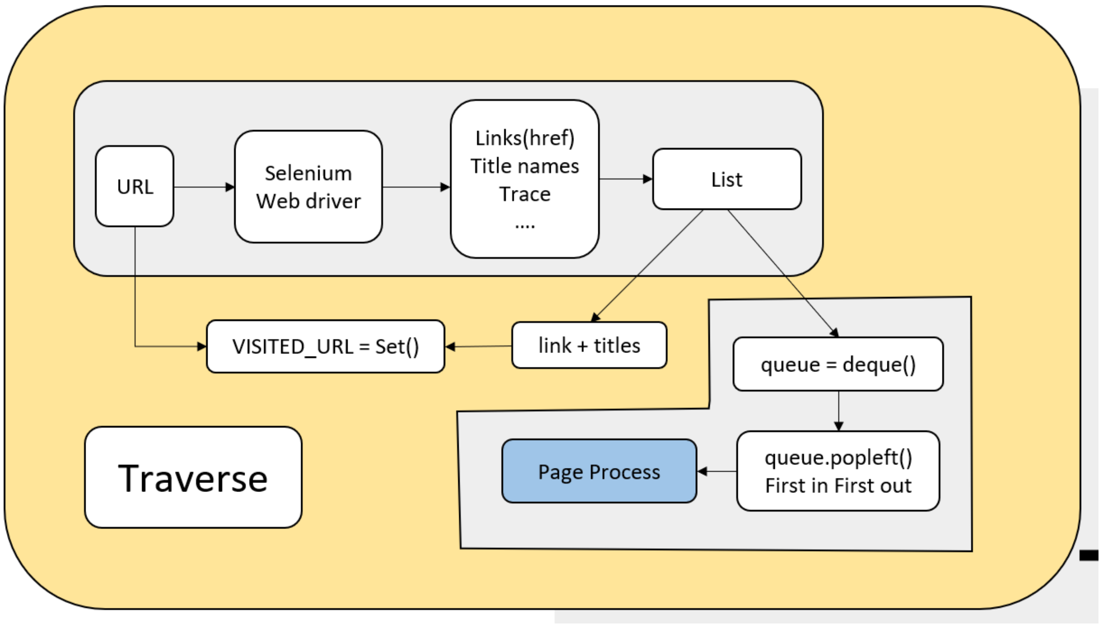
In the "URL Traveler" section, it retrieves all chapters.

In the “process page” section, we format the titles, detect keywords, and save the results to a specific file path.

The table below shows the comparison of the time required to generate files with different web driver. For the American Legal Publishing website, the time taken ranges between 12 to 13 hours. For the Code of Federal Regulations, it takes between 7 to 8 hours.

Website	State & Locality	System	Browser	Mode	Run Time
American Legal Publishing	California Los Angeles	MacOS	Google Chrome	“head” And “headless”	12 ~ 13 hours
			Firefox		12 ~ 13 hours
			IE		12 ~ 13 hours
			Safari	“head”	12 ~ 13 hours
Code Of Federal Regulations	Federal	MacOS	Google Chrome	“head”	7 ~ 8 hours

Traverse



When a URL is entered into Selenium, we can extract the source code of the webpage. The information that needs to be extracted includes the link, title name, and trace. We store this information in a list.

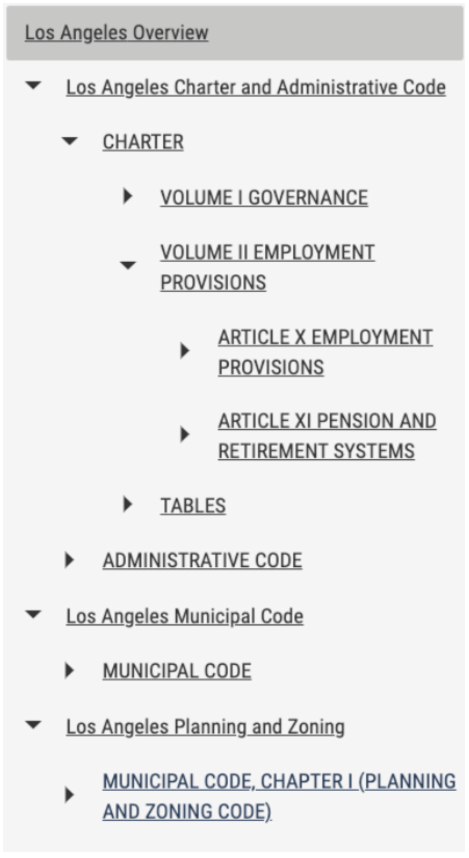
Regarding VISITED_URL, it contains the URL and the information on links and titles in the list, and the information does not appear repeatedly. We also put the information from the list into a queue, and then use the first-in-first-out method to bring the information into the page process function.

Next, Let's zoom in a little bit more on the Logic of Website Traversal and Page Process.

7. Website Traversal and Page Process Core Logic

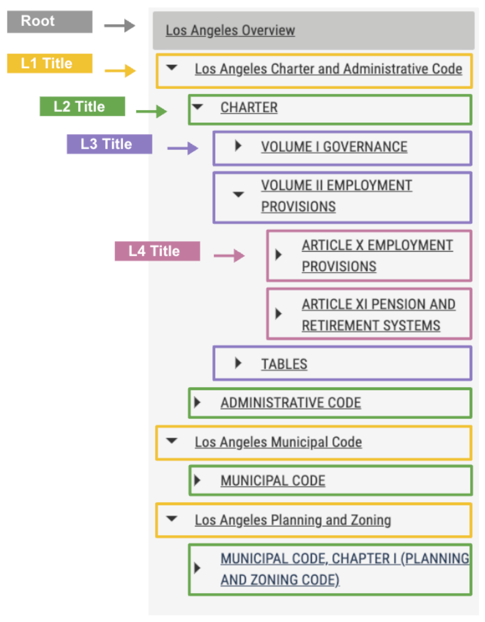
Website Traversal Logic

The picture below shows a general navigation structure of either the code of Federal website or a locality of the American Legal Publishing website. Here we use Los Angeles of American Legal Publishing as an example:

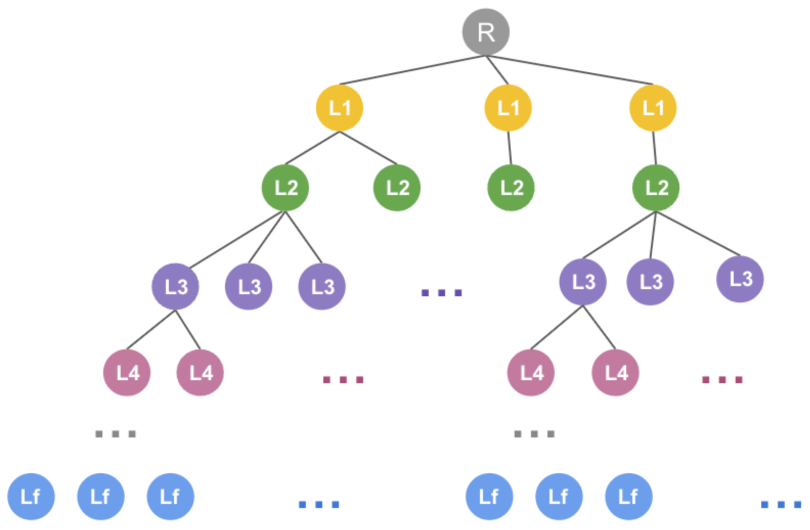


- Los Angeles Overview
- Los Angeles Charter and Administrative Code
  - CHARTER
    - VOLUME I GOVERNANCE
    - VOLUME II EMPLOYMENT PROVISIONS
      - ARTICLE X EMPLOYMENT PROVISIONS
      - ARTICLE XI PENSION AND RETIREMENT SYSTEMS
    - TABLES
  - ADMINISTRATIVE CODE
- Los Angeles Municipal Code
  - MUNICIPAL CODE
- Los Angeles Planning and Zoning
  - MUNICIPAL CODE, CHAPTER I (PLANNING AND ZONING CODE)

The navigation panel contains levels of subtitles. We can get a colorful navigation structure if we mark different levels of subtitles with different colors.

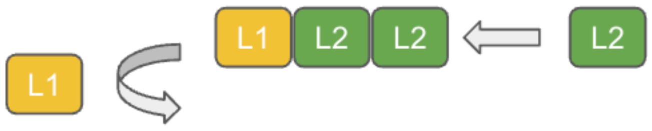

Root
L1 Title
L2 Title
L3 Title
L4 Title
Los Angeles Overview
Los Angeles Charter and Administrative Code
CHARTER
VOLUME I GOVERNANCE
VOLUME II EMPLOYMENT PROVISIONS
ARTICLE X EMPLOYMENT PROVISIONS
ARTICLE XI PENSION AND RETIREMENT SYSTEMS
TABLES
ADMINISTRATIVE CODE
Los Angeles Municipal Code
MUNICIPAL CODE
Los Angeles Planning and Zoning
MUNICIPAL CODE, CHAPTER I (PLANNING AND ZONING CODE)

And this colorful navigation structure can be transformed to an N-nary tree.



In the context of web scraping, the navigation structure of a legal website can be represented as a tree-like structure. In this case, the root node represents the Los Angeles Overview page, while the L1/2/3/4 nodes represent the subtitles of levels 1 to 4, respectively. The leaf nodes represent the deepest content pages, which we can choose to extract content from depending on our requirements.

To traverse this tree-like structure, we implemented a breadth-first search algorithm. The algorithm visits all the nodes at each level of the tree before moving on to the next level. To store the tree nodes, we used a queue data structure. By using a queue and a breadth-first search algorithm, we can ensure that all nodes in the tree are visited in a systematic and efficient manner.



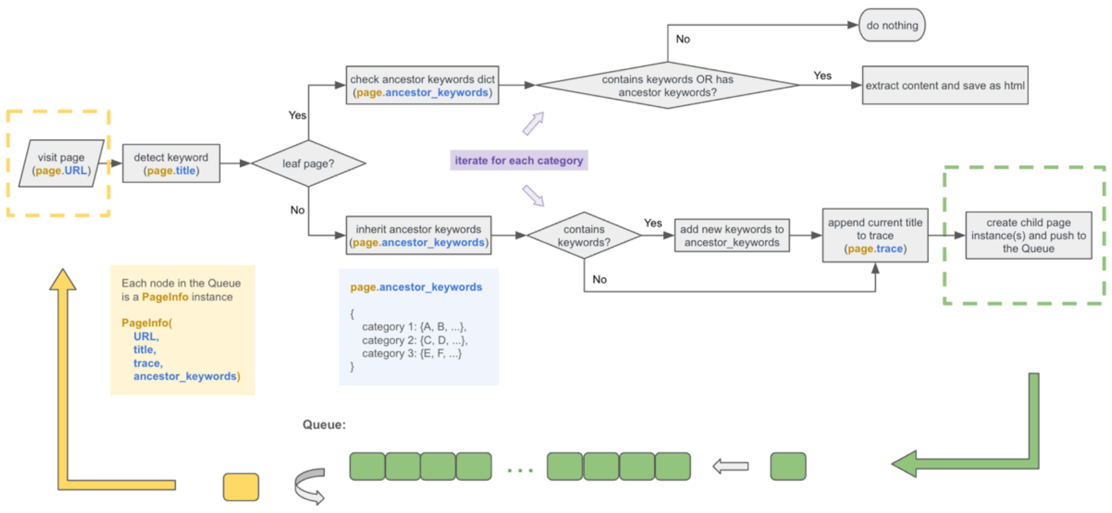
To begin, the root node is added to the queue, following the FIFO rule. This means that the root page will be the first to be removed from the queue. Next, the root page generates three L1 nodes, which are then added to the queue. These L1 nodes are removed in the order in which they were added.

When the first L1 node is removed, it generates two L2 nodes, which are added to the queue. Then, the second L1 node is removed, and it generates one more L2 node, which is also added to the queue. This process continues until all the leaf nodes in the last layer have been completely removed from the queue.

Each node in this process is represent an instance of a single web page, it includes 4 instance variables - URL, title, trace, ancestor_keywords. The ancestor_keywords variable is a dictionary that records any keywords detected along the way in the title of the current page's parent or grandparent. The next section, Page Process Logic, will explain how these variables are used to achieve the desired results.

Page Process Logic

This is the core logic for page process, each node (web page instance in this use case) will be popped out from the queue and loaded into the upper side program.


visit page (page.URL)
detect keyword (page.title)
leaf page?
Yes
No
check ancestor keywords dict (page.ancestor_keywords)
contains keywords OR has ancestor keywords?
No
do nothing
Yes
extract content and save as html
iterate for each category
inherit ancestor keywords (page.ancestor_keywords)
contains keywords?
Yes
No
add new keywords to ancestor_keywords
append current title to trace (page.trace)
create child page instance(s) and push to the Queue
Each node in the Queue is a PageInfo instance
PageInfo(URL, title, trace, ancestor_keywords)
page.ancestor_keywords
{ category 1: {A, B, ...}, category 2: {C, D, ...}, category 3: {E, F, ...} }
Queue:

The program will visit the webpage with the URL information, then it will detect and record whether there's any keyword in this title, and check if the current webpage a leaf page. If it is not a leaf page, means the current page has next layer child pages, then the program will execute the following steps:

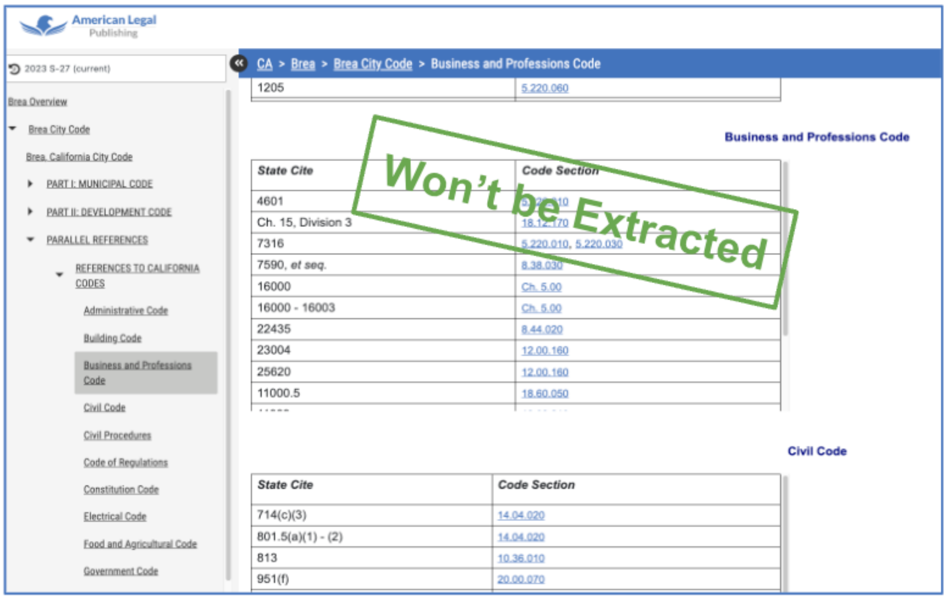
1. All child pages need inherit the ancestor keywords dictionary from the current page node.
2. If there is any new keyword detected in current page title, add it to the dictionary in the corresponding keywords category, which will be used for creating instance(s) of next layer child page(s). Given the ancestor_keyword is a dictionary, it will automatically deduplicate during to process.
3. The trace will be updated by appending the current title, which will be used for creating instance(s) of next layer child page(s).
4. Create child page instance(s) with the updated ancestor_keywords dictionary and trace and append them into the queue.

If the current page is a leaf page, then the program will read the ancestor keywords dictionary, if the current page title contains any keyword or if it has any ancestor keyword recorded, the program will execute content for this page and save it as html file under corresponding keywords category path, otherwise do nothing.

Corner Case Handling Status

During the content extraction, there are 2 corner cases we handled to reduce the generation of invalid html files.

Case 1: If there is no other content beside of table(s), it will be treated as a reference page and won't be extracted.



1205	5.220.060

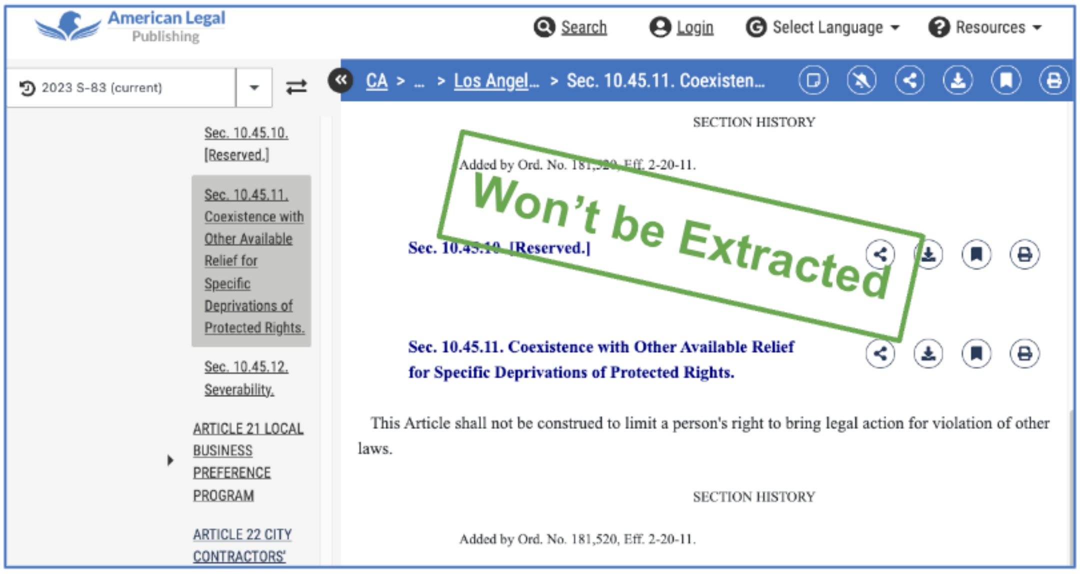
Business and Professions Code

State Cite	Code Section
4601	5.220.010
Ch. 15, Division 3	18.12.170
7316	5.220.010, 5.220.030
7590, *et seq.*	8.38.030
16000	Ch. 5.00
16000 - 16003	Ch. 5.00
22435	8.44.020
23004	12.00.160
25620	12.00.160
11000.5	18.60.050

Civil Code

State Cite	Code Section
714(c)(3)	14.04.020
801.5(a)(1) - (2)	14.04.020
813	10.36.010
951(f)	20.00.070

Case 2: If there are keywords "reserved", "repealed" in the title of the page, it won't be extracted.


American Legal
Publishing
Search
Login
Select Language
Resources
2023 S-83 (current)
CA > ... > Los Angel... > Sec. 10.45.11. Coexisten...
Sec. 10.45.10. [Reserved.]
Sec. 10.45.11. Coexistence with Other Available Relief for Specific Deprivations of Protected Rights.
Sec. 10.45.12. Severability.
ARTICLE 21 LOCAL BUSINESS PREFERENCE PROGRAM
ARTICLE 22 CITY CONTRACTORS'
Won't be Extracted

SECTION HISTORY

Added by Ord. No. 181,520, Eff. 2-20-11.

Sec. 10.45.10. [Reserved.]

Sec. 10.45.11. Coexistence with Other Available Relief for Specific Deprivations of Protected Rights.

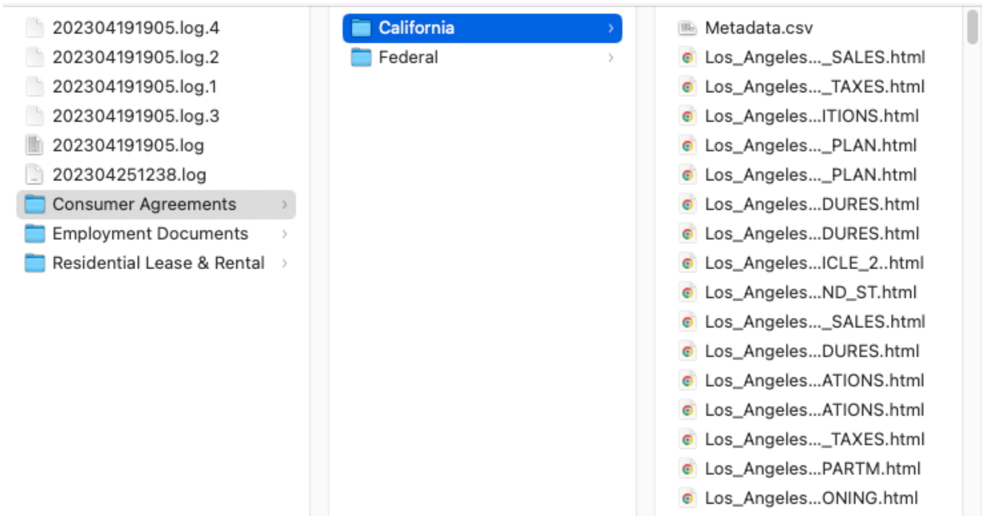
This Article shall not be construed to limit a person's right to bring legal action for violation of other laws.

SECTION HISTORY

Added by Ord. No. 181,520, Eff. 2-20-11.

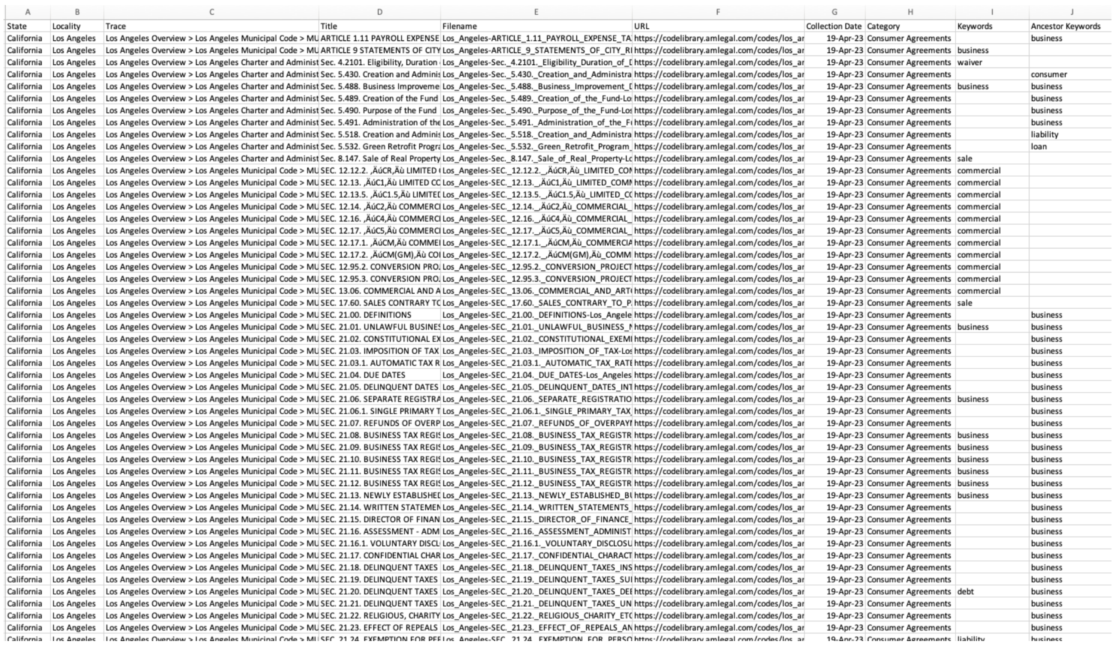
8. Crawling Result

Here are the outputs by fully implementing the crawler program. In the output folder, there will be different sub folders by different keywords category. Under each category, there are sub folders by States or just “Federal”. In each State or Federal folder, it contains the extracted html files and a Metadata summary list.



202304191905.log.4	California	Metadata.csv
202304191905.log.2	Federal	Los_Angeles..._SALES.html
202304191905.log.1		Los_Angeles..._TAXES.html
202304191905.log.3		Los_Angeles...ITIONS.html
202304191905.log		Los_Angeles..._PLAN.html
202304251238.log		Los_Angeles..._PLAN.html
Consumer Agreements		Los_Angeles...DURES.html
Employment Documents		Los_Angeles...DURES.html
Residential Lease & Rental		Los_Angeles...ICLE_2..html
		Los_Angeles...ND_ST.html
		Los_Angeles..._SALES.html
		Los_Angeles...DURES.html
		Los_Angeles...ATIONS.html
		Los_Angeles...ATIONS.html
		Los_Angeles..._TAXES.html
		Los_Angeles...PARTM.html
		Los_Angeles...ONING.html

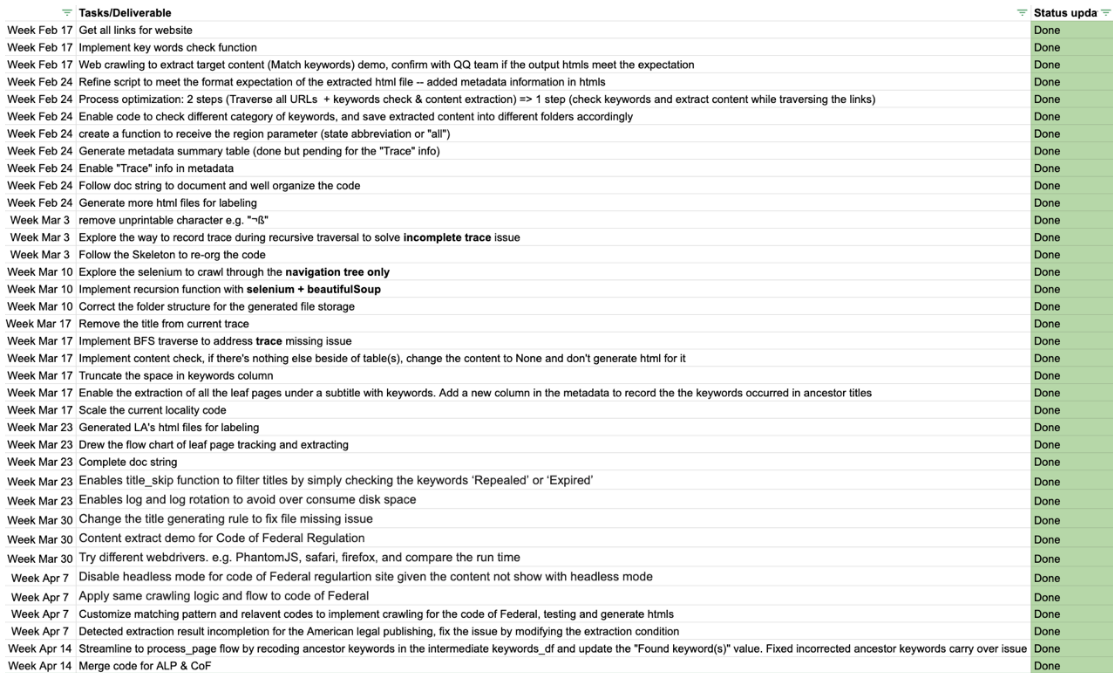
The picture below shows the example of metadata. There are features including State, Locality, Trace, Title, Filename, URL, Collection Date, Category, Keywords, Ancestor Keywords.



A	B	C	D	E	F	G	H	I	J
State	Locality	Trace	Title	Filename	URL	Collection Date	Category	Keywords	Ancestor Keywords
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	ARTICLE 1.11 PAYROLL EXPENSE	Los_Angeles-ARTICLE_1.11_PAYROLL_EXPENSE_TA	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	ARTICLE 9 STATEMENTS OF CITY	Los_Angeles-ARTICLE_9_STATEMENTS_OF_CITY_RI	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	business	
California	Los Angeles	Los Angeles Overview > Los Angeles Charter and Administ	Sec. 4.2101. Eligibility, Duration	Los_Angeles-Sec._4.2101._Eligibility_Duration_of_[	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	waiver	
California	Los Angeles	Los Angeles Overview > Los Angeles Charter and Administ	Sec. 5.430. Creation and Adminis	Los_Angeles-Sec._5.430._Creation_and_Administra	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		consumer
California	Los Angeles	Los Angeles Overview > Los Angeles Charter and Administ	Sec. 5.488. Business Improveme	Los_Angeles-Sec._5.488._Business_Improvement_[	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	business	business
California	Los Angeles	Los Angeles Overview > Los Angeles Charter and Administ	Sec. 5.489. Creation of the Fund	Los_Angeles-Sec._5.489._Creation_of_the_Fund-Lo	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Charter and Administ	Sec. 5.490. Purpose of the Fund	Los_Angeles-Sec._5.490._Purpose_of_the_Fund-Los	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Charter and Administ	Sec. 5.491. Administration of the	Los_Angeles-Sec._5.491._Administration_of_the_F	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Charter and Administ	Sec. 5.518. Creation and Adminis	Los_Angeles-Sec._5.518._Creation_and_Administra	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		liability
California	Los Angeles	Los Angeles Overview > Los Angeles Charter and Administ	Sec. 5.532. Green Retrofit Progra	Los_Angeles-Sec._5.532._Green_Retrofit_Program_	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		loan
California	Los Angeles	Los Angeles Overview > Los Angeles Charter and Administ	Sec. 8.147. Sale of Real Property	Los_Angeles-Sec._8.147._Sale_of_Real_Property-Lc	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	sale	
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 12.12.2. ,ÄúCR,Äù LIMITED	Los_Angeles-SEC._12.12.2._,ÄúCR,Äù_LIMITED_CON	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	commercial	
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 12.13. ,ÄúC1,Äù LIMITED CC	Los_Angeles-SEC._12.13._,ÄúC1,Äù_LIMITED_COMN	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	commercial	
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 12.13.5. ,ÄúC1.5,Äù LIMITED	Los_Angeles-SEC._12.13.5._,ÄúC1.5,Äù_LIMITED_CC	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	commercial	
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 12.14. ,ÄúC2,Äù COMMERCI	Los_Angeles-SEC._12.14._,ÄúC2,Äù_COMMERCIAL_	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	commercial	
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 12.16. ,ÄúC4,Äù COMMERCI	Los_Angeles-SEC._12.16._,ÄúC4,Äù_COMMERCIAL_	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	commercial	
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 12.17. ,ÄúC5,Äù COMMERCI	Los_Angeles-SEC._12.17._,ÄúC5,Äù_COMMERCIAL_	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	commercial	
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 12.17.1. ,ÄúCM,Äù COMMEI	Los_Angeles-SEC._12.17.1._,ÄúCM,Äù_COMMERCIA	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	commercial	
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 12.17.2. ,ÄúCM(GM),Äù COI	Los_Angeles-SEC._12.17.2._,ÄúCM(GM),Äù_COMM	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	commercial	
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 12.95.2. CONVERSION PRO.	Los_Angeles-SEC._12.95.2._CONVERSION_PROJECT	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	commercial	
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 12.95.3. CONVERSION PRO.	Los_Angeles-SEC._12.95.3._CONVERSION_PROJECT	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	commercial	
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 13.06. COMMERCIAL AND A	Los_Angeles-SEC._13.06._COMMERCIAL_AND_ART(	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	commercial	
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 17.60. SALES CONTRARY TC	Los_Angeles-SEC._17.60._SALES_CONTRARY_TO_P.	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	sale	
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.00. DEFINITIONS	Los_Angeles-SEC._21.00._DEFINITIONS-Los_Angele	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.01. UNLAWFUL BUSINES	Los_Angeles-SEC._21.01._UNLAWFUL_BUSINESS_N	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	business	business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.02. CONSTITUTIONAL EX	Los_Angeles-SEC._21.02._CONSTITUTIONAL_EXEMI	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.03. IMPOSITION OF TAX	Los_Angeles-SEC._21.03._IMPOSITION_OF_TAX-Los	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.03.1. AUTOMATIC TAX R	Los_Angeles-SEC._21.03.1._AUTOMATIC_TAX_RATE	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.04. DUE DATES	Los_Angeles-SEC._21.04._DUE_DATES-Los_Angeles	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.05. DELINQUENT DATES	Los_Angeles-SEC._21.05._DELINQUENT_DATES_INT	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.06. SEPARATE REGISTRA	Los_Angeles-SEC._21.06._SEPARATE_REGISTRATIO	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	business	business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.06.1. SINGLE PRIMARY T	Los_Angeles-SEC._21.06.1._SINGLE_PRIMARY_TAX_	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.07. REFUNDS OF OVERP	Los_Angeles-SEC._21.07._REFUNDS_OF_OVERPAYI	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.08. BUSINESS TAX REGIS	Los_Angeles-SEC._21.08._BUSINESS_TAX_REGISTR	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	business	business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.09. BUSINESS TAX REGIS	Los_Angeles-SEC._21.09._BUSINESS_TAX_REGISTR	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	business	business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.10. BUSINESS TAX REGIS	Los_Angeles-SEC._21.10._BUSINESS_TAX_REGISTR	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	business	business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.11. BUSINESS TAX REGIS	Los_Angeles-SEC._21.11._BUSINESS_TAX_REGISTR	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	business	business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.12. BUSINESS TAX REGIS	Los_Angeles-SEC._21.12._BUSINESS_TAX_REGISTR	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	business	business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.13. NEWLY ESTABLISHED	Los_Angeles-SEC._21.13._NEWLY_ESTABLISHED_BU	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	business	business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.14. WRITTEN STATEMEN	Los_Angeles-SEC._21.14._WRITTEN_STATEMENTS_	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.15. DIRECTOR OF FINAN	Los_Angeles-SEC._21.15._DIRECTOR_OF_FINANCE_	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.16. ASSESSMENT - ADM	Los_Angeles-SEC._21.16._ASSESSMENT_ADMINIST	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.16.1. VOLUNTARY DISCL	Los_Angeles-SEC._21.16.1._VOLUNTARY_DISCLOSU	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.17. CONFIDENTIAL CHAR	Los_Angeles-SEC._21.17._CONFIDENTIAL_CHARACT	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.18. DELINQUENT TAXES	Los_Angeles-SEC._21.18._DELINQUENT_TAXES_INS	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.19. DELINQUENT TAXES	Los_Angeles-SEC._21.19._DELINQUENT_TAXES_SUI	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.20. DELINQUENT TAXES	Los_Angeles-SEC._21.20._DELINQUENT_TAXES_DEI	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	debt	business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.21. DELINQUENT TAXES	Los_Angeles-SEC._21.21._DELINQUENT_TAXES_UN	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.22. RELIGIOUS, CHARITY	Los_Angeles-SEC._21.22._RELIGIOUS_CHARITY_ETC	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.23. EFFECT OF REPEALS	Los_Angeles-SEC._21.23._EFFECT_OF_REPEALS_AN	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements		business
California	Los Angeles	Los Angeles Overview > Los Angeles Municipal Code > MU	SEC. 21.24. EXEMPTION FOR PEF	Los_Angeles-SEC._21.24._EXEMPTION_FOR_PERSO	https://codelibrary.amlegal.com/codes/los_ar	19-Apr-23	Consumer Agreements	liability	business

9. Subtasks and Challenges in Development

Lastly, we would like to attach the list which covers most of the sub-tasks and challenges we encountered and closed during the crawler development.



	Tasks/Deliverable	Status upda
Week Feb 17	Get all links for website	Done
Week Feb 17	Implement key words check function	Done
Week Feb 17	Web crawling to extract target content (Match keywords) demo, confirm with QQ team if the output htmls meet the expectation	Done
Week Feb 24	Refine script to meet the format expectation of the extracted html file -- added metadata information in htmls	Done
Week Feb 24	Process optimization: 2 steps (Traverse all URLs + keywords check & content extraction) => 1 step (check keywords and extract content while traversing the links)	Done
Week Feb 24	Enable code to check different category of keywords, and save extracted content into different folders accordingly	Done
Week Feb 24	create a function to receive the region parameter (state abbreviation or "all")	Done
Week Feb 24	Generate metadata summary table (done but pending for the "Trace" info)	Done
Week Feb 24	Enable "Trace" info in metadata	Done
Week Feb 24	Follow doc string to document and well organize the code	Done
Week Feb 24	Generate more html files for labeling	Done
Week Mar 3	remove unprintable character e.g. "¬ß"	Done
Week Mar 3	Explore the way to record trace during recursive traversal to solve **incomplete trace** issue	Done
Week Mar 3	Follow the Skeleton to re-org the code	Done
Week Mar 10	Explore the selenium to crawl through the **navigation tree only**	Done
Week Mar 10	Implement recursion function with **selenium + beautifulSoup**	Done
Week Mar 10	Correct the folder structure for the generated file storage	Done
Week Mar 17	Remove the title from current trace	Done
Week Mar 17	Implement BFS traverse to address **trace** missing issue	Done
Week Mar 17	Implement content check, if there's nothing else beside of table(s), change the content to None and don't generate html for it	Done
Week Mar 17	Truncate the space in keywords column	Done
Week Mar 17	Enable the extraction of all the leaf pages under a subtitle with keywords. Add a new column in the metadata to record the the keywords occurred in ancestor titles	Done
Week Mar 17	Scale the current locality code	Done
Week Mar 23	Generated LA's html files for labeling	Done
Week Mar 23	Drew the flow chart of leaf page tracking and extracting	Done
Week Mar 23	Complete doc string	Done
Week Mar 23	Enables title_skip function to filter titles by simply checking the keywords 'Repealed' or 'Expired'	Done
Week Mar 23	Enables log and log rotation to avoid over consume disk space	Done
Week Mar 30	Change the title generating rule to fix file missing issue	Done
Week Mar 30	Content extract demo for Code of Federal Regulation	Done
Week Mar 30	Try different webdrivers. e.g. PhantomJS, safari, firefox, and compare the run time	Done
Week Apr 7	Disable headless mode for code of Federal regulartion site given the content not show with headless mode	Done
Week Apr 7	Apply same crawling logic and flow to code of Federal	Done
Week Apr 7	Customize matching pattern and relavent codes to implement crawling for the code of Federal, testing and generate htmls	Done
Week Apr 7	Detected extraction result incompletion for the American legal publishing, fix the issue by modifying the extraction condition	Done
Week Apr 14	Streamline to process_page flow by recoding ancestor keywords in the intermediate keywords_df and update the "Found keyword(s)" value. Fixed incorrected ancestor keywords carry over issue	Done
Week Apr 14	Merge code for ALP & CoF	Done

During the program development, we faced two major challenges that required considerable attention:

1. Adapting the program to different legal websites with minimal customization proved to be more challenging than we anticipated due to varying navigation structures across websites. For example, in the American Legal Publishing website, different layers of titles may point to the same URL, if we don't handle it well, the generated trace may skip certain layers of titles and come out an incomplete trace, whereas this is not the case for the Code of Federal website. To overcome this kind of issue, we maximize modularity during the programming, narrow down those very specific differences, and write customized code for them.
2. Testing and debugging the program proved to be time-consuming, and errors could occur after several hours of running. A crucial lesson learned from this experience is the importance of logging running progress to aid debugging, especially when information is limited.

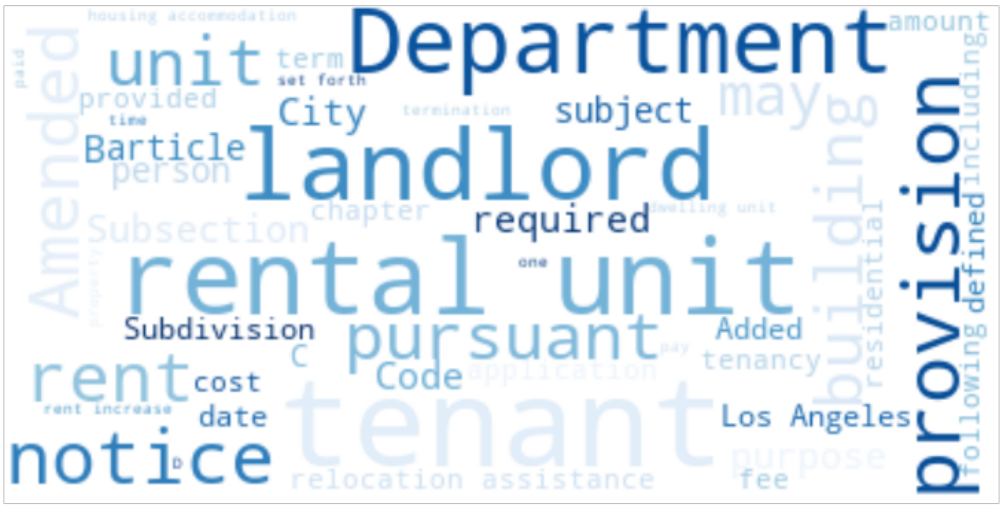
10. Text Classification Models

The problem being addressed in this project is to classify legal documents as “Good” or “Bad” based on their content. The dataset used for this project is highly imbalanced with most documents labeled as “Good”. This presents a challenge in developing an effective classification model, as the model may be biased towards the majority class and perform poorly on minority class. The dataset used for this project was obtained by web scraping. It contains documents labelled as “Good” and “Bad”, along with additional metadata such as “Title” and “Trace”. The dataset includes a total of 757 documents, with 651 documents labelled as “Bad” and 106 documents labelled as “Good”. The text column contains the cleaned text from the HTML files, which has undergone several data cleaning techniques to remove irrelevant information. This column is essential for text analysis and will be used to determine whether the text is considered “Good” or not. The title column contains the title of HTML file, which provides the additional context for the text. This column is helpful for identifying patterns or themes in the data. The trace column contains additional metadata about the HTML file. This column is helpful for understanding the context of the text and identifying any potential biases. It contains binary value of ‘True’ and ‘False’, indicating whether the text is ‘good’ or not. The label was assigned based on some criteria which could include the relevance of text to analysis.

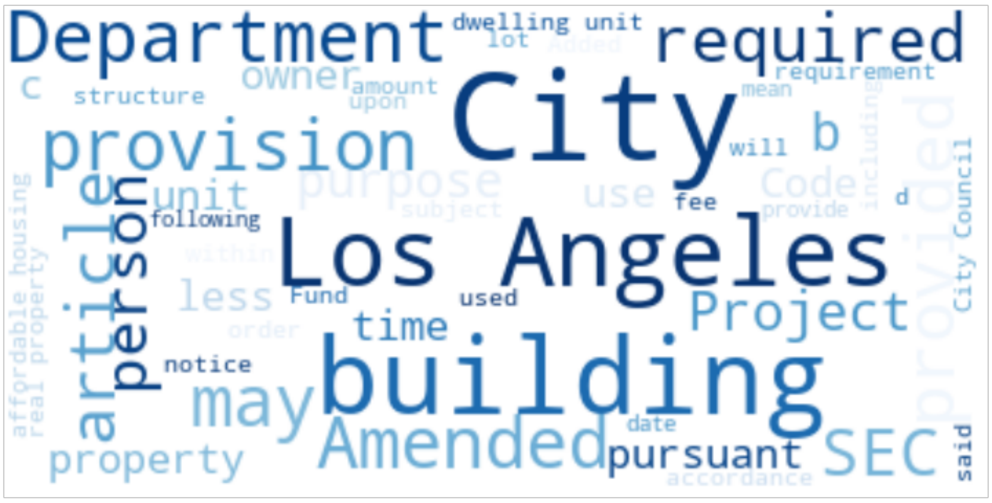
11. Exploratory Data Analysis

To understand the data better and generate insights, we did some data exploration.
Word Cloud: To visualize the most common words in ‘Good’ and ‘Bad’ files in the dataset, we created two-word clouds. These word clouds are generated using the ‘WordCloud’ library in Python.
Generating Word Clouds: To generate the word clouds, all the cleaned text from the ‘good’ files and the ‘bad’ files were joined separately. Then, a set of stop words were created, which included common words such as ‘Section’, ‘Ord’, and ‘Eff’, and add them to existing set of stop words. Then, a WordCloud object was created with specific parameters such as maximum font size, maximum number of words to include, and background color. The colormap was set to ‘Blues_r’ for a blue color scheme. The resulting word cloud shows the most common words in the ‘Good’ and ‘Bad’ files in the dataset.

Most Common Words in Good Files



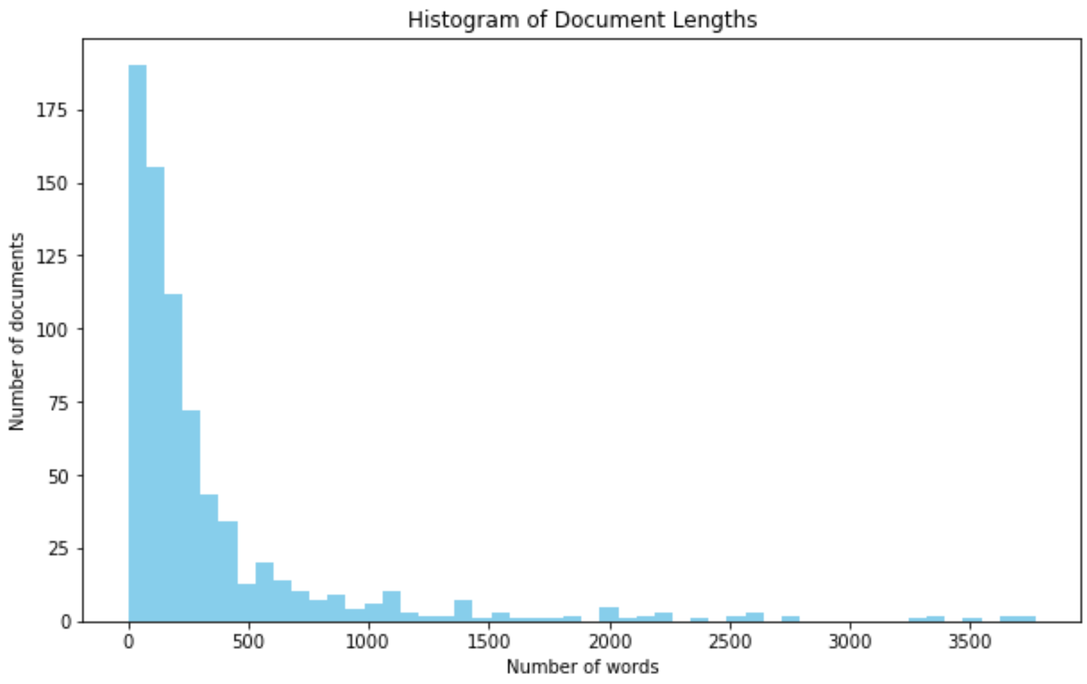
Most Common Words in Bad Files



Histogram of Document Lengths

To generate the histogram of document lengths, we first calculate the length of each document in the dataset using the 'Cleaned Text' column. We split each document by whitespace and count the number of resulting words. The resulting list of document lengths is stored in the variable 'doc_lengths'.

We then create a histogram of the document lengths using the 'plt.hist()' function. The histogram shows the number of documents on the y-axis and the range of document lengths on the x-axis. We use the 'bins' parameter to set the number of bins in the histogram and the 'range' parameter to exclude outliers from the plot. The resulting plot shows the distribution of document lengths in the dataset.

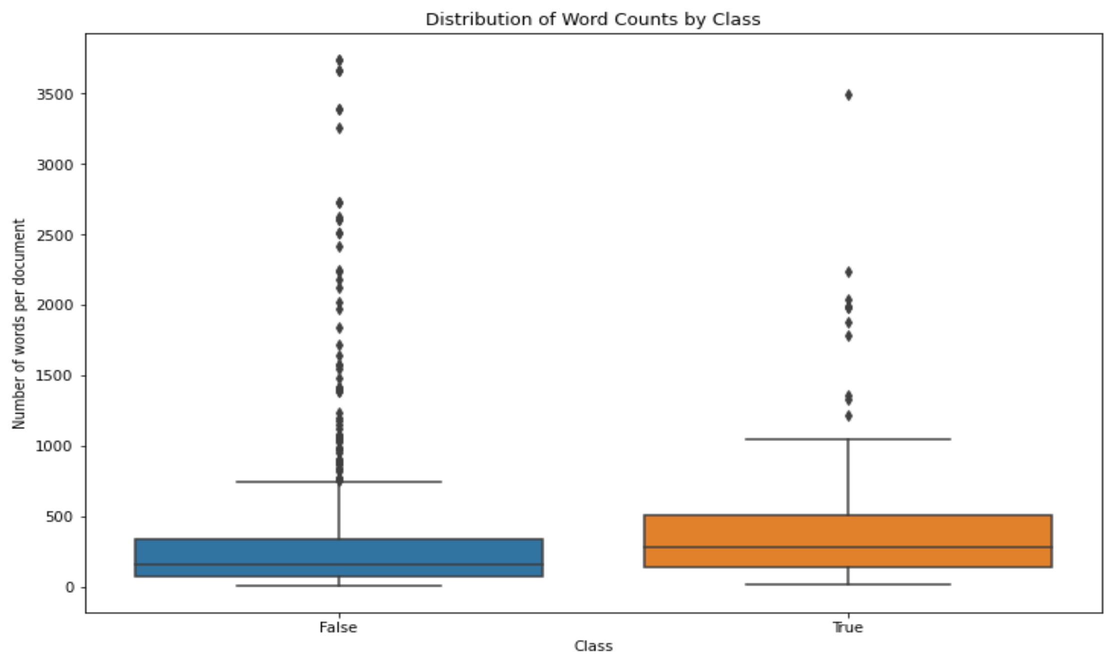


Results

The histogram of document lengths shows that most documents in the dataset are relatively short, with a length of fewer than 500 words. There are a few longer documents in the dataset, but they are relatively rare. This information can be useful for understanding the characteristics of the dataset and selecting appropriate models and hyperparameters for the text classification task.

BoxPlot

To generate box plot, we first add the list of document lengths to the original dataset as a new column called 'Word Count'. We then create a box plot using the 'sns.boxplot()' function. The box plot shows the distribution of document lengths for each class on the y-axis and the class labels on the x-axis. The boxes show the interquartile range (IQR) of the distribution, while the whiskers extend to the minimum and maximum values within 1.5 times the IQR.



Results

The box plot of document lengths shows that the distribution of document lengths is slightly different between the 'Good' and 'Bad' classes. The median and mean document lengths are slightly higher for the 'Good' class than the 'Bad' class, and there are a few more outliers for the 'Bad' class. However, the overall difference is relatively small, and there is a significant overlap between the two distributions. This information can be useful for understanding the characteristics of the dataset and selecting appropriate models and hyperparameters for the text classification task.

12. Preprocessing Step for All Models

To prepare the text for analysis, several data cleaning techniques were used. Firstly, any links present in the text were removed, as these are not relevant for analysis. Next, stop words were removed, which are commonly used words that are not useful for analysis, such as “the” and “a”. Moreover, special characters, digits, and punctuation marks were removed as these can interfere with analysis and lead to inaccuracies. Finally, any leading or trailing white spaces were removed which can also interfere with analysis. To ensure the consistency in the cleaned text, it was converted to lowercase. This was done to avoid any issues with case sensitivity that may arise during analysis.

13. Logistic Regression, Random Forest, XGBoost Preprocessing

A. Vectorizer

To perform text classification on a dataset containing traces, titles, and text, we need to extract features from these text columns. To extract features, we use the TfidfVectorizer from scikit-learn library. TfidfVectorizer converts the text data into a matrix of numerical features that can be used by a machine learning model. To vectorize the trace, title, and text columns, we first create three instances of the TfidfVectorizer class, named vectorizer_trace, vectorizer_title, and vectorizer_text. These vectorizers are used to transform each text column into a matrix of numerical features. Next, we use the fit_transform method of each vectorizer to transform the trace_column, title_column, and text_column into matrices of numerical features. The fit_transform method fits the vectorizer to the text data and transforms the text data into a matrix of numerical features. Each transformed matrix of numerical features is saved into a separate variable: trace_features, title_features, and text_features.

B. Combining Feature Matrices

To train a text classification model on the dataset containing traces, titles, and text, we need to combine the features extracted from each column. To combine the feature matrices, we use the hstack method from the scipy library. The hstack method stacks the feature matrices horizontally, which means that the resulting combined feature matrix will have all the feature columns from the three original feature matrices.

C. Label Encoder

We used the LabelEncoder class from the scikit-learn library to convert the binary target variable "Good" into a numerical format that can be used by a machine learning algorithm. We created an instance of the LabelEncoder class, fit it to the target variable data, and transformed the data into a numerical format using the fit_transform method. The resulting encoded data is saved into a new variable called "relevance_encoded", which will be used as the target variable in the text classification model.

D. Splitting Dataset into Training and Test Sets

We use the train_test_split function to split the combined feature matrix "combined_features" and the target variable "relevance_encoded" into training and testing sets. We pass the following arguments to the function:

- **combined_features**: The combined feature matrix that contains the extracted features from the trace, title, and text columns.
- **relevance_encoded**: The target variable that contains the numerical values of "Good" column after encoding using LabelEncoder.
- **test_size**: The proportion of the dataset that should be used for testing the model. In this case, we use 0.2, which means 20% of the dataset will be used for testing.
- **random_state**: A random seed used for reproducibility of the results. We set it to 42 in this case.
- **stratify**: We set this parameter to "relevance_encoded" to ensure that the proportion of the target variable is the same in the training and testing datasets.

The train_test_split function returns four variables: X_train, X_test, y_train, and y_test.

These variables contain the training and testing sets of the combined feature matrix and the target variable.

To find the best hyperparameters for a text classification model, we use a technique called hyperparameter tuning. Hyperparameter tuning involves searching over a range of hyperparameters to find the best combination that maximizes the performance of the model. We define three hyperparameter grids, one for each of the three classification models: Logistic Regression, Random Forest, and XGBoost. The hyperparameter grid for Logistic Regression contains three hyperparameters: C, penalty, and max_iter. C is the inverse of regularization strength, penalty is the type of regularization, and max_iter is the maximum number of iterations for the solver to converge. The hyperparameter grid for Random Forest classifier contains three hyperparameters: n_estimators, max_depth, and min_samples_split. n_estimators is the number of trees in the forest, max_depth is the maximum depth of each tree, and min_samples_split is the minimum number of samples required to split an internal node. The hyperparameter grid for XGBClassifier contains three hyperparameters: learning_rate, max_depth, and n_estimators. learning_rate is the step size shrinkage used in update to prevent overfitting, max_depth is the maximum depth of each tree, and n_estimators is the number of trees in the forest.

Why did we choose these hyperparameters?

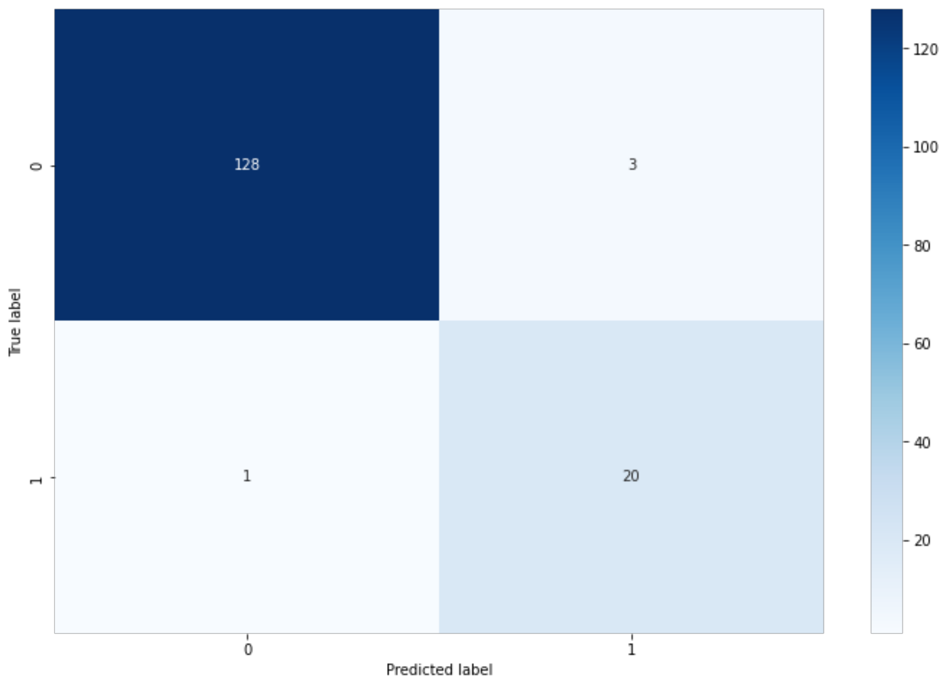
In the hyperparameter grid for Logistic Regression, we chose a range of values for the regularization strength parameter (C) that includes both small and large values. This is because small values of C will result in a stronger regularization, which can help prevent overfitting, while large values of C will result in a weaker regularization, which can help the model capture more of the complex relationships in the data. We also chose a range of values for the maximum number of iterations (max_iter) to ensure that the model has enough time to converge. Similarly, in the hyperparameter grid for Random Forest classifier, we chose a range of values for the number of trees in the forest (n_estimators), the maximum depth of each tree (max_depth), and the minimum number of samples required to split an internal node (min_samples_split) based on standard practices of machine learning experts for text classification tasks.

In the hyperparameter grid for XGBoost, we chose a range of values for the step size shrinkage parameter (learning_rate), the maximum depth of each tree (max_depth), and the number of trees in the forest (n_estimators) based on standard practices of machine learning experts for text classification tasks. After defining the hyperparameter grids, we used a technique called grid search to systematically search over these hyperparameters to find the best combination of hyperparameters that maximize the performance of the model.

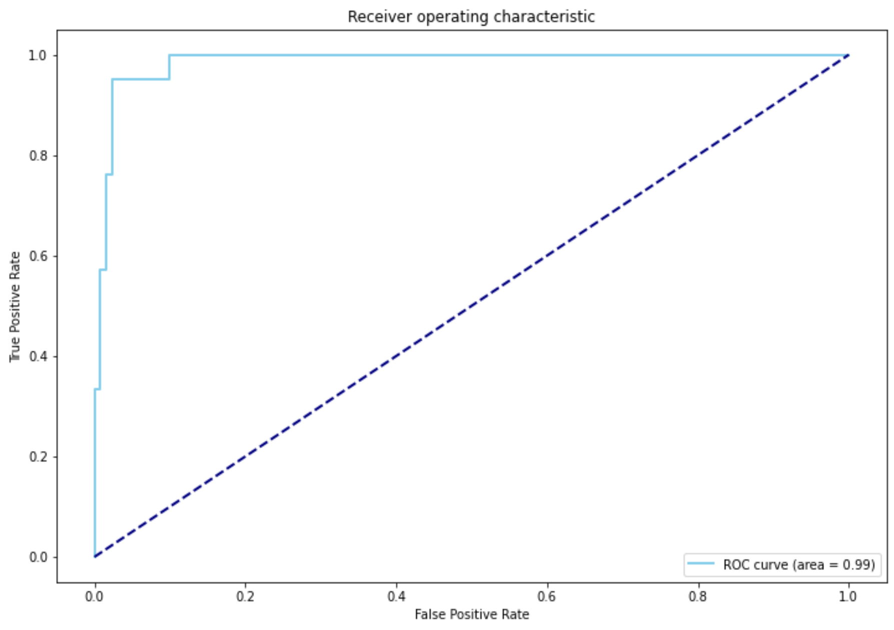
14. Results

Logistic Regression

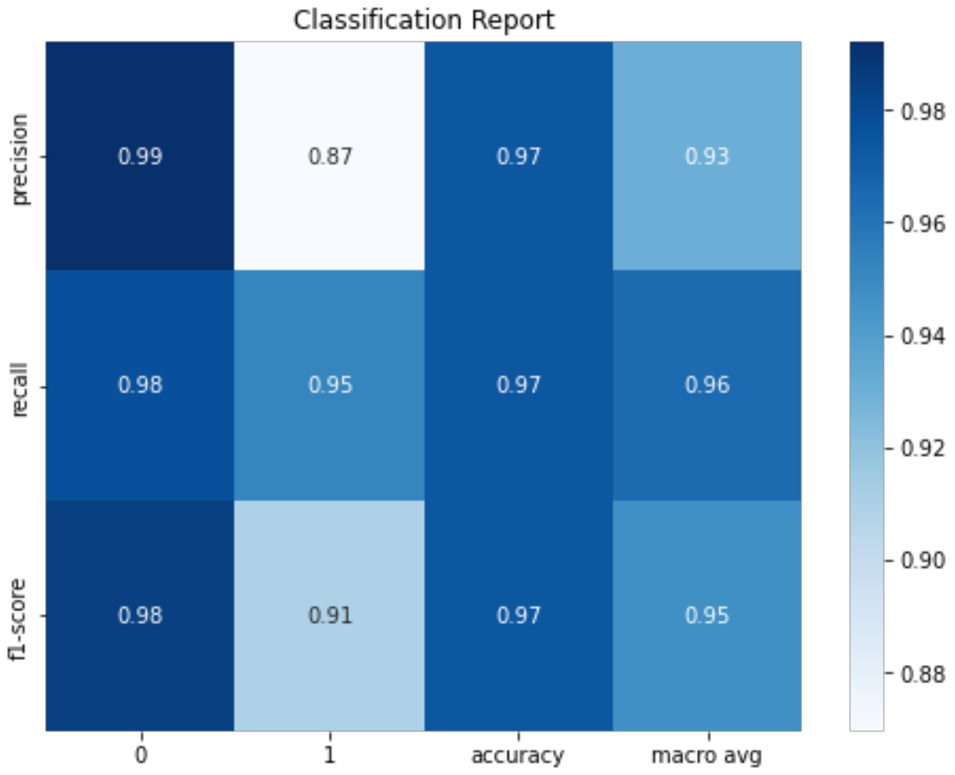
Confusion Matrix



ROC Curve



Classification Report



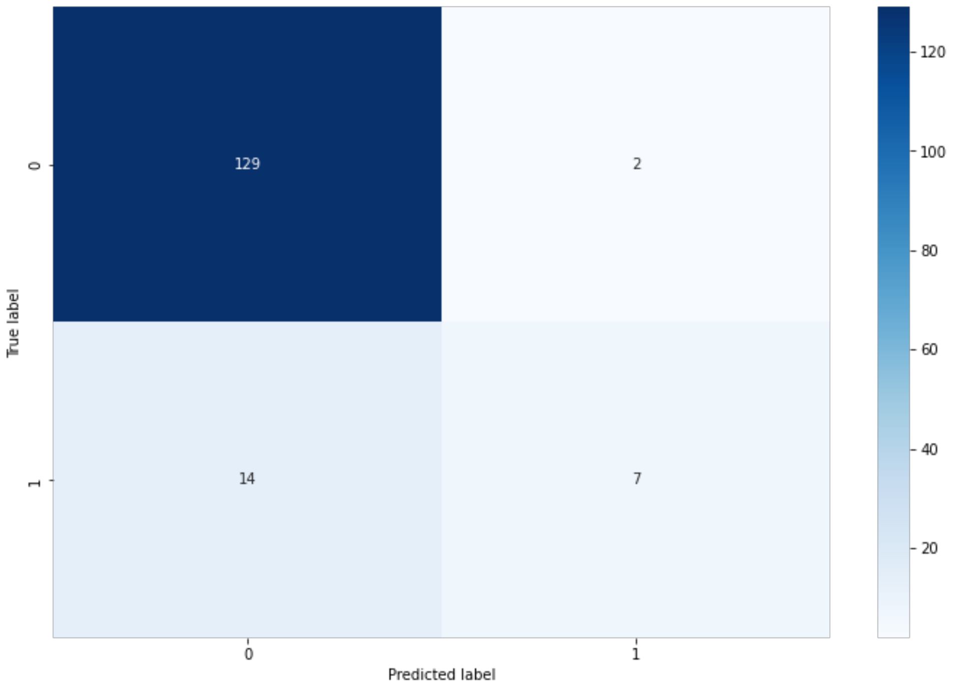
The logistic regression model was trained using the hyperparameters specified in the parameter grid. After training the model, predictions were made on the test set and a confusion matrix was generated to visualize the performance of the model. The confusion matrix shows the number of true positives, true negatives, false positives, and false negatives. We also generated an ROC curve to visualize the trade-off between the true positive rate and false positive rate at different probability thresholds. Finally, we generated a classification report, which provides precision, recall, and F1-score for each class.
The resulting confusion matrix shows that the model has high true positives and true negatives and low false positives and false negatives. The ROC curve shows that the model has a high true positive rate and a low false positive rate, indicating good performance. The classification report shows that the model has high precision, recall, and F1-score for both the 'good' and 'bad' classes.

Best hyperparameters: {'C': 100, 'max_iter': 100, 'penalty': 'l2'}

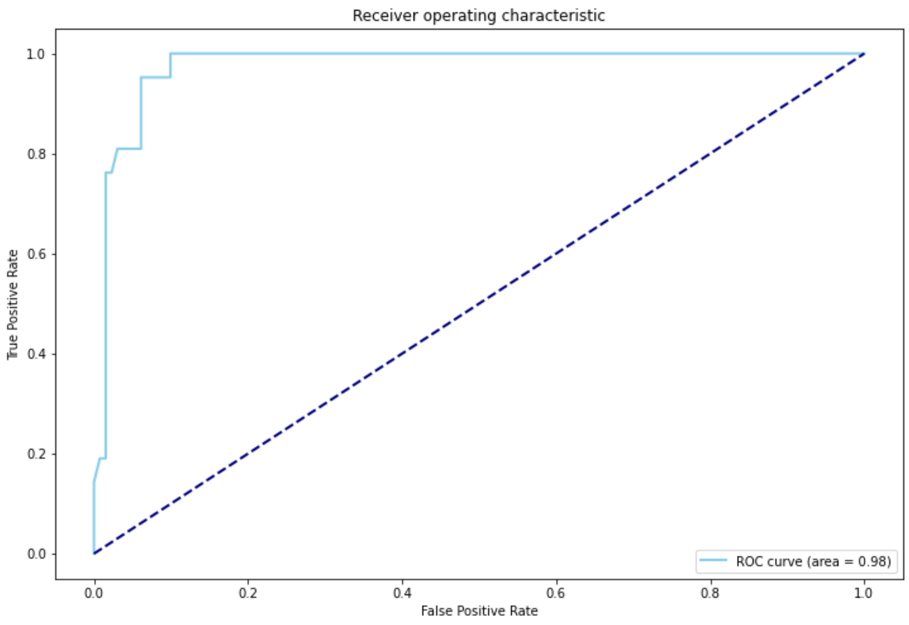
The best hyperparameters for the logistic regression model were found to be 'C' = 100, 'max_iter' = 100, and 'penalty' = 'l2'. These values were chosen based on the results of a grid search, and they indicate that the model should be heavily penalized for large coefficients, should perform a maximum of 100 iterations, and should use 'l2' regularization to help prevent overfitting.

Random Forest

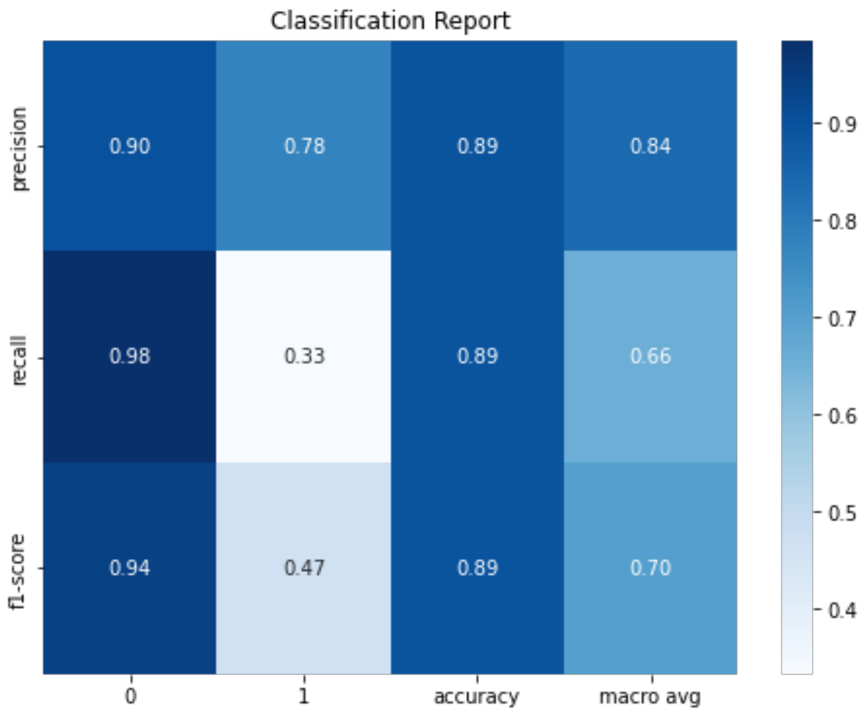
Confusion Matrix



ROC Curve



Classification Report



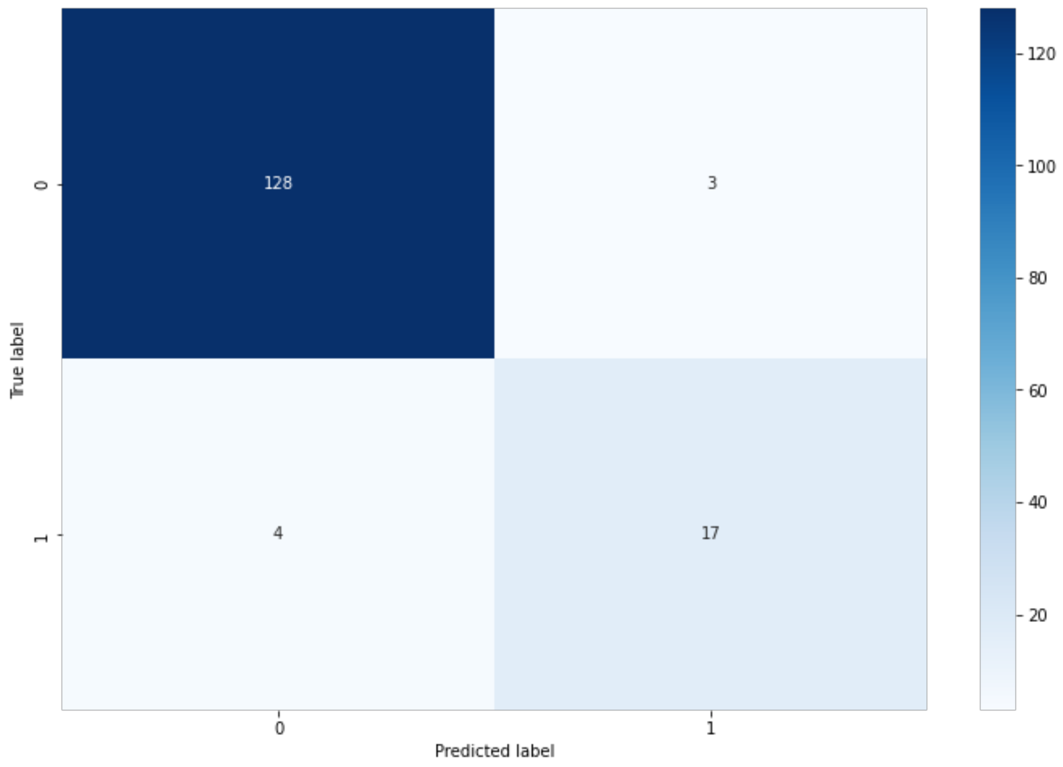
The Random Forest model was trained using the hyperparameters specified in the parameter grid. After training the model, predictions were made on the test set and a confusion matrix was generated to visualize the performance of the model. The confusion matrix shows the number of true positives, true negatives, false positives, and false negatives. We also generated an ROC curve to visualize the trade-off between the true positive rate and false positive rate at different probability thresholds. Finally, we generated a classification report, which provides precision, recall, and F1-score for each class.

Best hyperparameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 1 00}

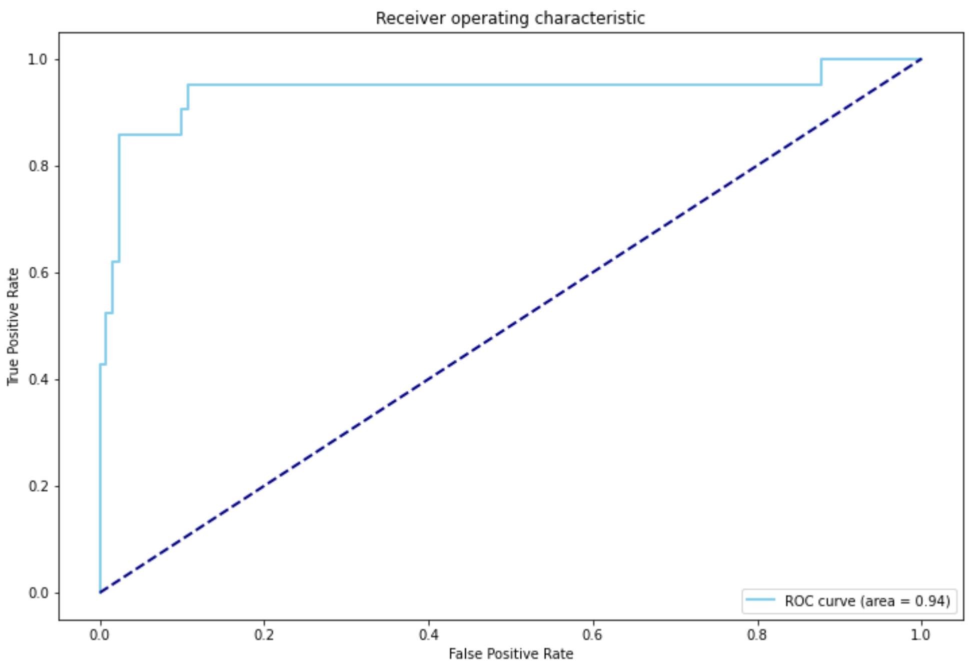
After performing the grid search, we print the best hyperparameters found using the 'best_params_' attribute of the 'GridSearchCV' object. In this case, the best hyperpar ameters are {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}. This means that the optimal Random Forest Classifier uses 100 estimators, no maximum depth limit, and a minimum sample size of 2 for splitting.

XGBoost

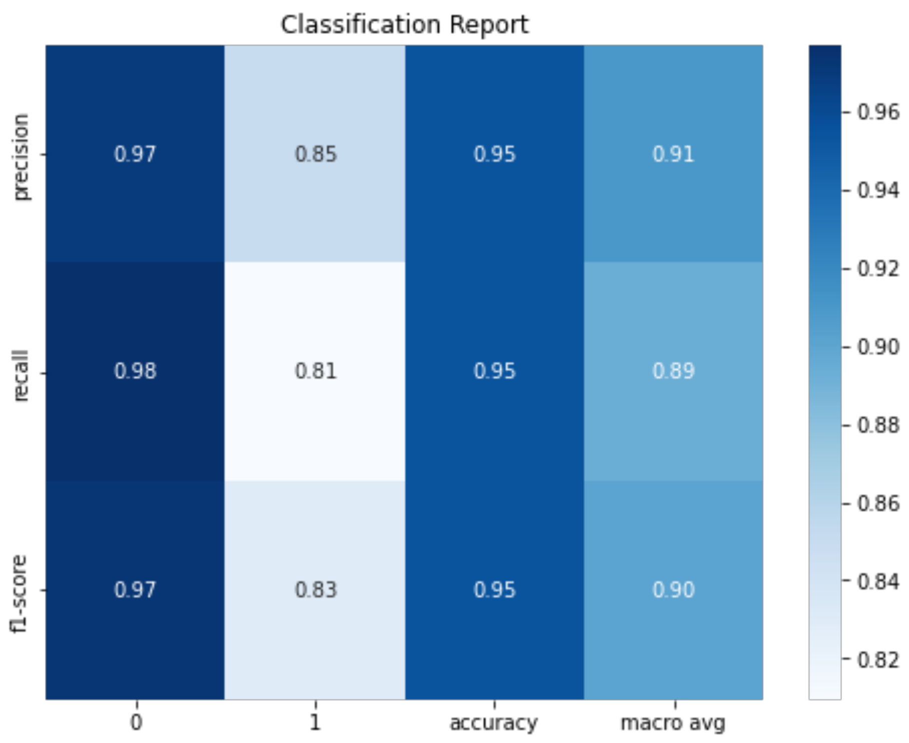
Confusion Matrix



ROC Curve



Classification Report



The XGBoost model was trained using the hyperparameters specified in the parameter grid. After training the model, I made predictions on the test set and generated a confusion matrix to visualize the performance of the model. The confusion matrix shows the number of true positives, true negatives, false positives, and false negatives. I also generated an ROC curve to visualize the trade-off between the true positive rate and false positive rate at different probability thresholds.

Finally, I generated a classification report, which provides precision, recall, and F1-score for each class. The resulting confusion matrix shows that the model has high true positives and true negatives and low false positives and false negatives. The ROC curve shows that the model has a high true positive rate and a low false positive rate, indicating good performance. The classification report shows that the model has high precision, recall, and F1-score for both the 'good' and 'bad' classes.

Best hyperparameters: {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 100}

The best hyperparameters for the XGBoost model were found to be 'learning_rate' = 0.1, 'max_depth' = 7, and 'n_estimators' = 100. These values were chosen based on the results of a grid search and they indicate that the XGBoost model should take small steps to converge, each decision tree can have a maximum depth of 7, and the model should have 100 decision trees.

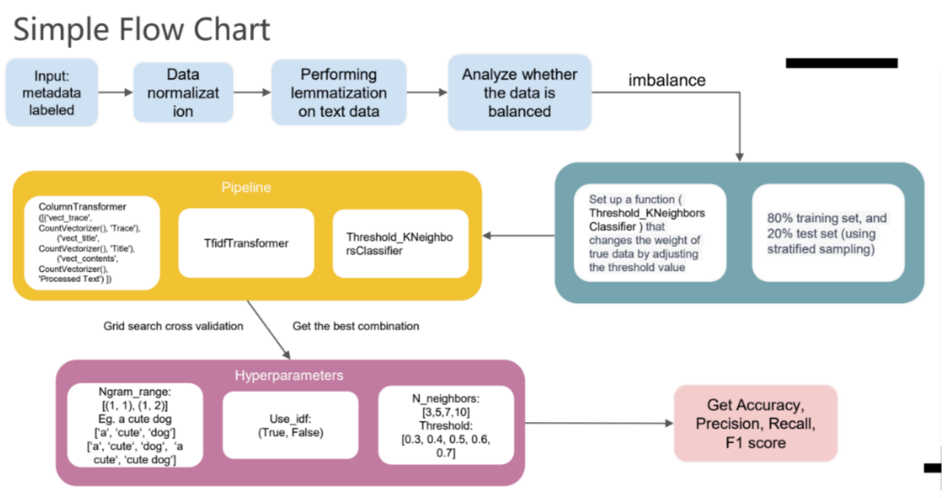
15. KNN, RNN-GRU

Data preprocessing: When using the preprocessed data files, to prevent any omissions, punctuation marks, spaces, and other content are removed again, and words are lemmatized.

Data analysis: The bar chart shows that in the dataset, there are over 600 fake samples and over 100 real samples, indicating that the dataset is imbalanced. To balance the data, first, the data is split into 80% for training and 20% for testing, and stratified sampling is used to ensure that the ratio of fake and real samples in the training and testing sets is the same.

However, this method alone cannot balance the data. Different approaches are used for KNN and GRU to address data imbalance, and there are some differences in how KNN and GRU handle data. These will be explained separately in the following sections.

A. KNN Model


Simple Flow Chart
Input: metadata labeled
Data normalization
Performing lemmatization on text data
Analyze whether the data is balanced
imbalance
Set up a function (Threshold_KNeighbors Classifier) that changes the weight of true data by adjusting the threshold value
80% training set, and 20% test set (using stratified sampling)
Pipeline
ColumnTransformer ([('vect_trace', CountVectorizer(), 'Trace'), ('vect_title', CountVectorizer(), 'Title'), ('vect_contents', CountVectorizer(), 'Processed Text')])
TfidfTransformer
Threshold_KNeighborsClassifier
Grid search cross validation
Get the best combination
Hyperparameters
Ngram_range: [(1, 1), (1, 2)] Eg. a cute dog ['a', 'cute', 'dog'] ['a', 'cute', 'dog', 'a cute', 'cute dog']
Use_idf: (True, False)
N_neighbors: [3,5,7,10] Threshold: [0.3, 0.4, 0.5, 0.6, 0.7]
Get Accuracy, Precision, Recall, F1 score

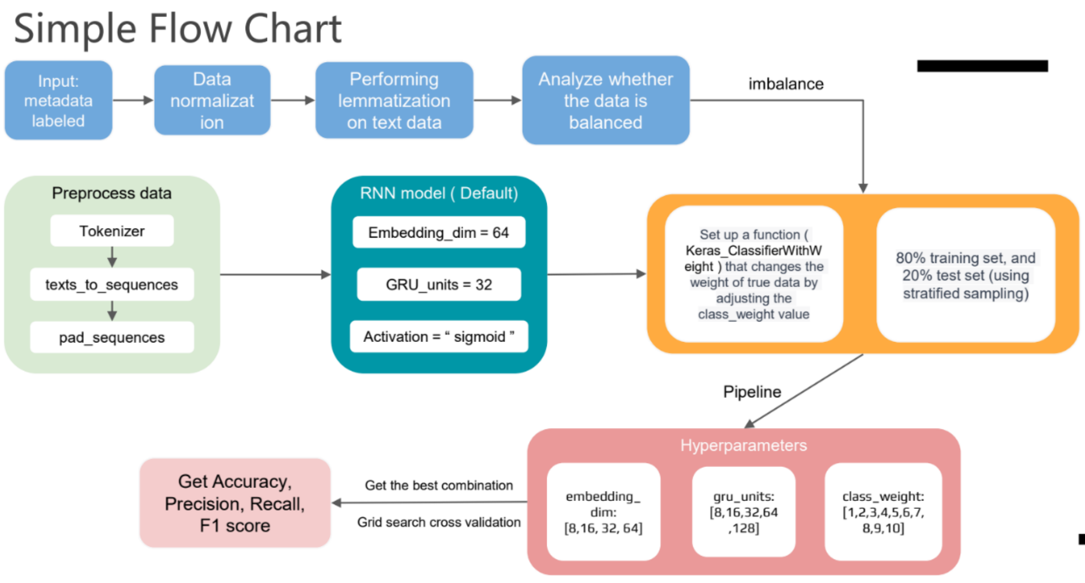
Model creation: It's important to create a KNN model based on Euclidean distance that can accept changes in the threshold. By default, the threshold is 0.5; when the probability of the positive class is greater than or equal to 0.5, the data point is considered to belong to the positive class; otherwise, it belongs to the negative class. Adjusting the threshold can change the classifier's preference, and in this way, the model can balance imbalanced data. However, since the model is uncertain about the choice of the optimal threshold, this parameter is put into the pipeline for cross-validation.

Pipeline setup: In the pipeline, first, the CountVectorizer function is used to convert text data into a feature matrix, tokenize text, and build a vocabulary. It has an ngram_range parameter to achieve different processing effects. Then, the TfidfTransformer function is used in the pipeline to convert the term frequency matrix into a TF-IDF representation. The third step is to run the KNN model that can accept changes in the threshold.

Parameter explanation: The CountVectorizer function has an ngram_range parameter. When the ngram_range value is [(1, 1), (1, 2)], the processing effects for the sentence "a cute dog" are ['a', 'cute', 'dog'] and ['a', 'cute', 'dog', 'a cute', 'cute dog'], respectively. Its value determines whether the model can better understand and capture the context relationships in the text. The TfidfTransformer function's use_idf parameter decides whether to apply IDF weights to the TF (term frequency) values to calculate the TF-IDF values by changing true and false values. When true, it uses IDF, which helps to reduce the weight of common words (such as stop words) while increasing the weight of less frequent words in a specific document. When false, it uses TF, and the weight of all words will be based solely on their frequency in the document, without considering their distribution across the entire document collection. N_neighbors is the most important K value for KNN, and the threshold, as mentioned earlier in the article, changes the classifier's preference to address data imbalance.

Hyperparameter selection: For training speed considerations, the ngram_range is set to [(1, 1), (1, 2)]. The use_idf parameter only has true and false options. The n_neighbors parameter is initially set to [5, 10, 15]. During the training process, it is found that the performance is best at 5. To explore better results, the parameter is changed to [3, 5, 7, 10]. At the same time, since the KNN model can accept changes in the threshold, based on the ratio of True and False data, the parameter is chosen from a larger range, [0.3, 0.4, 0.5, 0.6, 0.7].

B. GRU Model


Simple Flow Chart
Input: metadata labeled
Data normalization
Performing lemmatization on text data
Analyze whether the data is balanced
imbalance
Preprocess data
Tokenizer
texts_to_sequences
pad_sequences
RNN model (Default)
Embedding_dim = 64
GRU_units = 32
Activation = " sigmoid "
Set up a function (Keras_ClassifierWithWeight) that changes the weight of true data by adjusting the class_weight value
80% training set, and 20% test set (using stratified sampling)
Pipeline
Hyperparameters
embedding_dim: [8,16, 32, 64]
gru_units: [8,16,32,64,128]
class_weight: [1,2,3,4,5,6,7,8,9,10]
Get the best combination
Grid search cross validation
Get Accuracy, Precision, Recall, F1 score
[redacted]
[redacted]

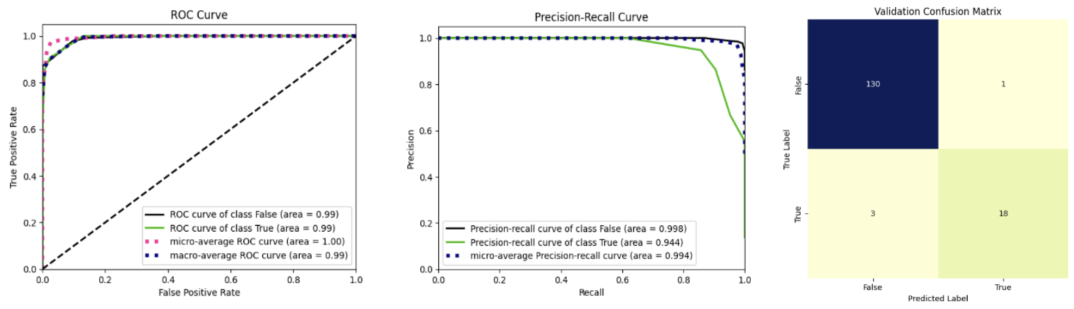
Data processing: 1) The tokenizer function is used to split the text data into individual vocabulary units. By setting the num_words value, the top num_words most frequently occurring words in the training dataset are retained. The purpose of this is to reduce the size of the vocabulary, thereby reducing the complexity and computational cost of the model. 2) texts_to_sequences are used to convert the text into numerical sequences. It traverses each word in the input text and replaces it with the integer index of the corresponding word in the vocabulary. Thus, the text is converted into sequences of integers, with each integer representing a word. This is done because neural networks cannot directly process text data. 3) pad_sequences are used to unify numerical sequences of different lengths into the same length. When processing natural language text data, the lengths of texts are usually inconsistent. However, neural networks typically require input sequences to have the same length. Padding is achieved by adding specific padding values (usually 0) at the beginning or end of the sequence. If the length of the sequence itself has exceeded the specified maximum length, pad_sequences will truncate the sequence to meet the maximum length requirement. A default GRU model is created with an embedding dim of 64 and a GRU units vector of 32 (the 64 and 32 here are just common data, and their values are not important here; their purpose is to create a GRU model, and in subsequent cross-validation, these values will be continuously modified to find the parameters most suitable for the model), and the activation function is sigmoid. The class_weight parameter in the KerasClassifier function can be used to assign different weights to each category. By assigning higher weights to minority categories, the model pays more attention to categories with higher weights during training, solving the problem of data imbalance.

Pipeline creation and Parameter explanation: To prepare for cross-validation, this pipeline contains only one step, the GRU model. Three parameters are used for selection, namely embedding dim, GRU units, and class weight. Embedding dim refers to the dimension of the vector space used when converting discrete features (such as words, characters, or other symbols) into continuous vector representations. A higher embedding dimension can capture more semantic information but may lead to greater computational complexity and require more training data. The number of GRU units represents the width of the network, i.e., the dimension of the hidden state vector. The larger the dimension of the hidden state vector, the more information the network can store and the more complex features it can capture. However, this may also lead to higher computational costs and an increased risk of overfitting. Class weight is the choice of allocating weights, which determines the ability to balance data.

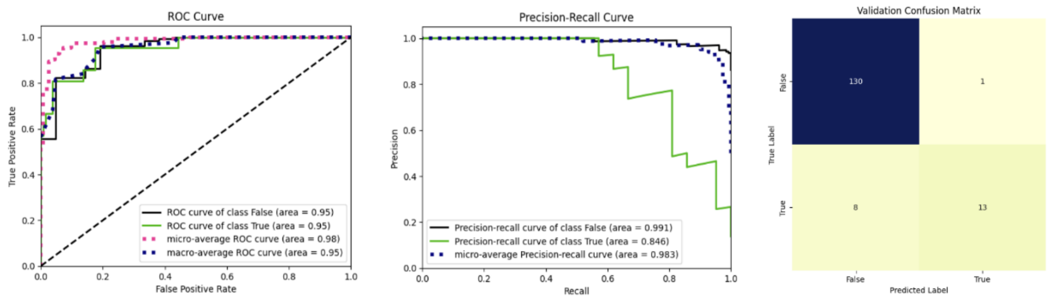
Hyperparameter selection: Initially, the embedding dim data was set to the commonly used values of [32, 64, 128]. After multiple runs, it was found that the values were mostly distributed below 64. Considering the computation time, the final selection was [8, 16, 32, 64]. Initially, the GRU units were set to the commonly used values of [32, 64, 128]. After multiple tests, it was found that the values were mostly distributed around 32. To determine a more precise value, the final choice was [8, 16, 32, 64, 128]. A loop was set for class weight. Since there are much fewer True data than False data, the weight for False data was set to 1, and the weight for True data was chosen from [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], hoping to select the weight that can best balance the data and achieve the best model. After cross-validation using the GridSearchCV function, the best combination obtained was embedding dim: 64, GRU units: 32, class weight: {0: 1, 1: 5}.

C. Results and Analysis

In the KNN model, the accuracy is 97%, and the precision is 95%. In the ROC plot, the area for both True and False is 0.99, which is relatively good. From the Precision-Recall plot, the area for False is 0.998, and for True is 0.944. The closer these numbers are to 1, the better the results. It can be seen that data imbalance has an impact on the KNN model, but the overall performance is still quite good.



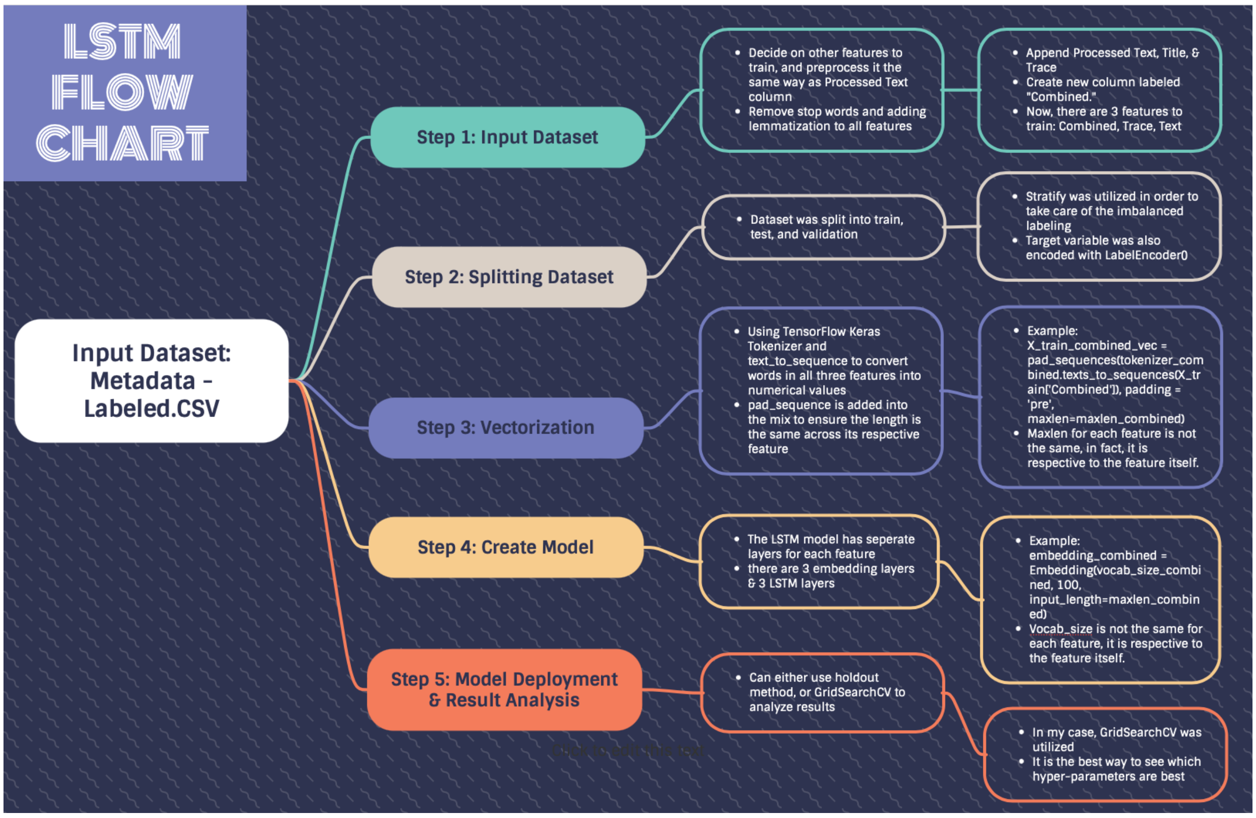
In the GRU model, the accuracy was 94%, and the precision was 93%. In the ROC plot, the area for both True and False is 0.95, which is still pretty good. From the Precision-Recall plot, the area for False is 0.991, and for True is 0.846. The closer these numbers are to 1, the better the results. It is evident that data imbalance has a significant impact on the GRU model, but the overall performance is still acceptable.



Considering the results of both KNN and GRU models, data imbalance has a noticeable effect on the training outcomes, and the limited data size further amplifies this issue. To achieve better results, it is recommended to increase the training data size and try to balance the data, which should lead to improved performance.

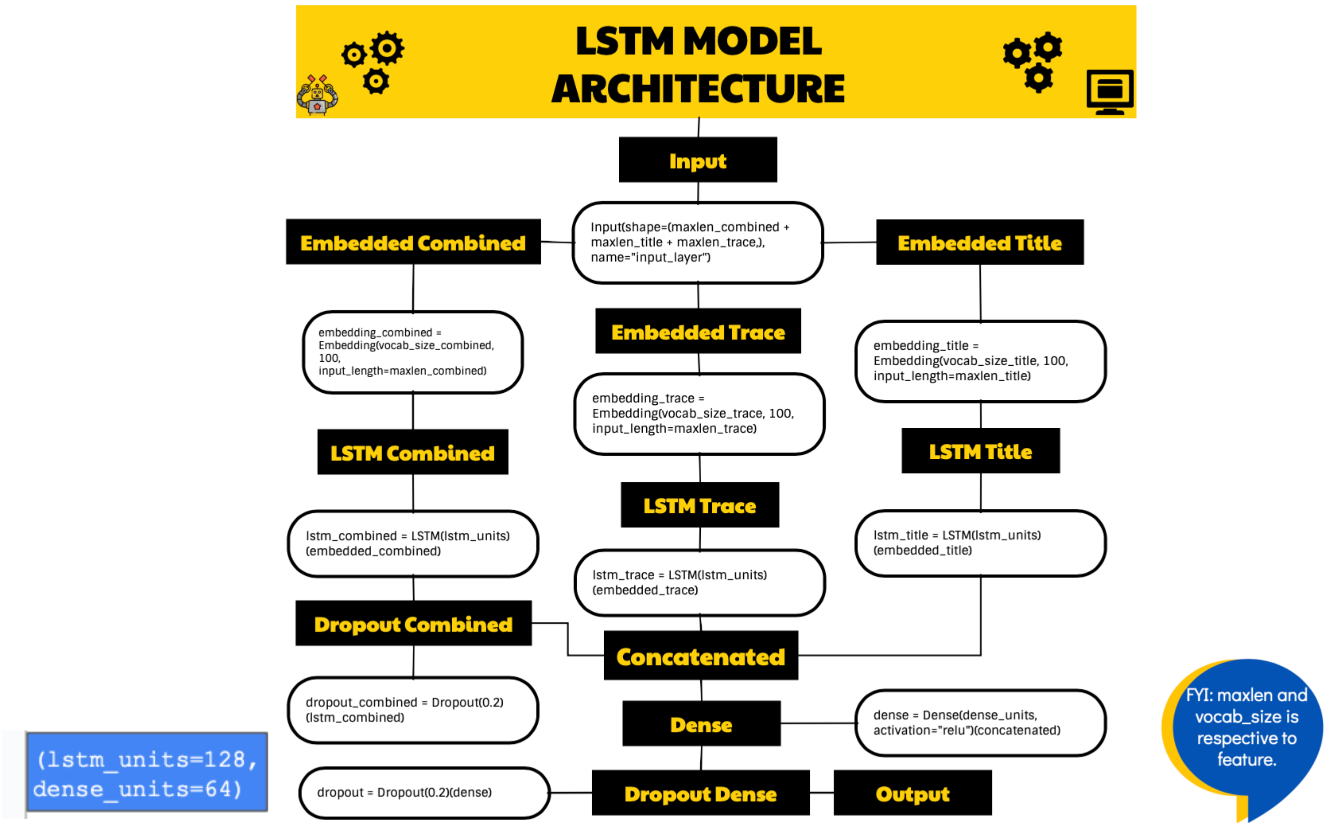
16. LSTM Model

Long Short-Term Memory network is a deep learning model which is an extension of Recurrent Neural Network. It is capable of learning long-term sequence data which is great for time-series analysis, image classification, text classification, and more. In the case for QQ tech, the LSTM model that has been constructed will implement a high-level text classification for the Los Angeles locality in category Residential Lease & Rental. There were five steps taken in the process of the creation of this model: input dataset, dataset split, vectorization, model creation, model deployment and result analysis; the figure below highlights the process step-by-step. The very first step, input dataset, includes some pre-processes before splitting the data. Selecting which features to first choose is important and needs to pertain to the task at hand and which end results is most favored or anticipated. The features selected for this model are ‘Processed Text’, ‘Title’, and ‘Trace’, plus the target variable ‘Good’. Moreover, it is important to remember to preprocess features which haven’t been already. As previously stated, ‘Processed Text’ is the very first feature to be properly cleaned when appended to the Metadata – Labeled, however, ‘Title’ and ‘Trace’ had not been. We apply the same concept to those features and add lemmatization and stop words across all features concurrently. Now that all features have been processed and cleaned, we start combining features. To further enable the model to learn patterns and sequence relationships, all three features were appended into a new feature called ‘Combined’, and ‘Title’ and ‘Trace’ were then used as separate features alongside the new feature.


LSTM FLOW CHART
Input Dataset: Metadata - Labeled.CSV
Step 1: Input Dataset
Decide on other features to train, and preprocess it the same way as Processed Text column
Remove stop words and adding lemmatization to all features
Append Processed Text, Title, & Trace
Create new column labeled "Combined."
Now, there are 3 features to train: Combined, Trace, Text
Step 2: Splitting Dataset
Dataset was split into train, test, and validation
Stratify was utilized in order to take care of the imbalanced labeling
Target variable was also encoded with LabelEncoder()
Step 3: Vectorization
Using TensorFlow Keras Tokenizer and text_to_sequence to convert words in all three features into numerical values
pad_sequence is added into the mix to ensure the length is the same across its respective feature
Example: X_train_combined_vec = pad_sequences(tokenizer_combined.texts_to_sequences(X_train['Combined']), padding = 'pre', maxlen=maxlen_combined)
Maxlen for each feature is not the same, in fact, it is respective to the feature itself.
Step 4: Create Model
The LSTM model has seperate layers for each feature
there are 3 embedding layers & 3 LSTM layers
Example: embedding_combined = Embedding(vocab_size_combined, 100, input_length=maxlen_combined)
Vocab_size is not the same for each feature, it is respective to the feature itself.
Step 5: Model Deployment & Result Analysis
Can either use holdout method, or GridSearchCV to analyze results
In my case, GridSearchCV was utilized
It is the best way to see which hyper-parameters are best

The next step includes splitting the dataset into training, testing, and validation sets. The dataset is split it into these three factors because a grid search cross-validation will occur at the very last step, as well as the holdout method to analyze the final results. During data split, it is important to ensure that the labels are not in a state of disarray as it normally is in the overall dataset… so the use of **stratify** is utilized. Stratify balances the target variable such that it is evenly proportioned in each of the data splits; also making sure the target variables are encoded 0 to 1 instead of True/False. The next step includes vectorization where every word in the selected features is then converted into numbers, simply because the model learns through numbers not words. For this step, TensorFlow Keras packages such as text_to_sequence and pad_sequences were used because the former ensures the text data is converted into numerical values (sequence integers) where “these sequences are then split into lists of tokens. They will then be indexed or vectorized” (TensorFlow). Whereas pad_sequences have the ability to ensure the correct size for each feature since maxlen is respective to the selected X values.

Now, onto the fun part: model selection, creation, and execution. Below is diagram of the LSTM model for our specific agenda as it contains all information relevant to the dataset itself; the diagram was made for Metadata – Labeled and its features.



```
Input(shape=(maxlen_combined + maxlen_title + maxlen_trace,), name="input_layer")
embedding_combined = Embedding(vocab_size_combined, 100, input_length=maxlen_combined)
embedding_trace = Embedding(vocab_size_trace, 100, input_length=maxlen_trace)
embedding_title = Embedding(vocab_size_title, 100, input_length=maxlen_title)
lstm_combined = LSTM(lstm_units)(embedded_combined)
lstm_trace = LSTM(lstm_units)(embedded_trace)
lstm_title = LSTM(lstm_units)(embedded_title)
dropout_combined = Dropout(0.2)(lstm_combined)
dense = Dense(dense_units, activation="relu")(concatenated)
dropout = Dropout(0.2)(dense)
(lstm_units=128, dense_units=64)
```

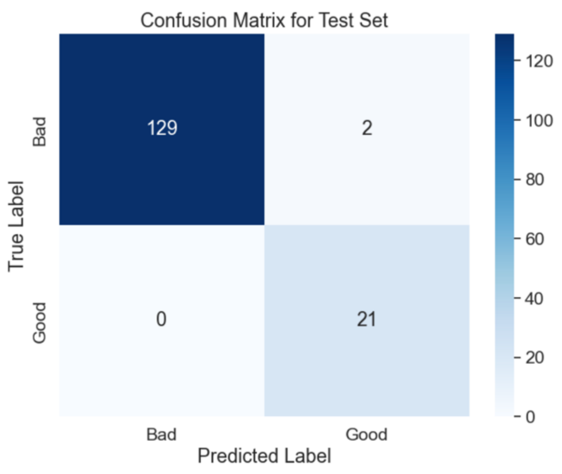
This LSTM Model Architecture contains a single input layer which takes a sequence of integer values with the length of all selected features. This input layer is fed into all three embedding layers: Embedded Combined, Embedded Trace, and Embedded Title. This layer takes in three arguments:

1. vocab_size: respective to individual feature
2. Dimension: default value of 100
3. Input_length: respective to individual feature

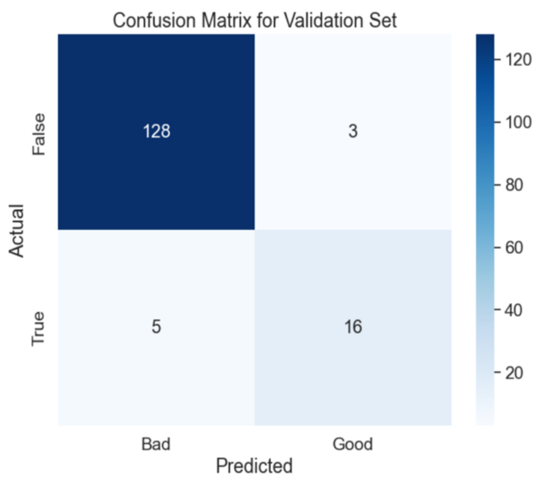
Each Embedded layer is then passed onto an LSTM layer. There are three total LSTM layers because there are three features. This layer contains two features: lstm_units and embedded feature sequence. The lstm_units are hyperparameters which is defined as 128 to improve performance. Including a lower number in this unit would mean that the model’s capability to learn long/complex sequences will not be as great since the dataset includes a vast number of sequences. Moreover, notice that the LSTM Combined layer has a dropout layer… this is to prevent overfitting which is very likely when powerful models such as LSTM-RNN are fed datasets that are not as challenging. The dropout unit is set to 0.2 simply because it gave a higher precision as opposed to 0.5. Next up, it is important to put dropout combined, lstm title and trace into a single tensor through concatenate. This allows the model to process info in each feature and decipher their relationship. The concatenated layer then goes in our fully connected dense layer where the model further learns information from all the previous features. And lastly, we apply dropout to the dense layer and then our output layer is what is used to classify between our label (y) data. After the model has been created, model execution comes next in the form of GridSearchCV and Holdout method for results and analysis. GridSearchCV is used to find the best parameters in the model, use grid_results to make predictions on test set and as well as the validation set to calculate their overall metrics such as accuracy, precision, recall, and F1 score.

Results

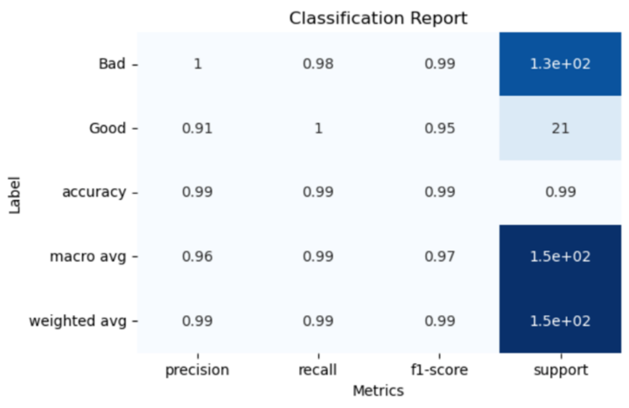
Below are the end results from the LSTM model:



Looking at the confusion matrix for the Test Set, we can see how the model performed, we see there is a total of 152 samples. And 129 out of those are true negatives meaning the model has correctly predicted the negative class, we have 2 false positives where the model has incorrectly predicted the positive class when it was actually negative, we have 0 false negative where the model has incorrectly predicted the negative class when it is actually positive, and finally 21 true positives where the model has correctly predicted the good classes. The overall number of good classes is 21, and the model had correctly identified 21 of those which indicates excellent model performance.



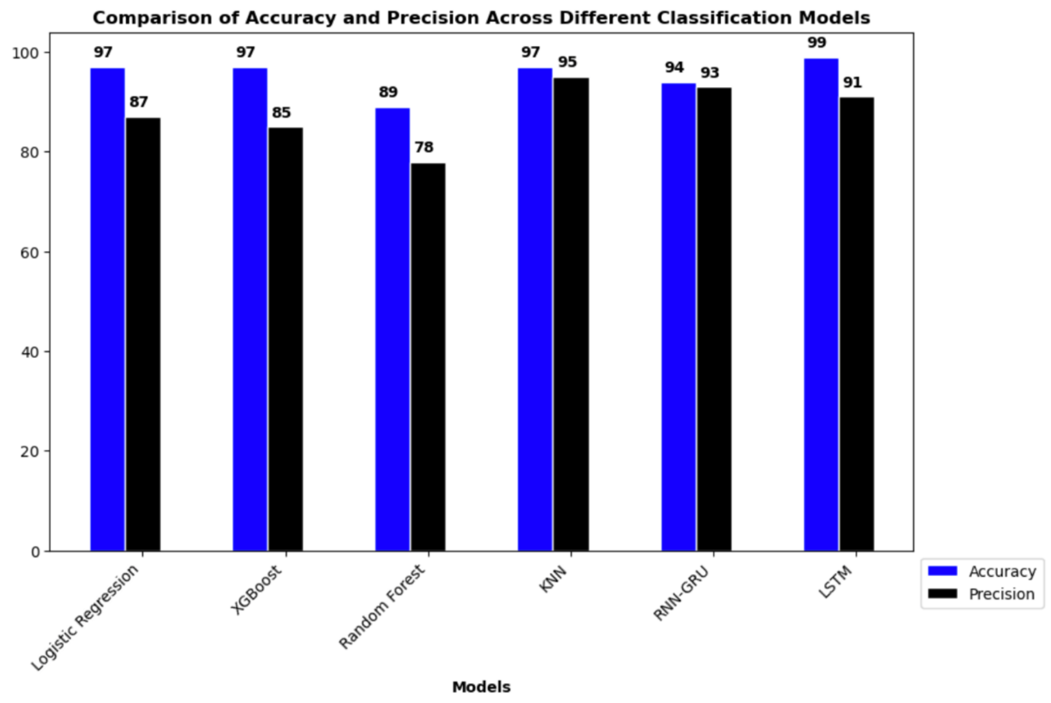
Looking at the confusion matrix for the Validation Set, the results are a bit dissimilar to our previous results. 128 samples were correctly predicted as negative class, we have 3 false positives where the model has incorrectly predicted the positive class when it was actually negative, we have 5 false negative where the model has incorrectly predicted the negative class when it is actually positive, and finally 16 true positives where the model has correctly predicted the good classes. The overall number of good classes is 21, and the model had correctly identified 16 of those which indicates the model does generalize good enough on the dataset but there is room for improvement in hyperparameter tuning or having a large-scaled data to train the model on.


Classification Report

Label	precision	recall	f1-score	support
Bad	1	0.98	0.99	1.3e+02
Good	0.91	1	0.95	21
accuracy	0.99	0.99	0.99	0.99
macro avg	0.96	0.99	0.97	1.5e+02
weighted avg	0.99	0.99	0.99	1.5e+02

This figure is a classification report for the Test Set. We can just focus on the accuracy and precision since that is what matters most. Precision is our correctly predicted good, recall is predicted good out of actual good. F1 score is the average of precision and recall. We can see the overall accuracy is 99% with 91% precision for the Good class. This figure suggests the model has a very strong performance overall, and can accurately predict between “Good” and “Bad” classes.

17. Conclusion



Given the results for each model for their Test Set, we can see that the LSTM model has the highest accuracy amongst all other models, although the highest precision is seen in the KNN model. In terms of accuracy, LSTM had outperformed the others because of its ability to retain important patterns and behaviors in a dataset while discarding information that is not needed. However, in terms of precision, LSTM came in second, first being KNN, which could be due to a multitude of reasons, one of which could be that LSTM is a more complex model and complex models are prone to overfitting when trained on a smaller dataset. All models, except Random Forest, can be considered in any text classification model that matches the data needs.

Model Advantages and Disadvantages

Model	Advantages	Disadvantages	Important parameters
Logistic Regression	**Interpretability:** • Produces coefficients for each predictor variable which, in our case, helps us understand which words or	**Limited power:** • It's a linear model which can limit its ability to capture complex non-linear relationships that	**Best hyperparameters: {'C': 100, 'max_iter': 100, 'penalty': 'l2'}**

	features are most important in determining the classification. **Simplicity:** • It's a simple and easy-to-understand model that has fewer hyperparameters to tune compared to other models such as Random Forest or XGBoost, which can make it faster to train and easier to interpret. **Efficiency:** • Logistic Regression can be computationally efficient, especially when the number of features is relatively small. This can make it a good option for text classification tasks with smaller datasets.	may be present in text data. **Imbalanced data:** • If the binary classification problem has imbalanced classes, i.e., there are significantly more examples of one class than the other, logistic regression may not perform well. • The model is optimized to minimize the overall error rate, which may result in poor performance for the minority class. **Limited ability to handle noise and outliers:** • Can be sensitive to noisy and outlier data points, that can negatively impact its performance.	
Random Forest	**Non-linear relationships:** • Can handle non-linear relationships between the features and the target variable, which can be common in text classification. Each decision tree in the ensemble can capture different non-linear relationships between the features and the target variable. **Large number of features:** • Can handle a large number of features. **Feature importance:**	**Black box model:** • It's a complex model that can be difficult to interpret. It is challenging to understand how individual features and data points affect the final predictions. This lack of interpretability can be a drawback, especially when trying to explain the model to stakeholders. **Sensitivity to noise and irrelevant features:** • Can be sensitive to noisy or irrelevant features,	**Best hyperparameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}**

	• Can provide feature importance scores, which can help us understand which features are most important in determining the classification outcome. This can be useful in feature selection or feature engineering. **Less prone to overfitting:** • Less prone to overfitting compared to a single decision tree. The ensemble of trees can help reduce the variance and improve the generalization performance of the model. **Imbalanced datasets:** • Can handle imbalanced datasets, which can be common in text classification where one class can be more dominant than the other.	which can impact the model's performance. If there are many irrelevant features or the text data contains a lot of noise, Random Forest may not perform as well as other algorithms.	
XGBoost	**Handling large datasets:** • Can handle large datasets with many features, which is common in text classification tasks. **Handling imbalanced datasets:** • Can handle imbalanced datasets, which can be common in text classification tasks where one class may be more dominant than the other.	**Data preprocessing:** • May not perform well if the text data is not preprocessed properly. For example, if the data contains a lot of noise, stopwords or irrelevant features, this may negatively impact the model's performance. **Sensitive to hyperparameters:** • Has many hyperparameters that need to be tuned carefully to achieve the best	**Best hyperparameters: {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 100}**

		performance. This can be time-consuming and require significant computing resources.	
KNN	• Easy to implement, no complex training process required, just classify based on the distance of nearest neighbors. • For small datasets, the KNN model has good performance. • The KNN model does not require pre-training, and when new samples are added to the dataset, the model updates quickly.	• For long text data, the KNN model may encounter large computational loads and slow speeds. • Choosing an appropriate K value and distance metric can be challenging. • For imbalanced datasets, the KNN model may be subject to bias and require additional processing.	**K-value(n_neighbors):** [3,5,7,10]**,** **threshold:** [0.3, 0.4, 0.5, 0.6, 0.7]**,** **n-gram**: [(1, 1), (1, 2)]
GRU	• Able to capture the contextual information and long-term dependencies in long text data. • During the training process, weight updates can help address the vanishing gradient problem. • Compared to LSTM, the GRU model has a simplified structure, fewer parameters, and higher computational efficiency.	• Selecting appropriate hyperparameters (such as embedding dimensions, the number of GRU units, etc.) can be challenging and may require multiple attempts. • For imbalanced datasets, the GRU model may need additional processing, such as adjusting class weights. • The GRU model might be prone to overfitting, especially when dealing with a small amount of data.	**Embedding Dimension:** [8,16, 32, 64]**,** **GRU Units:** [8,16,32,64,128]**,** **Class Weights:** [1,2,3,4,5,6,7,8,9,10]
LSTM	• The LSTM Model can take consider as many features as the user wants to test. As long as each feature is processed and vectorized	• Using cross-validation may take some time; approximately 10 minutes depending on epoch and batch size.	**In model**: Lstm_units: 128 Dense_units: 64 Dropout: 0.2

	properly. • This model can learn patterns and relationships; especially if it is fed a large amount of dataset.	• Each time model is run, there are different results. Setting seed does not work.	**Gridsearch:** Epochs: 10 Bachsize: 64 Cv = 2

Recommendations

Numerous enhancements can be introduced to both the website crawler and the machine learning models, which would greatly augment the efficiency of each aspect in accordance with the client's needs and purpose. For example, to further elevate the performance of our current Web Traverse Project, multi-threading could be implemented for improvement in efficiency. At present, our Breadth-First Search (BFS) traversal relies on a queue to hold web page nodes, which is a lengthy process which delays the overall performance of the project. Integrating multi-threading enables the traversal process to be segmented into smaller and more manageable portions that can be concurrently executed by several threads, thus expediting processing time and enhancing project performance. Moreover, we can delve into additional optimization methods, such as load balancing and caching to further refine the efficiency of the Web Traverse Project.

Furthermore, enhancing the performance of each machine learning model can be achieved by expanding the data collection or making use of the data synthesis technique to augment the current dataset. For further model productivity, QQ Tech can also explore a wide range of Neural Networks like Convolutional Neural Network, Bi-Directional LSTM, and Artificial Neural Network or even ensemble methods where multiple models are combined to work towards one aim. In addition, a comprehensive analysis exploring additional features from the dataset could provide valuable insights into the relationship between the extracted HTML pages and other contributing factors. It is important to more that if the Long Short-Term Memory model incorporates more than three feature variables, QQ Tech should then add the respective layers within the model to ensure precise analysis and outcomes.

18. References

Bruess, Martin. “Beautiful Soup: Build a Web Scraper with Python.” *Real Python*, 12 Mar. 2023, https://realpython.com/beautiful-soup-web-scraper-python/.

“Finding Web Elements.” *Selenium*, 22 Jan. 2023, https://www.selenium.dev/documentation/webdriver/elements/finders/.

“Text Classification with an RNN : Tensorflow.” *TensorFlow*, TensorFlow, https://www.tensorflow.org/text/tutorials/text_classification_rnn.

“Tf.keras.preprocessing.text.tokenizer : Tensorflow V2.12.0.” *TensorFlow*, TensorFlow, https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer.

“Web Crawler in Python.” *TopCoder*, 25 Jan. 2021, https://www.topcoder.com/thrive/articles/web-crawler-in-python.

Disclaimer: first three and last link were used to gain knowledge for the completion of this project.