Automated Tagging of Stack Exchange Data Science Posts Using Natural Language Processing

A Capstone Project in Fulfillment of
Springboard's Data Science Career Track Program

By

Laura Elliott

And In Association with

Ajith Patnaik

Springboard Mentor

Table of Contents

1. PROBLEM DEFINITION

For many question answering websites (Stack Overflow, StackExchange, Quora, Answerbag, Yahoo, WikiAnswers, etc.), a plethora of questions on many different subjects are submitted each day. On Stack Exchange, each question requires at least one tag to provide context for and to categorize that question. These tags are an important ranking factor in most of these website's search algorithms, so it is important for them to be as accurate as possible in order to ensure the best customer experience. The more quickly and accurately the tags can be assigned, the better the service can be about sending the right question to the right set of people for answering and the quicker the turnaround time for an answer to that question. The tags also provide the users with the ability to sort and filter through questions and answers on their topic of concern.

As of 2011, Stack Exchange used a 2-step process to assign tags (https://stackoverflow.blog/2011/08/09/improved-tagging/#:~:text=But%20how%20do%20you%20determine,what%20you've%20typed%20so%E2%80%A6&text=%2DFounder%20(Former)-,Every%20Stack%20Exchange%20question%20is%20required%20to%20have%20at%20least,%2C%20order%2C%20and%20find%20questions.). First, a recommender engine (tag completer) is used to suggest tags to the question asker based on the question text. Then the question asker chooses the best tags (minimum of one, maximum of five). A set of guidelines is provided. But tag selection can be challenging for inexperienced users and poor tag choice can affect the user's overall experience. With a site launch only 7 years ago, a subject that is gradually increasing in popularity and a total of only 100k users to date, it is fair to say that many Stack Exchange Data Science users are relatively inexperienced.

2. PROJECT OBJECTIVE

The objective of this capstone is to build an algorithm employing natural language processing techniques to automatically predict the tags using the text in the questions themselves, thereby reducing the need for human intervention and making the process of answering questions more efficient.

The client could be a Stack Exchange Data Science site developer (or other question answering entity in the Machine Learning and Data Science domain) who is looking for a faster and more accurate way to automatically tag questions. Based on the results of this analysis, the client may decide to deploy this new algorithm for auto-tagging questions in order to improve the efficiency of his team and the customers' experience.

The algorithms used for the project are supervised multi-classification learning types, because each question in the dataset is labelled with at least one tag and the goal is to identify the most pertinent tags from a list of multiple tags.

In order to identify the best algorithm, three different types of text to numeric transformations will be used and three different machine learning algorithm types will be employed. The different models / transformations will be compared against one another using a ROC - AUC scoring

technique (Area Under the Receiver Operating Characteristic Curve) to statistically summarize results for all labels and therefore determine the most accurate method overall.

3. DATA ACQUISITION

Questions and tags from the Data Science Stack Exchange database were used for this analysis. The Data Science Stack Exchange website is a question and answer site for Data Science professionals, Machine Learning specialists, and those interested in learning more about the field. It is one of 176 Stack Exchange Q&A communities (which includes the largest and most well-known site for programmers - Stack Overflow). The website link is:

https://datascience.stackexchange.com/

The dataset was extracted from the Data Science Stack Exchange database on July 22, 2020 via the Stack Exchange data dump site using this link:

https://archive.org/details/stackexchange

These data were archived on May 31, 2020, so contain information up to that point in time.

The data is formatted as a separate archive consisting of XML files zipped via 7-zip using bzip2 compression. Both the datascience.meta.stackexchange.com.7z and datascience.stackexchange.com.7z files were downloaded and extracted. The archive includes Badges, Comments, Posts, PostHistory, PostLinks, Tags, Users and Votes. The entire data schema is provided in Appendix 1. These data can be used for various use cases: tag prediction, automated question answering, future question status prediction (upvoted, downvoted, closed), and prediction of length of time to answer.

For preliminary examination of the xml files, a free application called XML ValidatorBuddy was used to view the file contents and structure. This application was particularly helpful for reviewing the larger xml files which could not be opened with other text readers due to size limitations.

The Tags.xml file in the meta archive was quite useful in evaluating the distribution and number of tags. The Posts.xml (79 MB) file contained within the datascience.stackexchange.com.7z contained all the necessary information for the analysis. Therefore, no joining of separate xml files was required.

The following features from the Posts.xml file were loaded for the analysis.

Features Name	*Data Type*	*Description*
`Id`	int	unique post identifier
`PostTypeId`	tinyint	1 = question; 2 = answer

Features Name	*Data Type*	*Description*
`CreationDate`	datetime	Date of post
`Score`	int	sum of upvotes - downvotes
`ViewCount`	int	count of number of views
`Body`	nvarchar	question / answer body text
`OwnerUserId`	int	id of owner/user
`LastActivityDate`	datetime	date of last post activity
`Title`	nvarchar	brief description of question
`Tags`	nvarchar	comma-separated list of the tags associated with the question
`AnswerCount`	int	count of the number of answers
`CommentCount`	int	count of the number of comments
`FavoriteCount`	int	count of the number the post has been marked as a favorite
`ClosedDate`	datetime	date the post was closed
`ContentLicense`	nvarchar	description of content license

Table 1: Feature Description from the Posts.xml file; note that question title is also included but this was not used for tag prediction; the goal is to predict tags from the question text only.

4. DATA WRANGLING

The following sections outline the steps involved in the data wrangling process.

1. Conversion of XML file to Pandas DataFrame

The initial process of converting the Posts.xml file into a pandas dataframe was conducted in the "CP2_01_Convert XML to DF.ipynb" Jupyter Notebook. The following pseudocode outlines the process:

1. Load and Parse the Posts.xml file using the xml.dom.minidom package.
2. Test whether the parsing worked by getting a sample of items from the file.
3. Create a header row from the items in each row of the xml; `df_cols = ['Id','PostTypeId','CreationDate','Score','ViewCount','Body','OwnerUserId','LastActivityDate','Title','Tags','AnswerCount','CommentCount','FavoriteCount','ClosedDate','ContentLicense']`

4. Create an empty list to append data to.
5. Write a for loop to extract all items from each row of the xml making sure to write an N/A for those fields that are empty.
6. Create a pandas dataframe with the extracted data and headers.
7. QC the dataframe and export as a pickle file.

Input is the original Posts.xml file from the Stack Exchange data dump. Output is a pandas dataframe called posts_df and associated pickle file called posts_df_09112020.pickle. The resulting posts_df dataframe consisted of 51395 rows and 15 columns (features listed in the table above).

For this notebook, only three libraries were used: xml.dom.minidom, pandas, and pickle.

2. Data Conditioning, Filtering, and Cleaning

Additional conditioning and cleaning steps were performed in the "CP2_02_Load DF Clean and Analyze.ipynb" Jupyter notebook. The following pseudocode outlines the preliminary filtering and cleaning steps :

1. Load the posts_df_09112020.pickle file created in the first Jupyter Notebook.
2. Examine contents to assure everything transferred properly.
    a. Identified some cells filled with only white space. These were replaced with nans.
3. Filter down to just questions (PostTypeId = '1')
4. Convert objects to appropriate integer and date dtypes (text attributes Body, Title, and Tag were converted in later steps).
5. Check for missing values.
6. Check for and remove duplicates.
7. Preprocess tags - remove html tags and convert to string.
8. …(the rest of steps in the notebook are EDA steps - see later section)

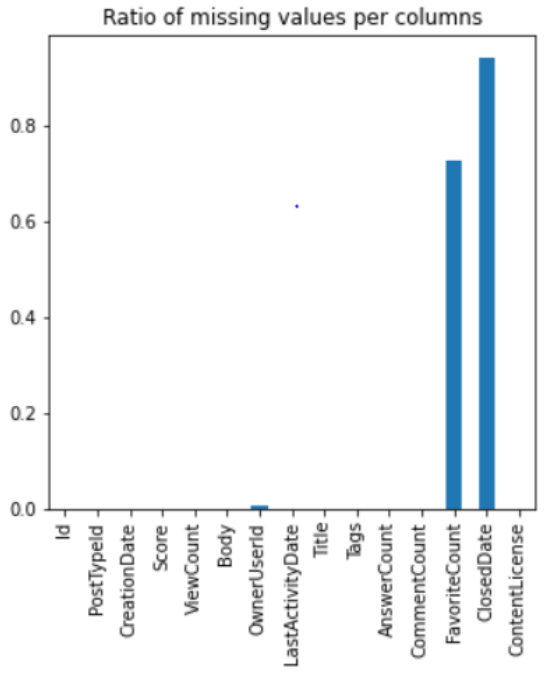
Input is the pandas dataframe called posts_df and associated pickle file called posts_df_09112020.pickle from the first Jupyter Notebook.

For these steps, the following libraries were used: pandas, pickle, numpy and collections
For visualizations, the following libraries were used: matplotlib.

Some blank fields were identified in the dataframe which had been left from the extraction script. These were replaced with nans. Then the dataframe was filtered down to just the question rows (PostTypeID = 1), leaving a total of 24363 rows.

1. *Check for Missing Data*

The following bar chart and table describe the number of missing values in each column. The chart is shown as a ratio of missing to total values per column.




```
Int64Index: 24363 entries, 0 to 51393
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Id                24363 non-null  object
 1   PostTypeId        24363 non-null  object
 2   CreationDate      24363 non-null  object
 3   Score             24363 non-null  object
 4   ViewCount         24363 non-null  object
 5   Body              24363 non-null  object
 6   OwnerUserId       24248 non-null  object
 7   LastActivityDate  24363 non-null  object
 8   Title             24363 non-null  object
 9   Tags              24363 non-null  object
 10  AnswerCount       24363 non-null  object
 11  CommentCount      24363 non-null  object
 12  FavoriteCount     6710 non-null   object
 13  ClosedDate        1422 non-null   object
 14  ContentLicense    24363 non-null  object
dtypes: object(15)
memory usage: 3.0+ MB
```

Fig 1: Ratio of missing to total values in each column (plotted with matplotlib) and results of dataframe.info query providing a count of non-null values.

The results indicate that the critical columns have no missing values. Only the OwnerUserId, FavoriteCount and ClosedDate columns have missing values. The missing values in the latter two features are expected, since not all question posts will be identified as having a Favorite answer and not all will be Closed. The OwnerUserId has a few missing values, but was not used in the analysis.

2. *Check of Duplicates*

The dataframe was checked for duplicates of both the question (Body) and Title and 10 duplicate entries were found. Two of the duplications are shown below.

Id	PostTypeId	CreationDate	Score	ViewCount	Body	OwnerUserId	LastActivityDate	Title	Tags	AnswerCount	CommentCount	FavoriteCount	ClosedDate	C
461	1	2014-06-18 22:10:58.497	12	1336	<p>There's this side project I'm working on wh...	986	2017-11-06 08:07:09.260	Preference Matching Algorithm	<bigdata><text-mining><recommender-system>	2	0	2	NaT	
477	1	2014-06-18 22:15:43.820	2	561	<p>There's this side project I'm working on wh...	986	2014-06-19 10:30:43.993	Preference Matching Algorithm	<algorithms>	1	3	NaN	2014-06-26 05:31:41.193	
2313	1	2014-10-20 06:38:16.490	5	1156	<p>Where is the difference between one-class, ...	4717	2015-03-03 14:21:51.033	Machine Learning - Where is the difference bet...	<machine-learning><data-mining><classification...	2	2	5	NaT	
2316	1	2014-10-19 21:03:35.390	1	47	<p>Where is the difference between one-class, ...	4717	2014-10-20 10:38:59.437	Machine Learning - Where is the difference bet...	<machine-learning><classification><data-mining>	0	1	NaN	2014-10-21 14:09:06.617	

Fig 2: Partial results of check for duplication in Title and Body.

The results illustrate that these duplicates were created by the same owner but have slightly different creation dates and only one of the grouped pairs has a closed date. Additionally Score, LastActivityDate, Tags, and ViewCount, AnswerCount, CommentCount, and

FavoriteCount are different. For this reason, the second entry with lower counts and closed dates were removed. Removal of these duplicates reduced the total row count of the dataframe from 24363 to 24353 entries.

3. Preprocess Tags

As noted in Figure 2 above, each of the tags in the Tags column are separated by html <> characters. These were removed and the text was converted to a space delimited string in preparation for further analysis.

3. Preprocess Body (Questions)

Many cleaning steps were required to condition the question text ('Body' attribute) for analysis. These steps were performed in a third Jupyter notebook "CP2_03_Clean_Body_Text.ipynb".

The following pseudocode outlines the steps taken:

1. Load the pickled pandas dataframe from 02 notebook
2. Examine the feature 'Body' (which are the questions) to identify cleaning tasks
3. …(Feature Creation Step)
4. ...(Feature Creation Step)
5. Cleaning tasks:
    a. Remove code snippets
    b. Remove html formatting
    c. Expand contractions
    d. Language detection to make sure everything is in English
    e. Remove special characters
    f. Simple lemmatization
    g. Named Entity Recognition Analysis
    h. Part Of Speech Tagging Analysis
    i. Convert to lowercase
    j. Remove stop words
6. Export the dataframe with cleaned text and additional features for further analysis

Input is the pandas dataframe called questions_df and associated pickle file called questions_df_09252020.pickle from the second Jupyter Notebook.

The 'textwrap' library was used to format the text data for increased readability. For cleaning the body text, the following libraries were used: Beautiful Soup, contractions, string, unicodedata, and nltk. For language detection, the 'fasttext' library was used. For named entity recognition, the spacy module was used.

To identify cleaning tasks, a sampling of the questions were printed for examination. Figure 3 shows a few examples. Please refer to the Exploratory Data Analysis section for more information about HighestRank and LowestRank questions mentioned in these examples.

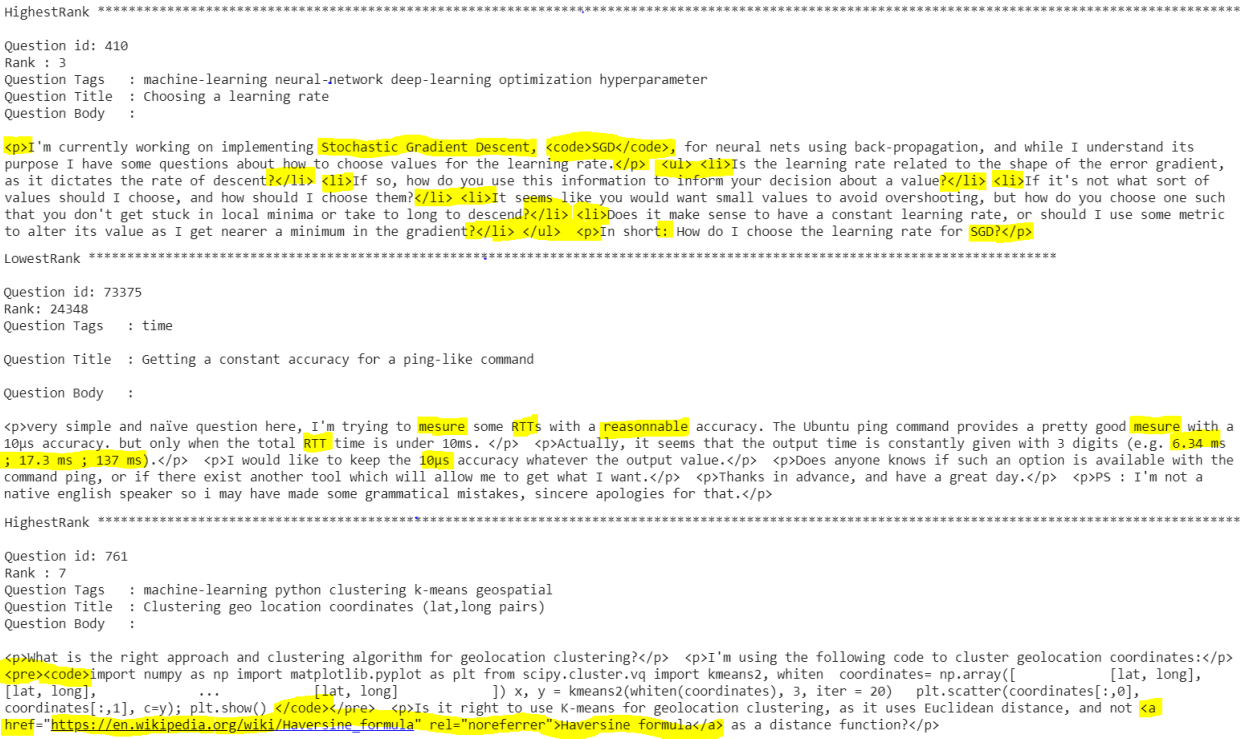


Fig 3: Question examples showing the raw question "Body" text, with various text highlighted to show important items identified for cleaning. Text was formatted using 'textwrap' for increased readability.

The first thing that stands out as an issue requiring cleaning is the html formatting, such as the   and <li> pairs that denote paragraph and line formatting. In some cases, text is bolded for emphasis. There are also hyperlinks and code snippets that tend to clutter the overall question but are provided as examples or reference within the question body. Standard special character removal and contraction expansion is also required.

Note in the examples shown that oftentimes more than one question is asked within the Body or sometimes no questions is asked at all. Question length can be quite variable also. Therefore, prior to performing the cleaning steps, certain features were extracted from the question body (such as the number of code snippets, number of question marks, body text length, and number of bolded text items).

It appears that acronyms like LDA, PCA, SGD, RTT are important to recognize, which indicates that we should attempt a Named Entity Recognition and Part of Speech tagging process as a part of the text processing.

Some misspellings were also identified, but have been kept in place. Some examples are shown above. In one case, "but" was misspelled as "bot", the latter being an important word with a very different meaning in the data science world.

Beautiful Soup was then used to identify and remove the code snippets and then to remove the remaining html formats. Figure 4 shows an example of a question before and after this process.

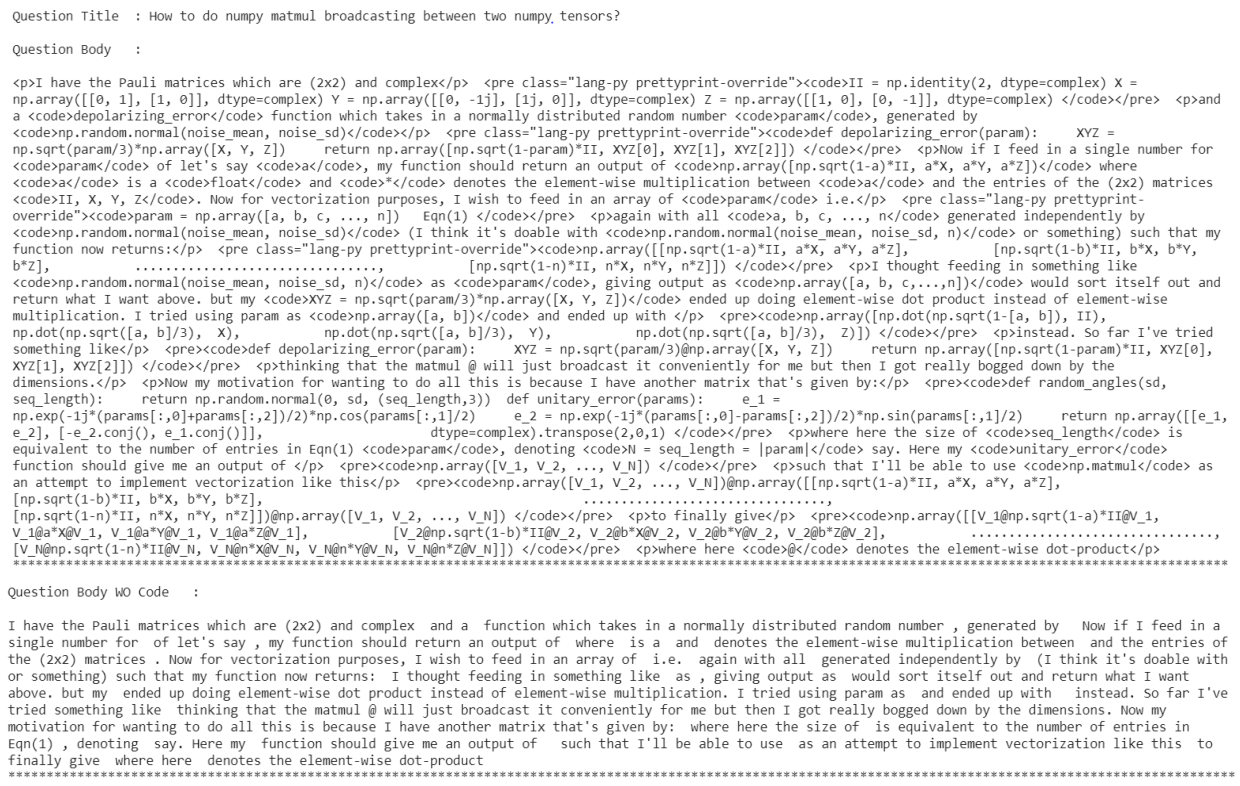


Fig 4: Example question before and after removal of code snippets and html formatting.

The next step was to expand contractions, using the contractions module. Figure 5 shows a before and after example of this process.

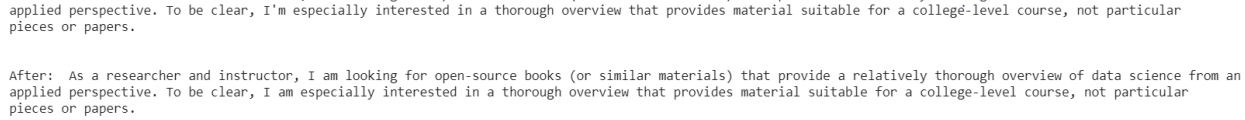


Fig 5: Example question before and after contraction expansion.

The next step was a check to see if all questions are in English, using Facebook's fasttext library and their prebuilt model (lid.176.bin). Only three questions returned as a different language (fr,ja,kn) but a review of these rows indicates that the processing of the questions done in previous steps reduced them to just one word or character, and were therefore producing erroneous language results. A preliminary pass of the English language detection routine prior to cleaning the code snippets identified some string examples within the text that were in an alternate language, but these were eliminated by first removing the code snippets. To summarize, all text from the Stack Exchange data dump is in English, although some examples provided within the question code snippets were in a different language. These examples were all removed as a part of the code snippet removal process.

The next step was to remove all special characters in the text. In order to identify what characters should be removed, further analysis was performed using the nltk dictionary 'punct_only'. There are 329 unique special characters in the Body text. Most of the special characters are punctuation and symbols and these were removed:

!"#$%&\'()*+,./:;-<=>?@[\\]^_`{|}~

However Greek characters such as α, β, θ, σ, ⅄,Ψ, and μ were maintained because they have special significance in the data science field.

NLTK's WordNetLemmatizer was then used to perform a simple lemmatization of the text. The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. As opposed to stemming which simply chops off the ends of words, lemmatization refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma (https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html#:~:text=Lemmatization%20usually%20refers%20to%20doing,is%20known%20as%20the%20lemma%20.). Sample results of the special character removal and lemmatizing process are shown in Figure 6 below.

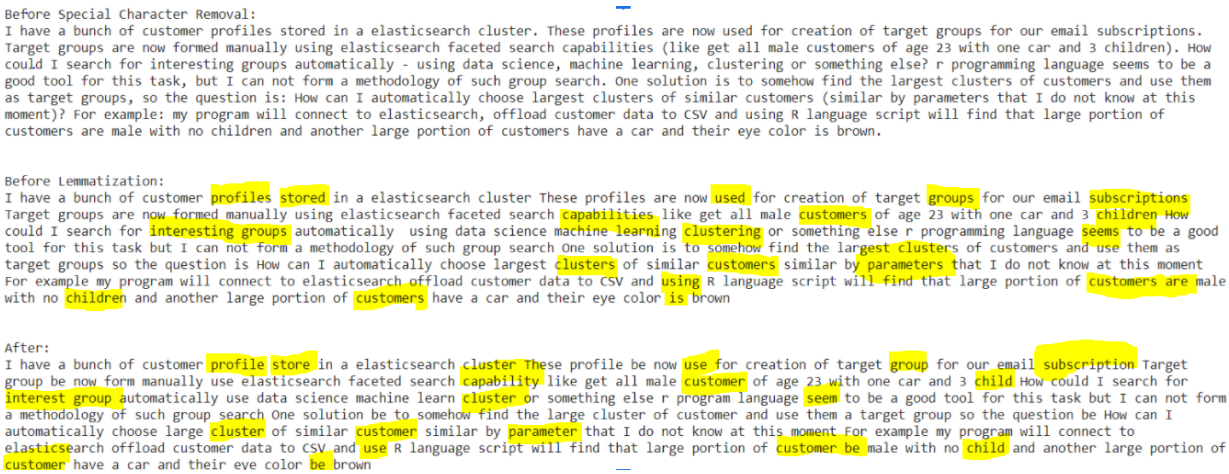


Fig 6. Sample showing questions text before and after special character removal and lemmatizing.

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. As opposed to stemming which simply chops off the ends of words, lemmatization refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma (https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html#:~:text=Lemmatization%20usually%20refers%20to%20doing,is%20known%20as%20the%20lemma%20.).

An attempt was made to recognize, extract, and label named entities using Spacy's Named Entity Recognition and corresponding corpus 'en_core_web_sm'. Running this process, even with this small dataset takes quite a bit of time, and while effective at recognizing and extracting the named entities, the labelling results were poor. Figure 7 below shows that terms like "Libsvm", "Liblinear", and "MapReduce" are identified erroneously as a Geopolitical Entity (GPE), an Event, and a Person, respectively. This is a result of the fact that the Spacy algorithm is trained on the general corpus but our dataset is quite technical and specific to the world of Machine Learning

and Data Science. This leads to a general bias in the results where the entity tags are relevant in the context of the general corpus. Further work to improve the labelling results was deemed beyond the scope of this capstone project.

```
Before:  I use Libsvm to train data and predict classification on semantic analysis problem But it have a performance issue on largescale data because semantic analysis
concern ndimension problem Last year Liblinear be release and it can solve performance bottleneck But it cost too much memory Is MapReduce the only way to solve
semantic analysis problem on big data Or be there any other method that can improve memory bottleneck on Liblinear


After:  [['Libsvm', 'GPE'], ['Last year', 'DATE'], ['Liblinear', 'EVENT'], ['MapReduce', 'PERSON']]
```

Fig 7. Sample showing poor labelling results for technical data science terms during the NER process.

Nouns were also recognized and extracted using NLTK's pos_tag module.

```
Before:  I use Libsvm to train data and predict classification on semantic analysis problem But it has a performance issue on largescale data because semantic analysis
concerns ndimension problem Last year Liblinear was release and it can solve performance bottleneck But it cost too much memory Is MapReduce the only way to
solve semantic analysis problem on big data Or are there any other methods that can improve memory bottleneck on Liblinear


After:  ['data', 'classification', 'analysis', 'problem', 'performance', 'issue', 'data', 'analysis', 'concerns', 'ndimension', 'problem', 'year', 'Liblinear', 'release', 'performance'
```

Fig 8. Same sample as in Figure 7 showing results of POS tagging for nouns. Some, but not all, named entities are recognized also through this process.

The next cleaning step for the question text was to convert all to lowercase. It was determined via experimentation that performing the lowercase step prior to NER and POS tasks produced less satisfying results. The final step of text cleaning was to remove stop words. The following words were removed from the stopwords list since they are meaningful for data scientists - 're', 'r', and 'q'. A before and after sample of the lowercase conversion and stop word removal is shown below.

```
Before LC Conversion:
We create a social network application for eLearning purpose it be an experimental project that we be research on in our lab It have be use in some case study
for a while and the data in our relational DBMS SQL Server 2008 be get big it be a few gigabyte now and the table be highly connect to each other The
performance be still fine but when should we consider other option Is it the matter of performance


Before StopWord Removal:
we create a social network application for elearning purpose it be an experimental project that we be research on in our lab it have be use in some case study
for a while and the data in our relational dbms sql server 2008 be get big it be a few gigabyte now and the table be highly connect to each other the
performance be still fine but when should we consider other option is it the matter of performance


After:  create social network application elearning purpose experimental project research lab use case study data relational dbms sql server 2008 get big gigabyte table
highly connect performance still fine consider option matter performance
```

Fig 9. Sample showing results of lowercase conversion and stop word removal.

Final Output is a pandas dataframe called questions_df consisting of 24363 entries and 63 columns. The associated pickle file exported is named questions_df_ner_results_10282020.pickle.

5. EXPLORATORY DATA ANALYSIS

4. Tag Distribution Analysis

The following pseudocode outlines the process used to analyze the tag distributions. This analysis was important to determine on which tags to focus our prediction power. These steps were performed in the last portion of the "CP2_02_Load DF Clean and Analyze.ipynb".

7. …(Previous steps were data conditioning and cleaning steps)
8. Get a list of all the Tags and Frequency Counts
9. Exploratory analysis / visualization of tags related to their frequency
10. Exploratory analysis / visualization on other features such as ViewCount and Score
11. Create TopTagsList based both on frequency and ranking by ViewCount and Score
12. Write out the filtered data frame with new features

For these steps, the following libraries were used: collections, Beautiful Soup, nltk, and re. For visualizations the following libraries were used: matplotlib, seaborn, and wordcloud.

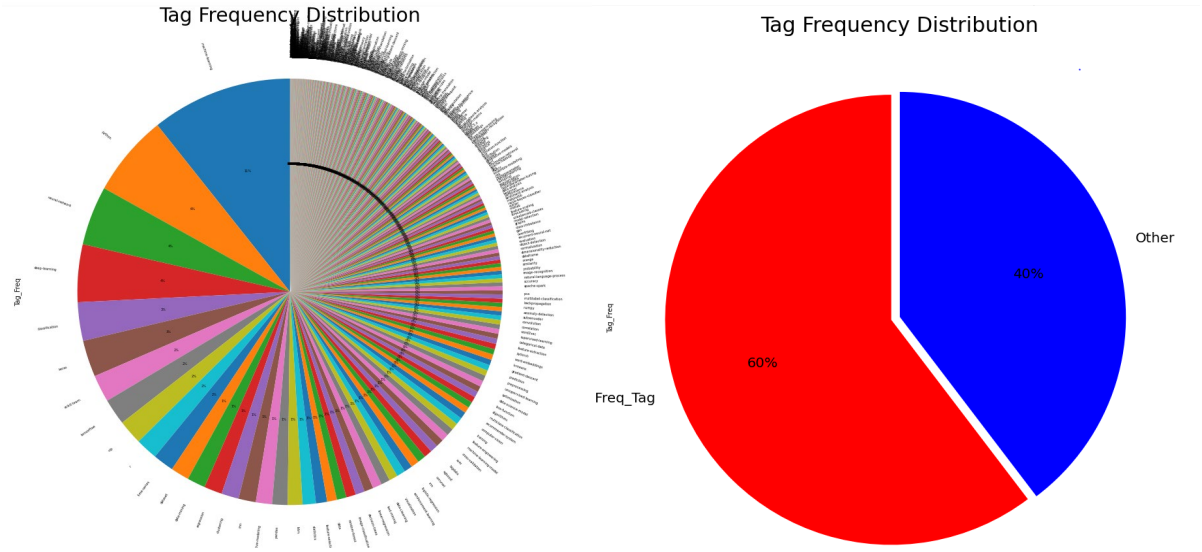


Fig 10: Pie chart showing the frequency distribution of all of the tags. There are a total of 590 unique tags in the dataset, but the top 32 tags represent 60% of the data.

As shown in Figure 10, of a total of 590 unique tags, the top 32 most frequent tags encompass 60% of the total data.

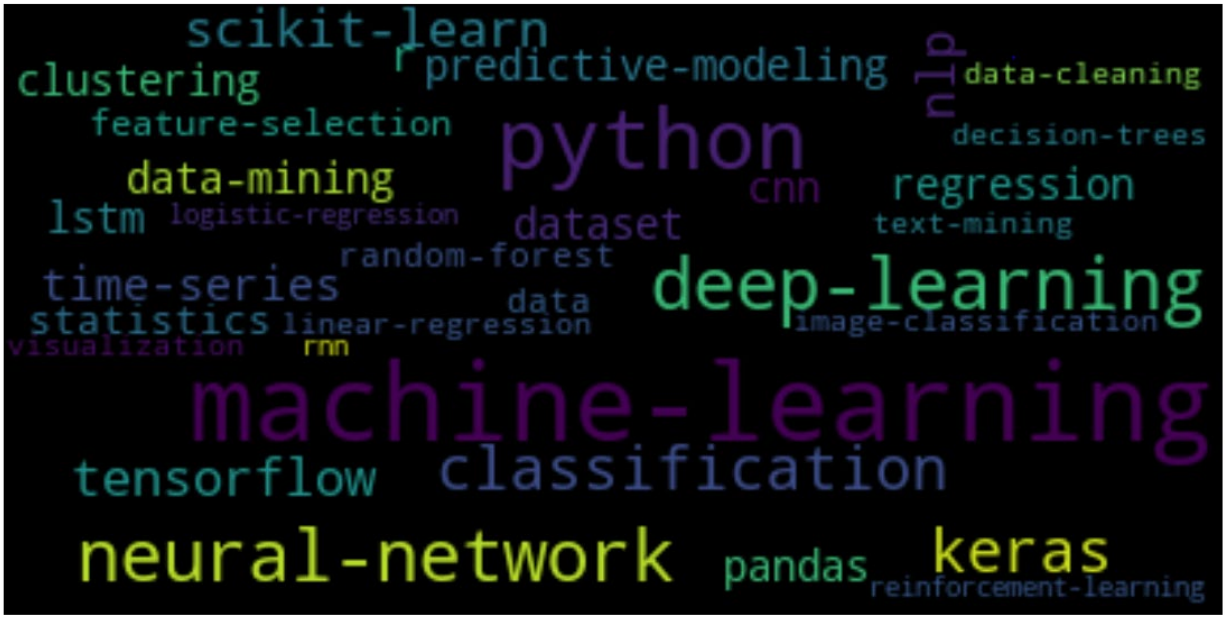


Fig 11: Wordcloud of top 32 tags, showing the popularity of machine and deep learning topics.

Figure 11 shows the top 32 tags as a wordcloud. You can see that machine-learning, deep-learning, and neural-networks and their associated software and libraries such as keras and tensorflow are the hottest topics in the forum.

Additional exploratory analysis was also performed on a newly created feature, the number of tags per question (TagCount), ViewCount, and Score, and the correlations between them.

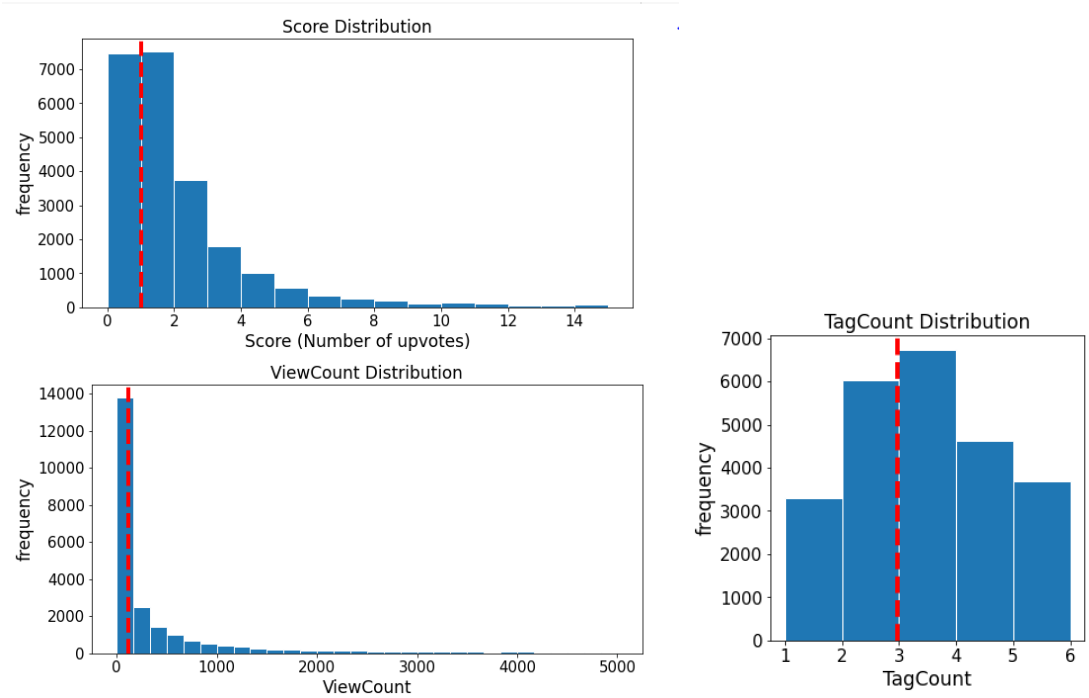


Fig 12: Frequency Distribution of Score, ViewCounts, and TagCount.

	Score	ViewCount	TagCount
count	24353.000000	24353.000000	24353.000000
mean	2.005872	1387.329651	2.974705
std	5.433220	6895.850492	1.257866
min	-6.000000	2.000000	1.000000
25%	0.000000	36.000000	2.000000
50%	1.000000	114.000000	3.000000
75%	2.000000	578.000000	4.000000
max	214.000000	234355.000000	5.000000

Table 2: Feature Statistics

As would be expected for count features and shown in Figure 12 and Table 2, both Score and ViewCount have a skewed poisson distribution with mean values of 2 and 1387 respectively. TagCount has a normal distribution with min of 2, and maximum number of tags per question of 5.

Further exploratory analysis was performed to identify whether the questions with top tags have higher Scores, ViewCounts or TagCounts on average than the others. The questions were split into 2 groups, those with tags on the TopTagsList and those without, and means for these attributes were calculated for each group. The results are shown in Figure 13 below.

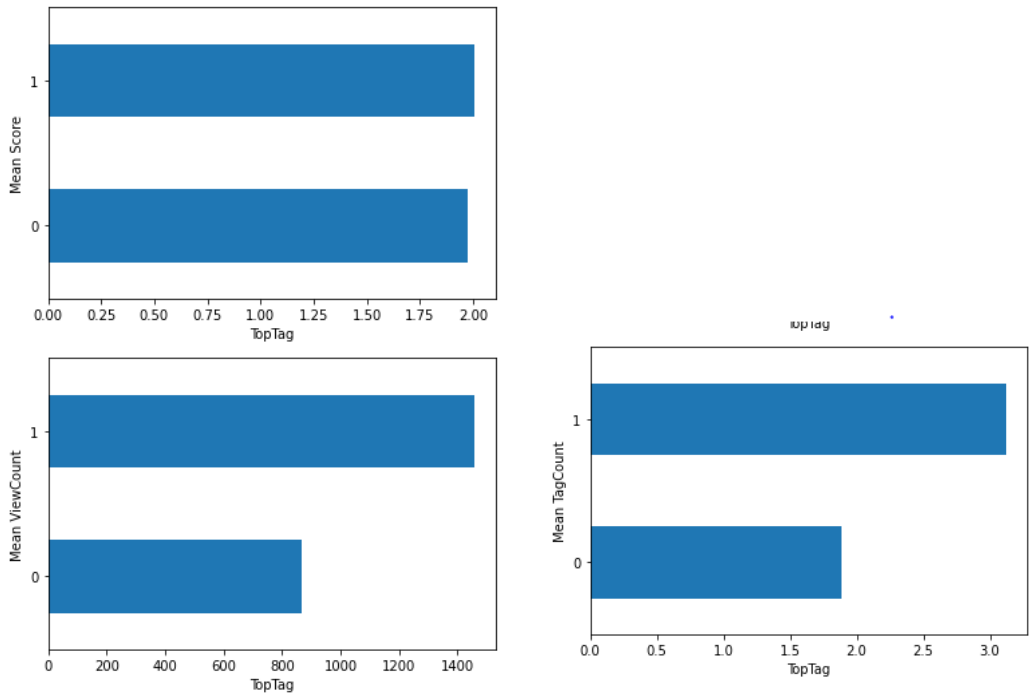


Fig 13: Mean Score, ViewCount, and TagCount by TopTag Grouping

The results indicate that the mean difference of the Score between the questions with Top Tags and those without is statistically insignificant; however the ViewCounts and TagCounts are statistically significant being almost twice the value where questions are tagged with one or more of the top 32 most frequent tags.

Based on the strong correlations between these latter two attributes and the top tags, the tags were further high-graded, and re-ranked using not only the tag frequency but also the ViewCount and TagCounts attributes. Rank order was determined by tag frequency + ViewCount + TagCount.

During this analysis, it became clear that use of all tags (up to 5 per question) produced more scatter and variation in the TopTag results. Using only the first tag for each question and excluding all the remaining tags (plus rank ordering) produced a list of 33 Top Tags that comprised 88% of the data (as opposed to the initial 60%). It was therefore decided to use only the first (and most frequent) tag for each question for tag prediction.

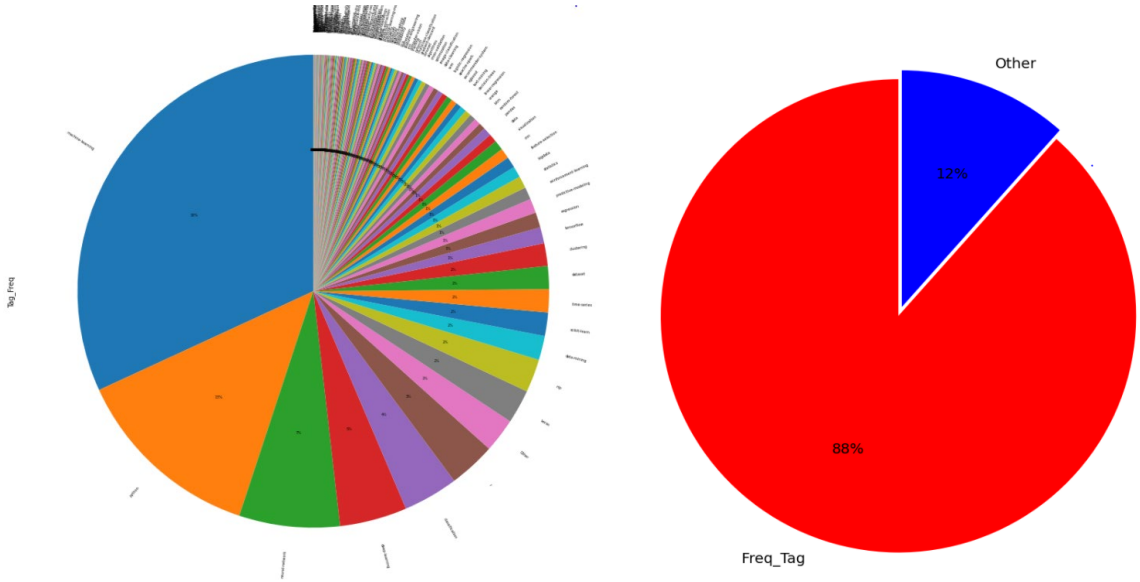


Fig 14: Pie chart showing the frequency distribution of the ranked ordered first tags. There are a total of 349 unique first tags in the dataset, but the top 33 tags represent 88% of the data.

As shown in Figure 14, of a total of 349 unique first tags, the top 33 most frequent tags now encompass 88% of the total data.

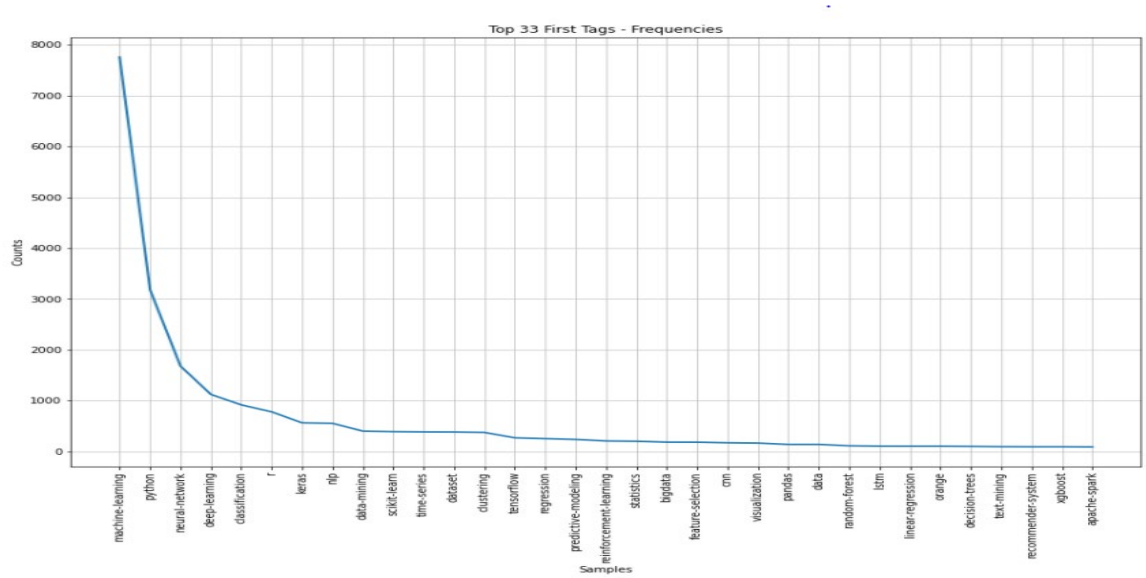


Fig 15: Revised Frequency distribution of the ranked ordered first 33 tags.

Figure 15 shows the frequency distribution of the revised 33 first top tags to be used for the prediction. All other tags were renamed to 'OTHER'.

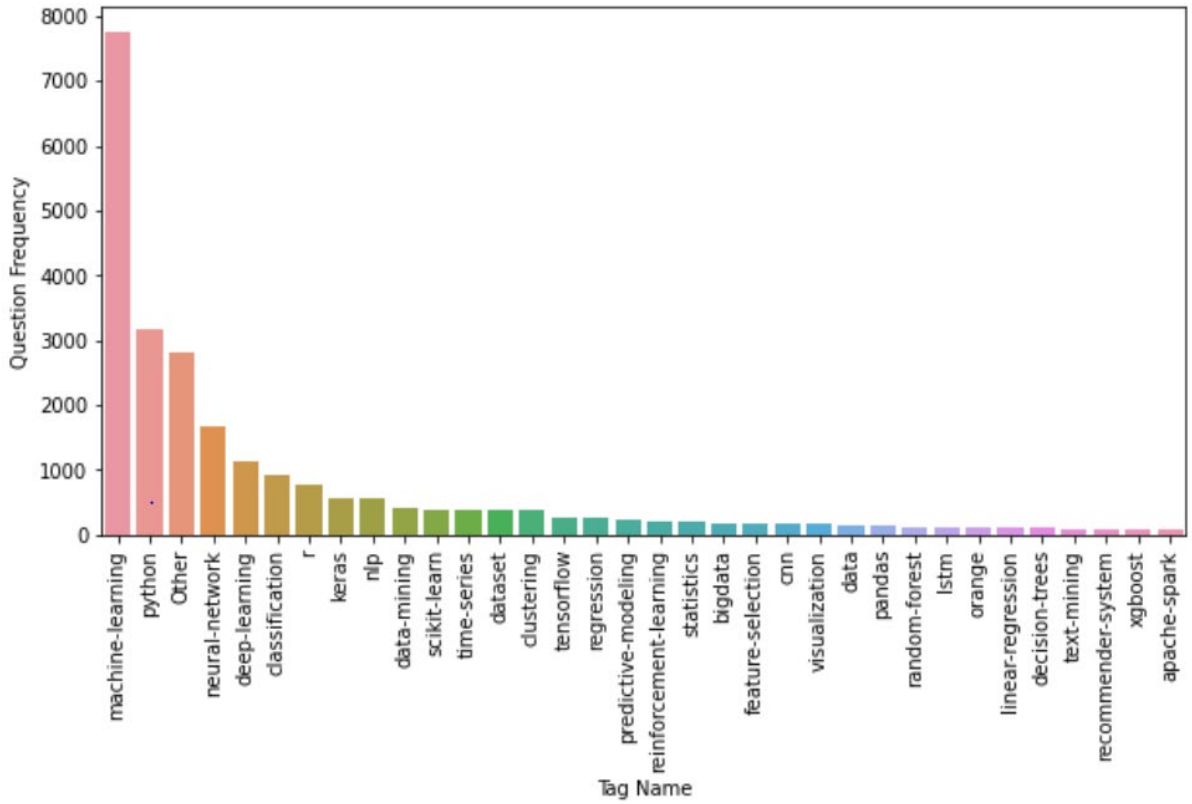


Fig 16: Bar chart showing the frequency distribution of the ranked ordered first 33 tags.

The results are a highly imbalanced distribution, with the 'machine-learning' tag accounting for the majority of question tags.

Output is a pandas dataframe called questions_df consisting of 24363 entries and 40 columns. The associated pickle file exported is named questions_df_09252020.pickle.

5. Question Text "Body" Analysis

The following pseudocode outlines the process used to analyze the 'Body' attribute - the question text. Some of these steps were performed in the preliminary portion of the "CP2_03_Clean_Body_Text.ipynb".

2. …
3. Examine the feature 'Body' which are the questions to gain statistical insights
4. Get counts of certain special characters to add to features prior to cleaning
    a. Question Marks
    b. Bolded Text
    c. Number of Paragraphs
    d. Code Examples

Additional exploration and visualization steps were performed in the "CP2_04_Analyze_Features.ipynb" notebook.

1. Load the pickled pandas dataframe from 03 notebook
2. Examine contents to assure everything transferred properly
3. Analyze length of body (both characters and words) against other features and id any correlations
4. Analyze number of question marks against other features and id any correlations
5. Analyze text bolding against other features and id any correlations
6. Analyze number of paragraphs against other features and id any correlations
7. Analyze number of code examples and length of those examples and id any correlations
8. Analyze NER results
9. Export the dataframe with cleaned text and additional features

Input is a pandas dataframe called questions_df consisting of 24363 entries and 63 columns.
The associated pickle file is named questions_df_ner_results_10282020.pickle.

For these steps, the following libraries were used: pandas, numpy, pickle, collections.
For visualizations the following libraries were used: matplotlib and seaborn.

The following figure illustrates the frequency distribution of the question length, showing a poisson distribution typical of count features.

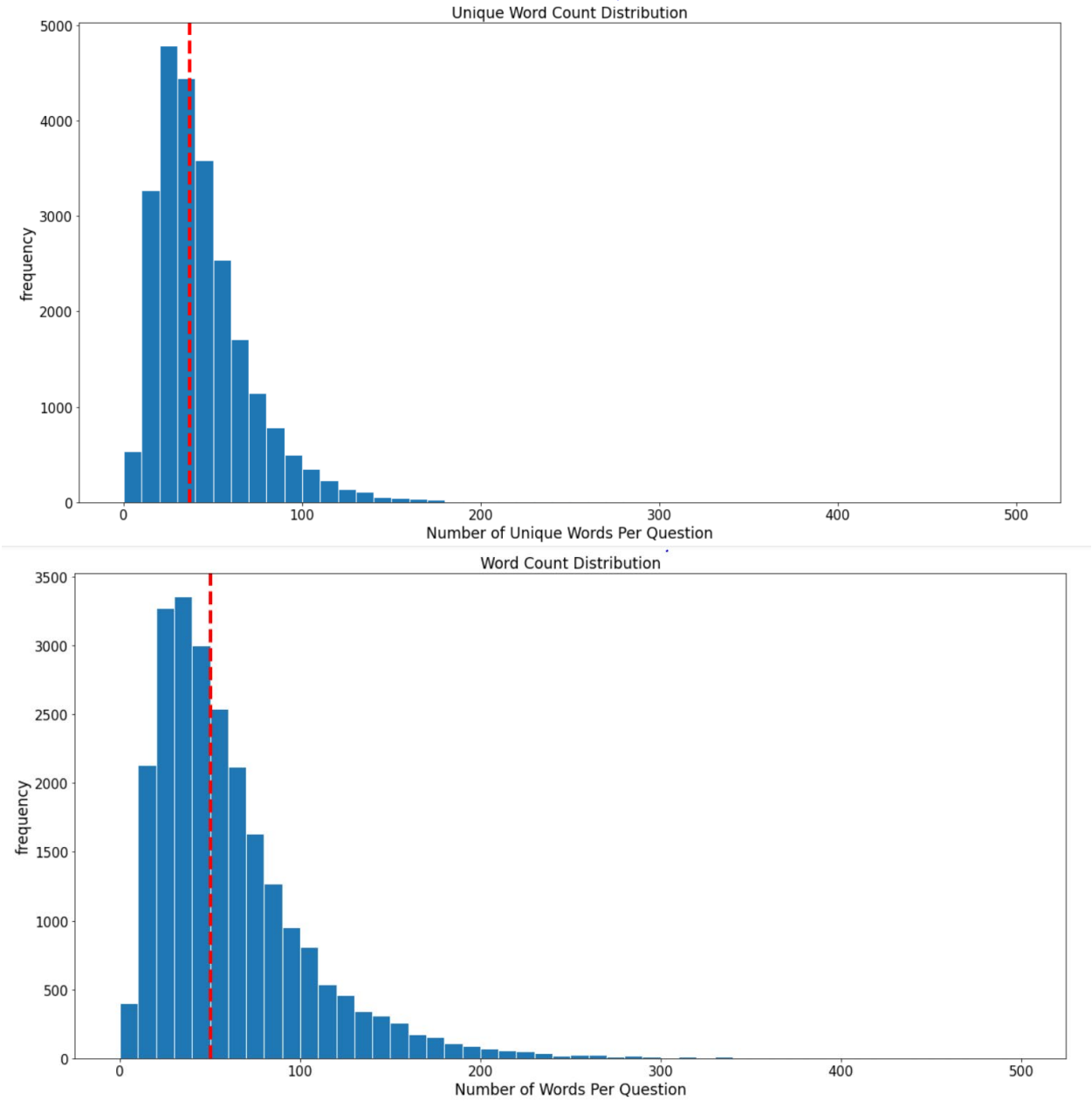


Fig 17: Frequency distribution of the number of unique words and number of words per question in the dataset, showing the skewed results (poisson distribution) as would be expected by a count feature. Red dashed line denotes the median, which is 44 and 62 respectively.

Figure 18 indicates a weak positive correlation between question length and questions that have top tags.

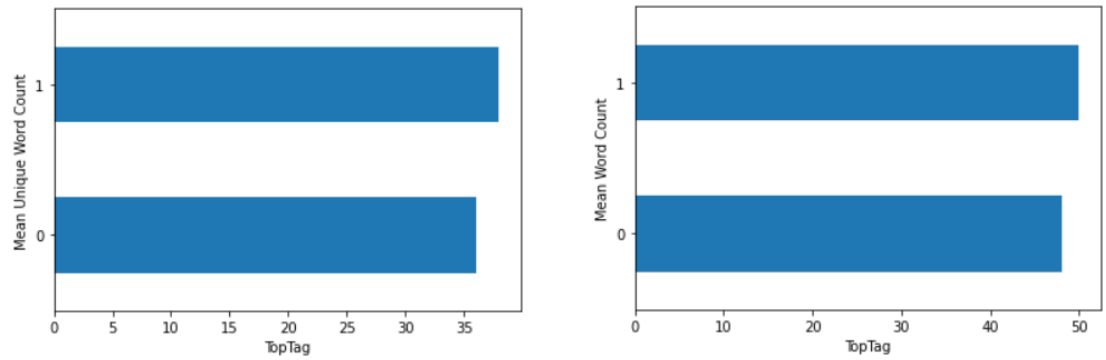


Fig 18: Mean unique word / word counts between questions with Top Tags (1) and without Top Tags (0). Questions with Top Tags tend to have slightly more words but this is a very weak correlation.

Figure 19 shows an inverse relationship between both ViewCount/ Score versus question length, suggesting that longer questions tend to have lower ViewCounts and Scores and vice versa.

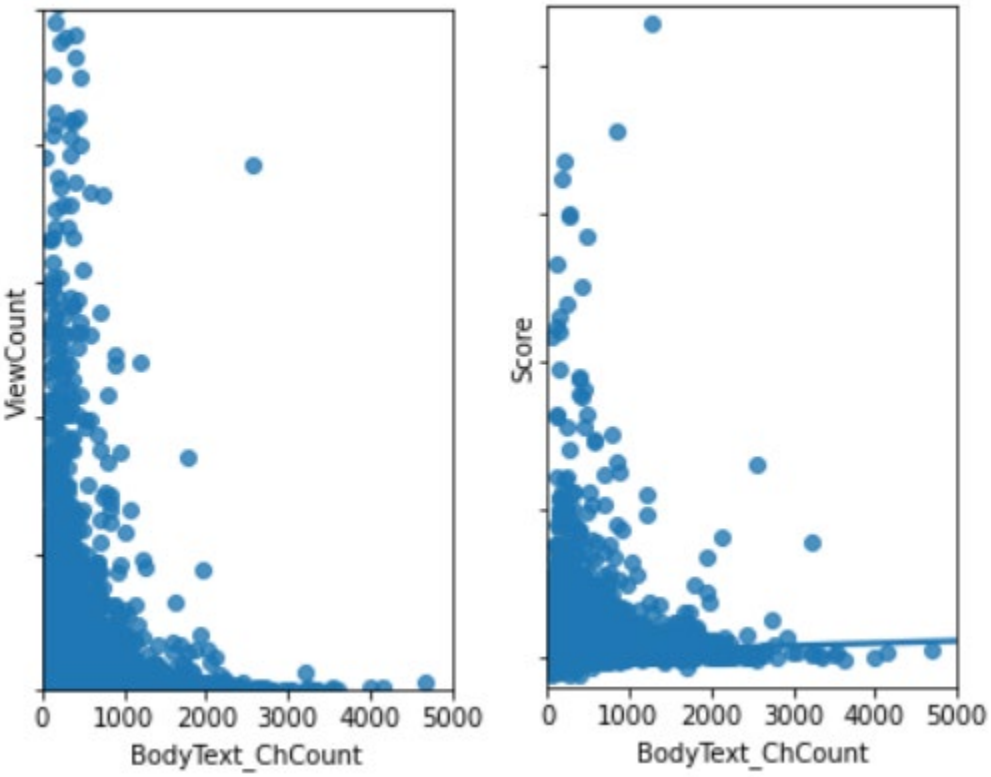


Fig 19: Crossplots of ViewCount and Score versus character count per question showing the inverse relationship between these features and question length.

A seaborn pairplot was employed to examine the relationships between additional features such as number of questions asked within each question body, bold count, paragraph counts, and code count (Figure 20). The results indicate that Score and ViewCount are positively correlated to one another, but the majority of correlations shown in the plot are inverse. For instance, when a lot of questions are asked, the bold count, code snippet count, and paragraph count goes down. This may speak to the different and unique styles of the question askers:

some are big question askers, others are big on bolding text for emphasis. Or in the case of code examples, the number of code snippets may tend to be higher for an ‘r’ or ‘python’ tag.

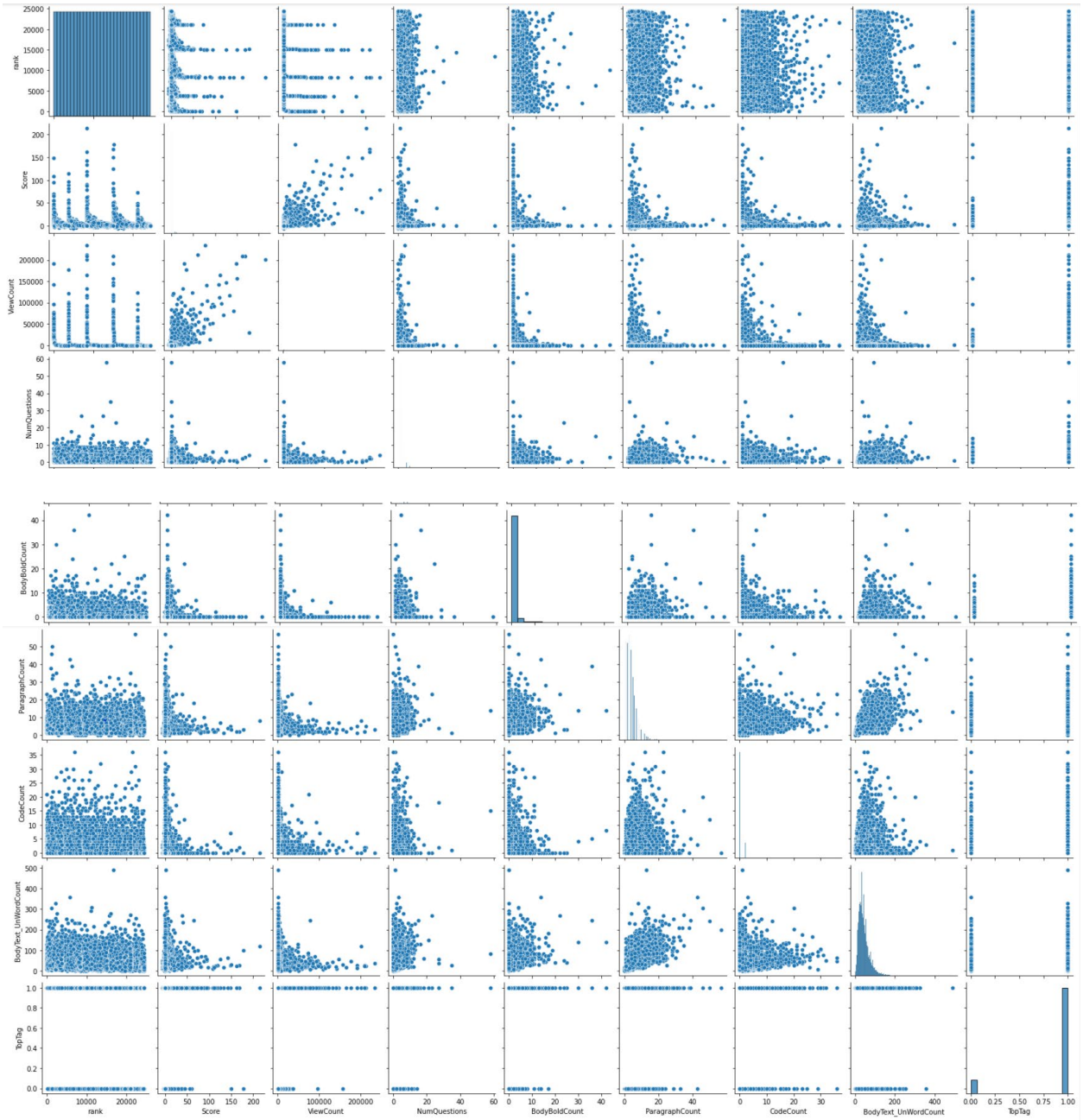


Fig 20: Seaborn pairplot of all features including counts of number of questions asked within each body, bold count, paragraph counts and code count.

Figure 21 is a visualization of the average question length by tag. Note that programming related ‘python’ and ‘r’ tags have shorter question length, likely because these tend to have more code examples in the question body, which have been removed during pre-processing. ‘Text-mining’ and ‘reinforcement-learning’ have the highest average question lengths.

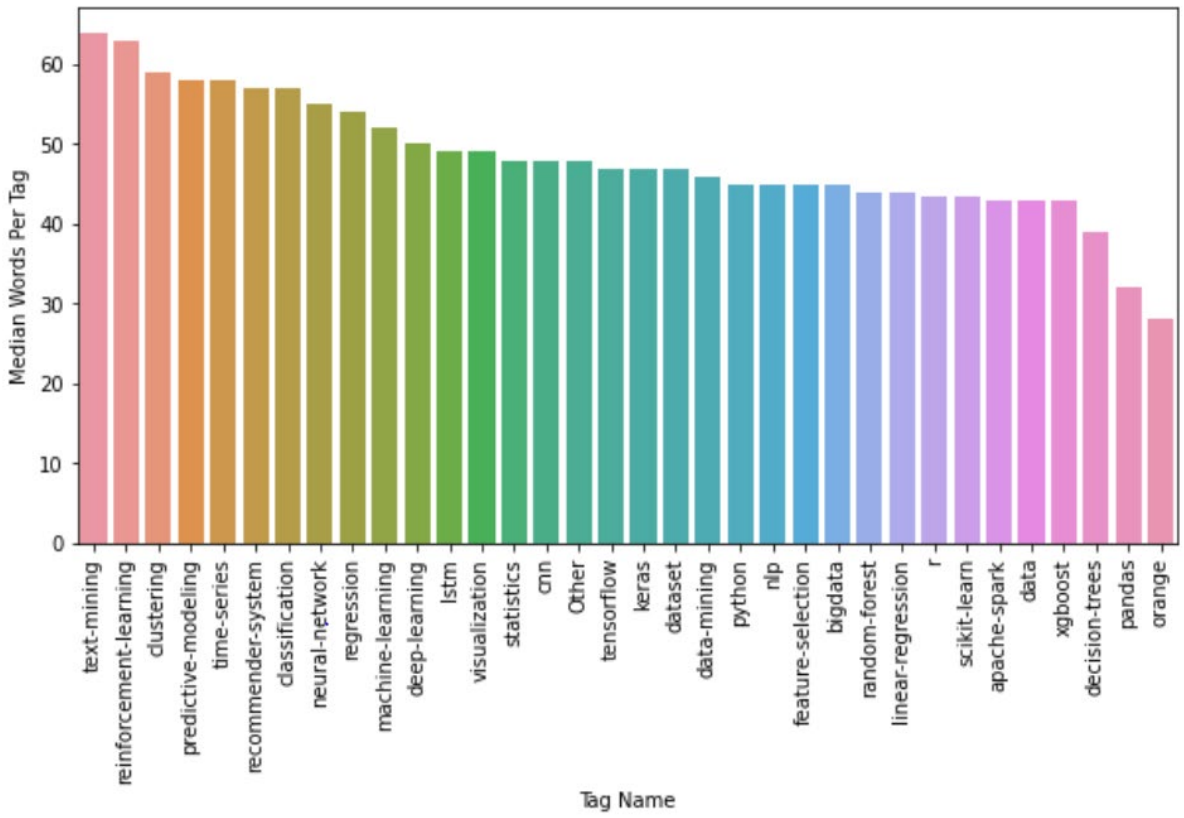


Fig 21: Bar chart showing the average question length by tag.

6.

7. Final DataFrame

During the cleaning and EDA processes, 33 blank question bodies were identified. Since no text remained, these rows were deleted leaving a remaining 24330 rows.

In preparation for the next transformation and modelling steps, unnecessary columns were dropped leaving only the 'Tag' (target variable) and 'BodyText_Clean' (predictor variable).

	Id	Tag	BodyText_Clean
0	5	machine-learning	always interest machine learn figure one thing...
1	7	Other	researcher instructor look opensource book sim...
2	14	data-mining	sure data science discus forum several synonym...
3	15	Other	situation would one system prefer relative adv...
4	16	machine-learning	use libsvm train data predict classification s...

Fig 22: Sample of final dataframe to be used for transformation and modelling steps.

This dataframe was saved out in pickle format with the following name: 'questions_df_clean_11052020.pickle'. The complete dataframe with all created and analyzed features was archived for safekeeping: 'questions_df_all.pickle'.

6. TEXT TRANSFORMATIONS

The final step in data conditioning is to transform the question text to a numerical format. Three different methods were employed and were compared to determine which method produced the best results during modelling.

The first and most basic way to represent the text numerically is via one-hot encoding or count vectorization. It provides a simple count of words in the document, with no attention paid to the ordering of these words, which is why it is also known by the term ' bag-of-words'.

An alternative method to the basic count vectorizer is to calculate word frequencies. The most popular method to do this is called TF-IDF (an acronym for "Term Frequency – Inverse Document Frequency)". TF-IDF evaluates how relevant a word is to a document within a collection of documents. Two metrics are considered: how many times a word appears in a document multiplied by the inverse document frequency of the word across a set of documents. The tf-idf value (or score) increases proportionally to the number of times a word appears in a document, but is offset by the number of documents that contain the word. The higher the score, the more relevant that word is in that particular document.

And finally the most sophisticated method of vectorizing the text to create document embeddings was employed: Gensim's Doc2Vec. Doc2Vec is an unsupervised algorithm that learns fixed-length feature vectors for each document and an extension of the better known Word2Vec algorithm. Doc2Vec is able to detect relationships among words and understands the semantics of the text based on the context of each word around the sentence it is surrounded with. Models with a range of vector sizes (25,50,100,200,500) were attempted to see which returned the best results.

The process of transforming the text was conducted in the "CP2_05_Transformation.ipynb" Jupyter Notebook. The following pseudocode outlines the process:

1. Load the pickled pandas dataframe from the last notebook and check file contents
2. Prep the dataset:
   a. Factorize Tags column to a numeric column
   b. Split into dev, cv, and test sets
   c. Verify the distribution of tags within the splits
   d. Create a separate dataframe for the predictor values in each of the splits
3. Count Vectorization
   a. Perform the process
   b. Hypertune parameters
   c. Score the optimized model
   d. Visualize the results
4. TF-IDF
   a. Perform the process
   b. Hypertune parameters
   c. Score the optimized model
   c. Visualize the results

5. Doc2Vec
    a. Perform the process
    b. Hypertune parameters
    c. Score the optimized model
    d. Visualize the results

Input is the 'questions_df_clean_11052020.pickle' file from the 04 notebook. For this notebook, the following additional libraries were used: itertools, gc, sklearn, gensim, and pretty table.

8. Data Preparation

Prior to performing these transformations, the tags were factorized, turning each of the tag's text into digits. The data set was then split into target (y) and predictor (X) variable sets. Then each of these sets were further split into train and test sets using a 70-30 % split. The train set was further split into development and cross validation sets using another 70-30% split. Due to the smaller dataset size (24330 rows), the same development, cross validation and test sets were used for all transformations.

Because the tag distribution is imbalanced, the 'stratify' parameter of the standard scikit learn module was employed. The results were quality checked to assure that the tag distributions of the splits were consistent with the overall tag distribution.

9. Count Vectorization

The count vectorization process was performed by first creating an instance of the CountVectorizer class from sci-kit learn's feature extraction module. The fit() function then learns the vocabulary of the questions in the train dataset while the transform() function encodes each word as a vector. An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appears in the document.

Because the vectors returned are sparse, they were transformed back to numpy arrays to view and better understand by calling the toarray() function. The initial count vectorization process run with default parameters resulted in a dictionary size of 42953 unique words from the development dataset.

The parameters min_df, max_df, and n-grams are the most important parameters to adjust to get the best results from the process. Min_df is used for removing terms that appear too infrequently. For example, a min_df = 0.01 means "ignore terms that appear in less than 1% of the documents". A min_df = 5 means "ignore terms that appear in less than 5 documents". The default min_df is 1, which means "ignore terms that appear in less than 1 document". Thus, the default setting does not ignore any terms. Keeping the default value for this dataset is ill-advised, since there are many mis-spellings, merged words, and hyperlinks. A reasonable range of min_df values was selected for cross validation of 10, 0.001, 0.005, and 0.01.

Max_df is used for removing terms that appear too frequently, also known as "corpus-specific stop words". For example, using a max_df = 0.50 means "ignore terms that appear in more than

50% of the documents". Using a max_df = 25 means "ignore terms that appear in more than 25 documents". The default max_df is 1.0, which means "ignore terms that appear in more than 100% of the documents". Thus, the default setting does not ignore any terms. Keeping the default value for this dataset is ill-advised, since there are many words in our vocabulary that are non-specific and do not help us predict the subject matter of the question. The default setting also leads to a large feature space and an algorithm that is fit on a large feature space will have computational complexities, thus hindering our ability to explore more models or iterations. A reasonable range of max_df values was selected for cross validation of 0.5,0.8,0.9,0.99, and 1000.

The count vectorization process also allows for the inclusion of strings of words (unigrams, bigrams and/ or trigrams). During the cross validation process, the optimal range of n-grams was also evaluated between values of 1 and 3.

Hypertuning was performed on the cross validation dataset using the reasonable range of values listed above for max_df and min_df and a base random forest model using 1000 n_estimators. An n-gram range of between 1 and 3 was also evaluated. Scoring was calculated using a ROC-AUC (this scoring method is best suited to evaluating multi-classification problems). The results indicated that the optimal params are max df = 0.99, min df = 0.005 and using only unigrams producing a dictionary size of 1231 words from the training dataset and a ROC-AUC score on the cross validation data set of 0.8627, as shown in the results table below.

Param Set #	min_df	max_df	ngram_range	feature size	train auc	val auc
1	10	1000	(1, 3)	9931	1.00	0.8349
2	10	1000	(1, 1)	3403	1.00	0.8444
3	0.001	0.999	(1, 1)	3144	1.00	0.8627
4	0.005	0.99	(1, 3)	1580	1.00	0.8614
5	0.005	0.999	(1, 2)	1563	1.00	0.8622
6	0.005	0.99	(1, 2)	1563	1.00	0.8622
7	0.005	0.999	(1, 1)	1231	1.00	0.8627
8	0.005	0.99	(1, 1)	1231	1.00	0.8627
9	0.01	0.5	(1, 3)	846	1.00	0.8407
10	0.01	0.9	(1, 1)	752	1.00	0.8444
11	0.01	0.8	(1, 1)	752	1.00	0.8444
12	0.01	0.5	(1, 1)	751	1.00	0.8446

Table 3: Parameter combinations tested during the count vectorization hypertuning process. The highlighted row shows the optimized parameters used.

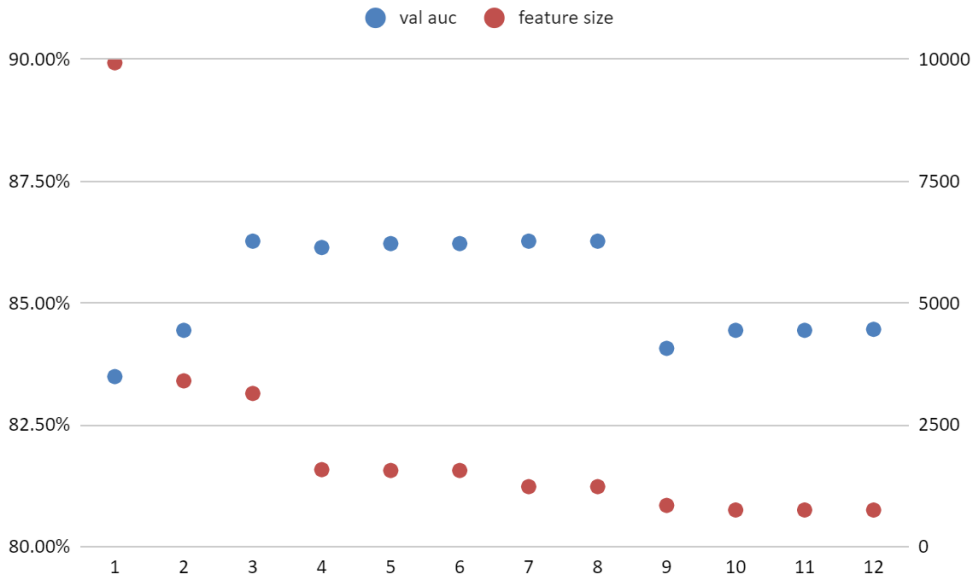


Fig 23: Scatter plot of cross validation set scores and feature size. Parameter set #8 was chosen as it resulted in the highest cross validation score with smaller feature size.

The diagrams below show the top unigrams, bigrams, and trigrams using the optimal parameters for 3 of the tags where we would expect a distinct segregation.

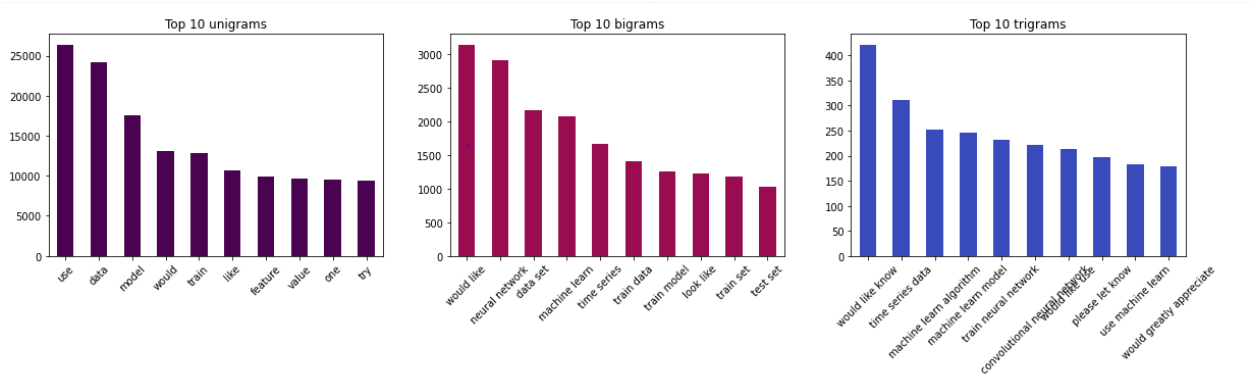
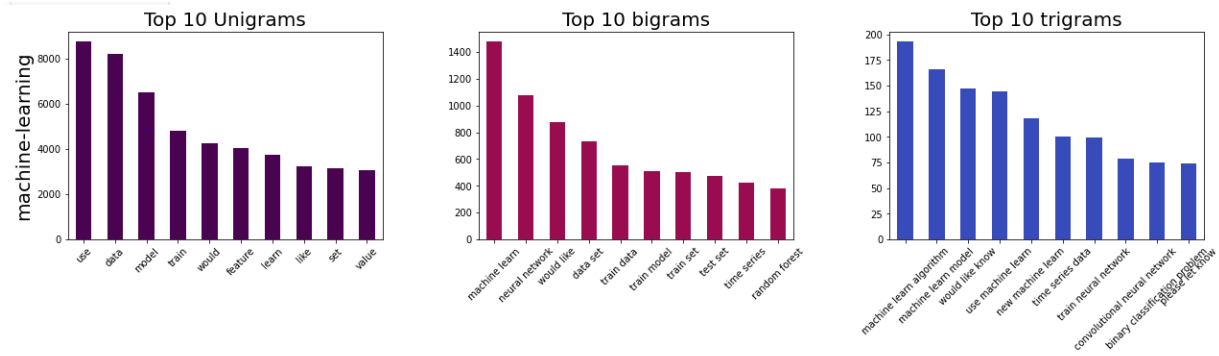


Fig 24: Top 10 unigrams, bigrams, and trigrams using the optimized count vectorizer process.



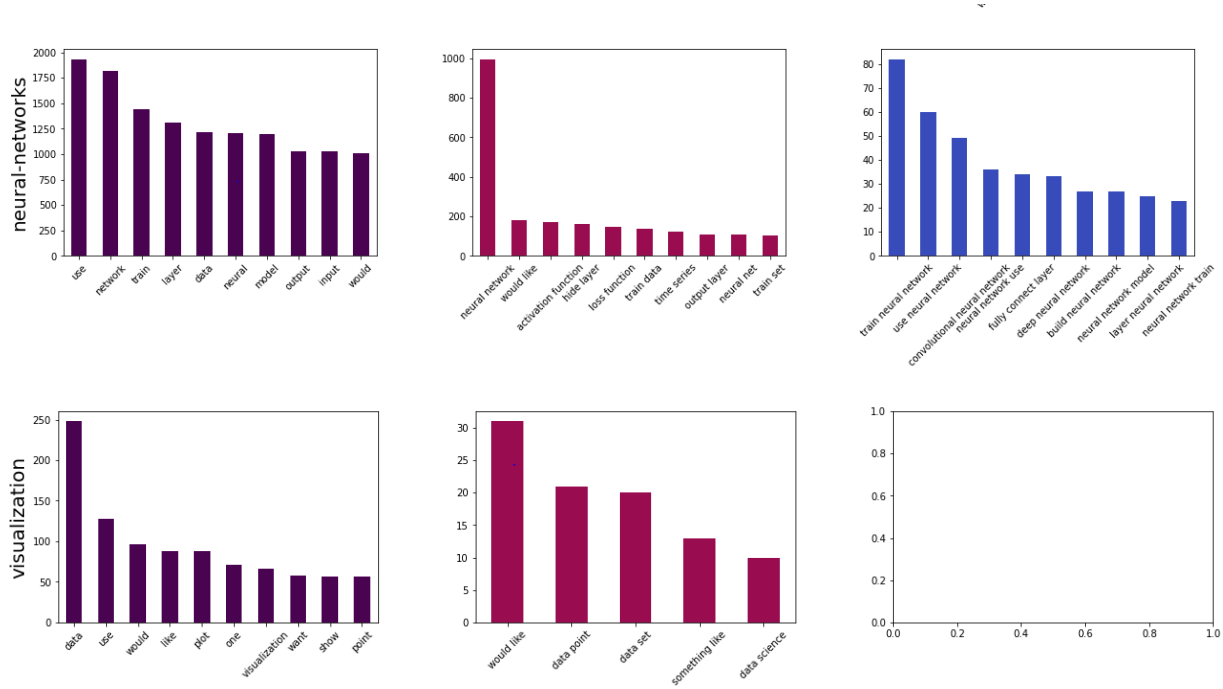


Fig 25: Top 10 unigrams, bigrams, and trigrams for three of the Top Tags using the optimized count vectorizer process. These tags were chosen because they were expected to show a clear segregation of words.

10. TF-IDF

The Tf-Idf vectorization method converts a collection of raw documents to a matrix of TF-IDF features, with the weight of each word returned. The tf-idf process was performed by first creating an instance of the TfidfVectorizer() class from sci-kit learn's feature extraction module. Like the CountVectorizer process, the fit() function then learns the vocabulary of the questions in the train dataset while the transform() function encodes each word as a vector. An encoded vector is returned with a length of the entire vocabulary and the tf-idf weight of each or the words in the document.

Again, because the vectors returned are sparse, they were transformed back to numpy arrays to view and better understand by calling the toarray() function. The initial tf-idf process run with default parameters resulted in a dictionary size of 42953 unique words from the development dataset.

The parameters min_df, max_df, and n-grams are the most important parameters to adjust to get the best results from the process and work very similarly to the count vectorization procedure.

Hypertuning was performed on the cross validation dataset using the same range of values listed above for max_df and min_df for the count vectorization process and a base random forest model using 1000 n_estimators. An n-gram range of between 1 and 3 was also evaluated. Scoring was calculated using a ROC-AUC (this scoring method is best suited to evaluating multi-classification problems). The results indicated that the optimal params are max df = 0.999, min df = 0.001 and using only unigrams producing a dictionary size of 3144 words from the training

dataset and a ROC-AUC score on the cross validation data set of 0.8678, just slightly better than the count vectorization process, as shown in the results table below.

Param Set #	min_df	max_df	ngram_range	feature size	train auc	val auc
1	10	1000	(1, 3)	9931	1.00	0.8501
2	10	1000	(1, 1)	3403	1.00	0.8426
3	0.001	0.999	(1, 1)	3144	1.00	0.8678
4	0.005	0.99	(1, 3)	1580	1.00	0.8624
5	0.005	0.999	(1, 2)	1563	1.00	0.8606
6	0.005	0.99	(1, 2)	1563	1.00	0.8606
7	0.005	0.999	(1, 1)	1231	1.00	0.8637
8	0.005	0.99	(1, 1)	1231	1.00	0.8637
9	0.01	0.5	(1, 3)	846	1.00	0.8443
10	0.01	0.9	(1, 1)	752	1.00	0.846
11	0.01	0.8	(1, 1)	752	1.00	0.846
12	0.01	0.5	(1, 1)	751	1.00	0.8439

Table 4: Parameter combinations tested during the TF-IDF hypertuning process. The highlighted row shows the optimized parameters used.

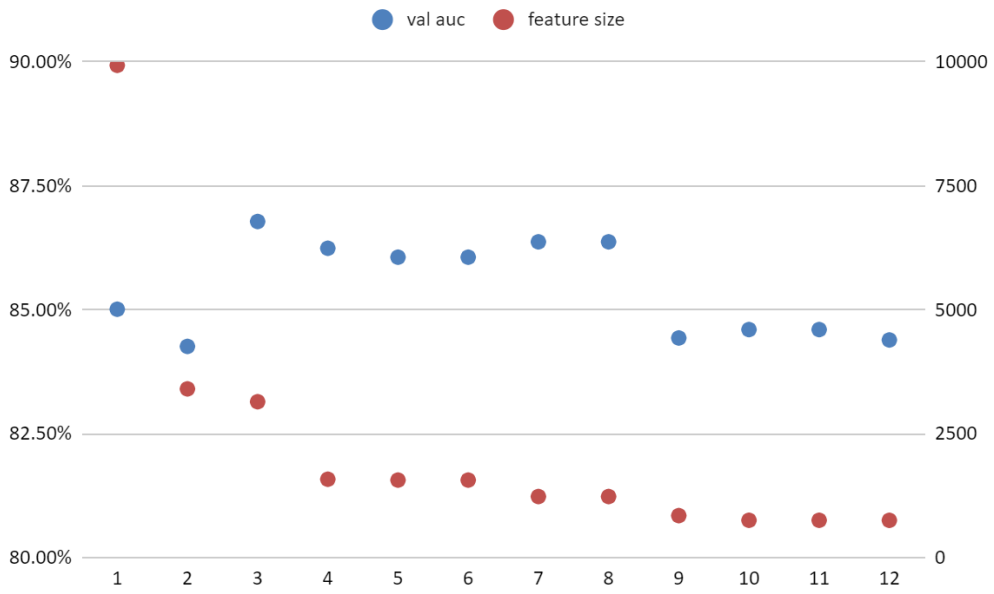


Fig 26: Scatter plot of cross validation set scores and feature size. Parameter set #3 was chosen as it resulted in the highest cross validation score with smaller feature size.

The diagrams below show the top unigrams, bigrams, and trigrams using the optimal tf-idf parameters for the same 3 tags visualized with the count vectorizer, where we would expect a distinct segregation.

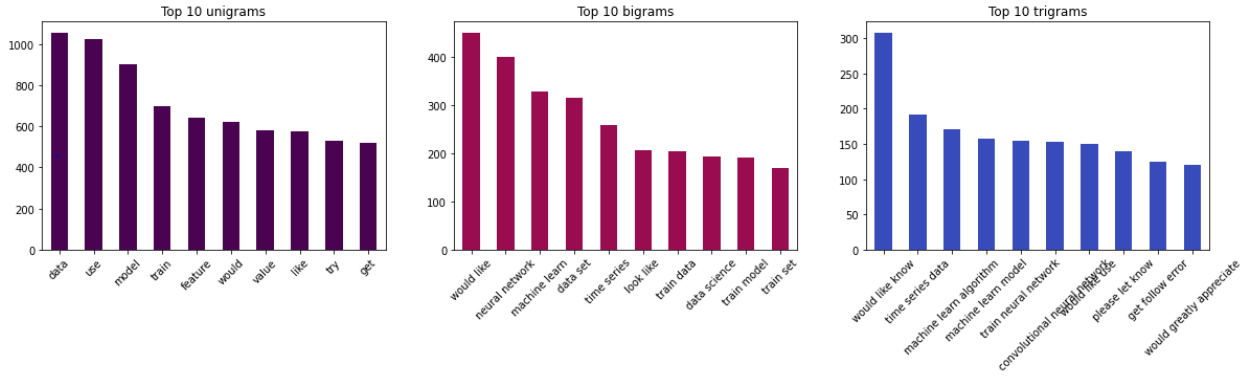


Fig 27: Top 10 unigrams, bigrams, and trigrams using the optimized tf-idf process.

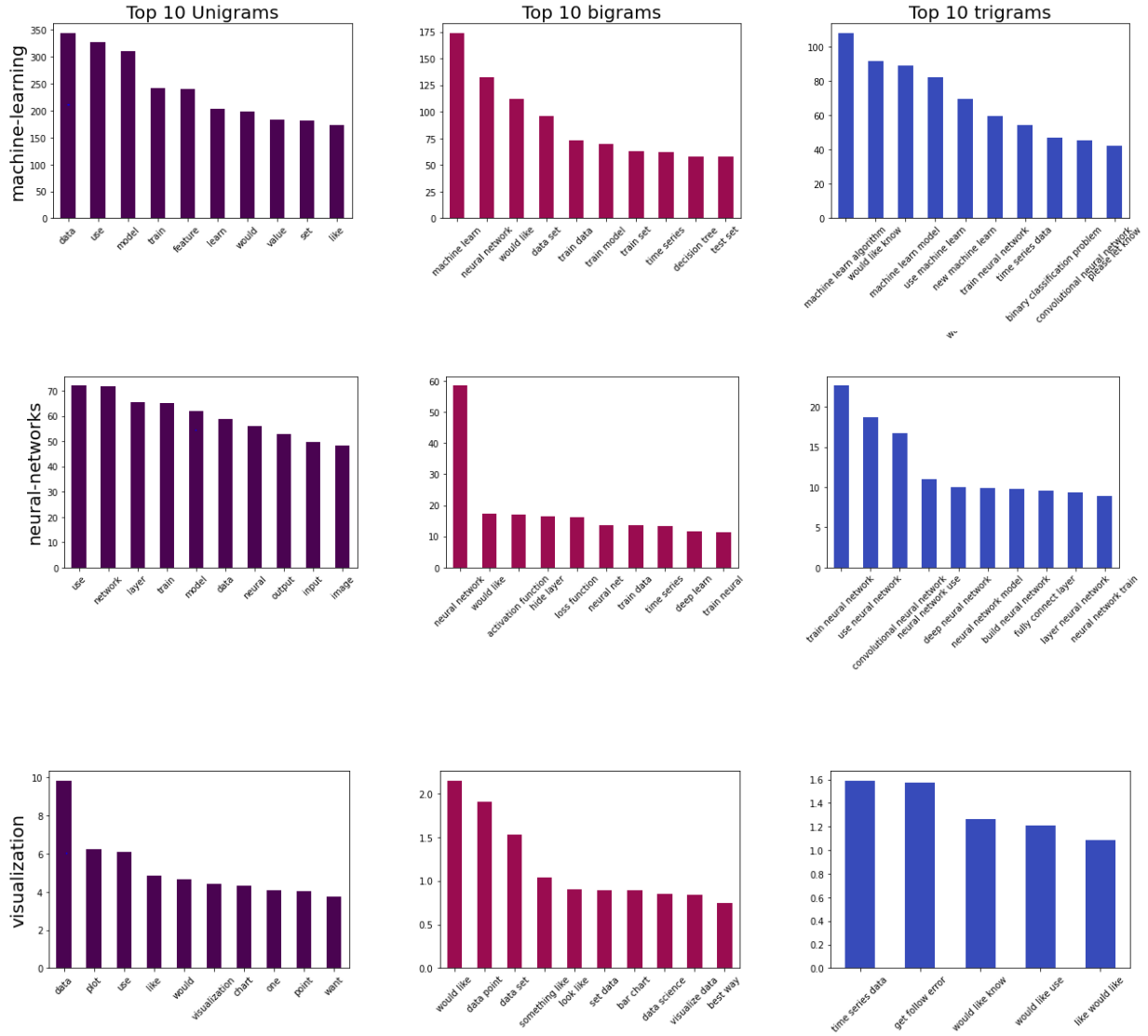


Fig 28: Top 10 unigrams, bigrams, and trigrams for three of the Top Tags using the optimized tf-idf process.

11. Doc2Vec

Doc2Vec is the most sophisticated transformation process that will be used in this analysis to obtain a mathematical representation of each question in multi-dimensional vector space. Le and Mikolov in 2014 introduced the Doc2Vec algorithm which usually outperforms such simple-averaging of Word2Vec vectors(https://cs.stanford.edu/~quocle/paragraph_vector.pdf). The basic idea is to create a floating word-like vector for each document (paragraph ID), which contributes to all training predictions and is updated like other word-vectors.

Steps are: 1) convert each of the predictor variable data splits to a tagged document; 2) Build the model by creating an instance of the Doc2Vec classifier from Gen-sim's model module; 3) Build the vocabulary with the tagged documents; and 4) Train, Test, Evaluate.

In contrast to the first two transformation processes utilized, the Doc2Vec algorithm has quite a few more parameters to consider. There are two main implementations, controlled by the 'dm' parameter:

Paragraph Vector - Distributed Memory (PV-DM) / dm = 1
Paragraph Vector - Distributed Bag of Words (PV-DBOW) / dm = 0

PV-DM is analogous to Word2Vec CBOW. The doc-vectors are obtained by training a neural network on the synthetic task of predicting a *center* word based on an average of both context word-vectors and the full document's doc-vector. This implementation preserves the word order in a document.

PV-DBOW is analogous to Word2Vec SG. The doc-vectors are obtained by training a neural network to predict the vector of surrounding words in the given paragraph and paragraph id vector based on a given word in the paragraph. This implementation uses the bag of words approach, not preserving any word order.

Another important parameter is the pre-set vector size (or dimensionality) used to represent each document. For the hyptuning process, sizes of 25, 50, 100, 200, and 500 were evaluated. Other parameters included in the hypertuning process include window, hs, and epochs. Window is the maximum distance between the current and predicted word within a sentence. Sizes of 1 through 5 were evaluated. Hs is short for 'hierarchical softmax'. If set to 1, hierarchical softmax will be used for model training. If set to 0, and negative is non-zero, negative sampling will be used. Epochs are the number of iterations over the corpus. The default is 10, but values of 10, 25, 50, and 100 were evaluated.

The hypertuning process employed the library itertools to loop over the parameter settings and score the cross validation results. Table 3 lists the parameter setting combinations producing the 10 highest ROC-AUC scores on the cross validation dataset.

Rank	dm	vector_size	window	hs	epoch	AUC

1	0	200	5	0	10	0.8638
2	0	500	4	0	10	0.8632
3	0	500	5	0	10	0.8628
4	0	500	3	0	10	0.8627
5	0	500	1	0	10	0.8617
6	0	500	2	1	10	0.8614
7	0	200	4	0	10	0.8610
8	0	500	5	1	10	0.8603
9	0	100	2	0	10	0.8603
10	0	500	2	0	10	0.8603

Table 5: Parameters for the Doc2Vec models with the top 10 highest ROC-AUC cross-validation scores; note that the larger vector sizes tend to give the highest scores.

Notice that the highest Doc2Vec score (0.8638) is slightly worse than the optimal tf-idf transformation and only slightly better than count vectorization. The best results were obtained from the PV-DBOW implementation (dm = 0), scoring 10-20% better than the PV-DM implementation. This was surprising given literature suggesting that the PV-DM model is state of the art and often gives the best results. Although slightly poorer than the highest scoring 200 vector size model, the second highest 500 vector-size model was used in the modelling phase as the larger vector size was predicted to regularize better than the 200 vector size model.

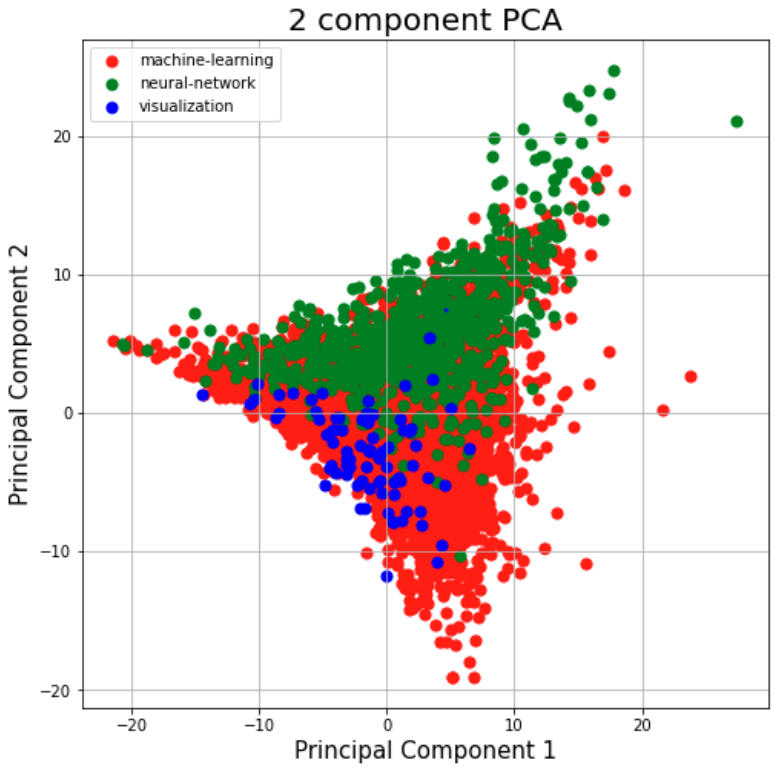


Fig 29: 2-Component PCA plot showing optimized Doc2Vec transformation results for 3 tags in vector space. These tags are the same tags visualized with the count vectorizer and tf-idf transformation previously, and show a distinct segregation.

7. MODELLING

For this supervised learning, multi-classification problem, three different techniques were employed to determine which could make the best predictions from the transformed question text, using the three different transformation processes from above: Random Forest, Bernoulli Naive Bayes, and LightGBM. The objective function was cross entropy - to measure the difference between two or more probability distributions for a given set of events.

The Random Forest model was chosen because of its popularity, effectiveness, and wide application. The algorithm uses an ensemble technique of both decision trees and bagging (bootstrap aggregating). The bagging process takes a subset of observations and a subset of variables to build each tree. The method is characterized by having quite high accuracy but with a tendency to overfit and was slower to run than the other models examined. The overfitting can be remediated by increasing the number of trees (n_estimators parameter).

A Naive Bayes algorithm was also chosen. Naive Bayes is a probabilistic classifier based on Bayes Theorem which gives the likelihood of the event occurring and is often used in text analysis problems. Multinomial Naive Bayes is frequently used for document classification. It generates one term from the vocabulary in each position of the document. An alternative is the Bernoulli Naive Bayes model, which generates an indicator for each term of the vocabulary, either indicating presence or absence of that term in the document (https://nlp.stanford.edu/IR-book/html/htmledition/the-bernoulli-model-1.html). These models run very fast with few parameters. During preliminary evaluation with this dataset, Bernoulli Naive Bayes gave higher scores than the Multinomial Naive Bayes model, so the prior was chosen to evaluate in more detail.

LightGBM is a newer algorithm and is gaining increased popularity. Like Random Forest, it is an ensemble technique using both decision trees, but is complimented by a gradient boosting technique. The latter uses a loss function to turn weaker learners into stronger ones through iteration. Growth is leaf-wise rather than depth-wise. The algorithm runs relatively fast and is low on memory consumption. The increase in the speed of the algorithm is facilitated by binning continuous input features, thus leading to faster splits. However, there is a tendency to overfit, particularly with smaller datasets.

12. Random Forest

During the transformation process described in the previous chapter, the Random Forest classifier was used as a baseline model to evaluate and score the results of the different transformations. These results were re-evaluated and tabulated in the 'CP2_06_RandomForestModelling.ipynb' Jupyter Notebook. The Random Forest was run with both n_estimators = 10 and 1000, with the latter showing a significant improvement in ROC AUC cross validation scores. However, the difference between the train and cross validation scores indicates significant overfitting.

Model	Estimators	Transformation	AUC Train Score	AUC Val Score
Random Forest	1000	Tf-idf	1.0	0.8678
Random Forest	1000	Count Vectorizer	1.0	0.8627
Random Forest	1000	Doc2Vec	1.0	0.8574
Random Forest	10	Count Vectorizer	0.9999	0.6557
Random Forest	10	Doc2Vec	1.0	0.6724
Random Forest	10	Tf-idf	0.9999	0.6285

Table 6: ROC-AUC Cross Validation Scores for Random Forest Model Using 3 Different Transformations.

The table above shows that all models are highly over-fitted and that the tf-idf transformation with the higher number of estimators gives the higher AUC score on the cross validation dataset of 0.8678.

As a final step, a hypertuning step using gridsearchcv was performed on the random forest model, using the tf-idf transformation to attempt to improve the score and reduce overfitting. The following parameters were tested as these were identified as parameters that could reduce overfitting: 'n_estimators': [10,100,500,1000], 'min_samples_leaf' : [1,2,5], 'max_depth' : [None, 5,10,15]. The results indicated that the optimal parameters are: {'max_depth': 15, 'min_samples_leaf': 5, 'n_estimators': 1000}, resulting in an auc train score of 0.976 and auc val score of 0.8836.

13. Bernoulli Naive Bayes

The results of the Bernoulli Naives Bayes approach were evaluated and tabulated in the 'CP2_08_BernoulliNaiveModelling.ipynb' Jupyter Notebook.

Model	Transformation	AUC Train Score	AUC Val Score
Bernoulli Naive Bayes	Doc2Vec	0.9682	0.8546
Bernoulli Naive Bayes	Count Vectorizer	0.9356	0.8323
Bernoulli Naive Bayes	Tf-idf	0.9056	0.7832

Table 7: ROC-AUC Cross Validation Scores for Bernoulli Naive Bayes Using 3 Different Transformations.

The table above shows that the Bernoulli Bayes model has slightly lower cross validation scores while resulting in less significant overfitting than the Random Forest model. The Doc2Vec transformation gives the higher AUC score on the cross validation dataset, but overfitting is still an issue.

As a final step, a hypertuning step using gridsearchcv was performed on the Bernoulli Naive Bayes model, using the Doc2Vec transformation to attempt to improve the score and reduce overfitting. The alpha parameter was tested as it was identified as the only parameter that could reduce overfitting. Using the default value of 1.0 (already used) produced the best results.

14. LightGBM

The results of the LightGBM approach were evaluated and tabulated in the 'CP2_07_LightGBMModelling.ipynb' Jupyter Notebook. The algorithm has many parameters that must be considered. Since it is prone to overfitting, an additional regularization approach was employed to identify the best candidate models (those with best accuracy and least overfit). A brute force method using a random sampling of 1000 permutations with random LightGBM parameter values for 12 different parameters was used. Parameters evaluated include: subsample, learning_rate, min_leaf_in_data, lambda_l1, lambda_l2, colsample_bytree, max_depth, max_bin, min_gain_to_split, num_leaves, miin_split_gain, and n_estimators. A loop was used to take each row of randomized parameters and measure an AUC score for the dev, cv, and test datasets.

Model	Transformation	Avg Overfit	Min Overfit	Max Val AUC
LGBM	Count Vectorizer	5.57%	0.03%	0.8837
LGBM	Tf-idf	7.78%	0.44%	0.8759
LGBM	Doc2Vec 50	19.38%	11.83%	0.8440
LGBM	Doc2Vec 100	18.86%	10.54%	0.8507
LGBM	Doc2Vec 200	19.24%	11.96%	0.8463
LGBM	Doc2Vec 500	19.37%	12.27%	0.8440

Table 8: Summary of LightGBM Regularization Process Results

The table above is a summary of the LightGBM regularization process for each of the different transformations, including Doc2Vec 50, 100, 200, and 500 vector sizes. The Count Vectorizer transformation method produces the highest validation scores and least overfit. The Tf-idf transformation method comes in a close second. Maximum val auc scores for both of these transformations are higher than the best Random Forest and Bernoulli Naive Bayes models, and result in significantly less overfitting. Unfortunately all Doc2Vec vector sizes produce lower maximum validation AUC scores and significantly more overfitting.

8. CANDIDATE MODEL SELECTION

Six candidate models were selected to evaluate further, based on their auc scores and overfit percentages. All of these candidates are LGBM models using the count vectorizer and tf-idf transformations. No LGBM - Doc2Vec models were chosen because these all appear to be too highly overfitted and have poorer scores as well. Likewise, no Random Forest or Naive Bayes model were chosen, as these were determined to have lower auc scores and too highly overfitted.

Ajith - using the final lightgbm model parameter set to calculate the top 5, 10, 15, 20 tags

Params Set #	Model Type	subsample	learning_rate	min_data_in_leaf	lambda_l2	colsample_bytree	max_depth	max_bin	min_gain_to_split	num_leaves	min_split_gain	n_estimators	lambda_l1	train_auc	val_auc	oot_auc	Overfit Diff
106	CV / LGBM	1	0.01	100	1	0.2	5	50	1	85	0.5	500	1	0.9243	0.8784	0.8832	4.97%
209	CV / LGBM	0.2	0.005	25	25	0.4	5	200	1	5	0	1500	1	0.8934	0.8684	0.8691	2.79%
468	CV / LGBM	1	0.001	50	0	0.4	15	100	10	245	0.01	1500	1	0.8761	0.8574	0.8619	2.14%
907	TF-IDF / LGBM	0.4	0.1	25	50	0.4	13	100	1	125	1	750	1	0.9121	0.8668	0.8698	4.96%
209	TF-IDF / LGBM	0.2	0.005	25	25	0.4	5	200	1	5	0	1500	1	0.9061	0.8643	0.8679	4.61%
468	TF-IDF / LGBM	1	0.001	50	0	0.4	15	100	10	245	0.01	1500	1	0.8920	0.8592	0.8623	3.68%

Table 9: LGBM Candidate Models Selected

The table above is a summary of the LGBM parameters, ROC-AUC scores, and overfit percentages of the six candidate models chosen.

9. RESULTS and CONCLUSIONS

The ROC-AUC scores on the test dataset, using the six candidate models selected, range from 85-88% with overfitting ranges of 2-5%. These positive results indicate that a new method of automated tagging of the primary tag on Stack Exchange Data Science is viable.

Three different text to vector processes were tested. The success of these vectorization processes is dependent on the model used. For Random Forest modelling, the TF-IDF method produced the highest scores. For Bernoulli Naive Bayes modelling, the Doc2Vec method produced the highest scores. And for the LGBM modelling, the count vectorization method produced the highest scores.

Since we have found now that the LGBM model is preferred to use going forward, it may be wise to perform further hypertuning of the count vectorizer and tf-idf transformations using this model instead of the random forest model used in this capstone project. We could also benefit by hypertuning the Doc2Vec transformation further to reduce overfitting.

The LGBM method produced both the highest ROC-AUC scores on the validation set and the lowest overfitting percentage, making it the preferred model over the Random Forest and Bernoulli Naive Bayes techniques. The LGBM algorithm also gave us the flexibility to regularize the algorithm better. Since it runs fast, experimentation was easier.

A next step would be to work to build a model that can predict multiple tags now that we've succeeded at predicting the first most popular tag. A possible additional task would be to create our own NER dictionary of Data Science terms to improve the results of the Named Entity Recognition process.

10. REFERENCES

XML Parsing:

https://www.guru99.com/manipulating-xml-with-python.html
https://medium.com/analytics-vidhya/converting-xml-data-to-csv-format-using-python-3ea09fa18d38
https://stackabuse.com/reading-and-writing-xml-files-in-python/
https://medium.com/@robertopreste/from-xml-to-pandas-dataframes-9292980b1c1c
https://www.xml-buddy.com/ValidatorBuddy.htm

Text Pre-Processing:

https://towardsdatascience.com/nlp-text-preprocessing-steps-tools-and-examples-94c91ce5d30
https://towardsdatascience.com/preprocessing-text-data-using-python-576206753c28
https://www.kdnuggets.com/2018/08/practitioners-guide-processing-understanding-text-2.html
https://www.crummy.com/software/BeautifulSoup/bs4/doc/
http://www.linguisticsweb.org/doku.php?id=linguisticsweb:tutorials:linguistics_tutorials:automaticannotation:stanford_pos_tagger_python
https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da
https://medium.com/mysuperai/what-is-named-entity-recognition-ner-and-how-can-i-use-it-2b68cf6f545d
https://towardsdatascience.com/building-a-text-normalizer-using-nltk-ft-pos-tagger-e713e611db8
https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html#:~:text=Lemmatization%20usually%20refers%20to%20doing,is%20known%20as%20the%20lemma%20.

Vectorizing:

https://monkeylearn.com/blog/what-is-tf-idf/#:~:text=TF%2DIDF%20is%20a%20statistical,in%20a%20collection%20of%20documents.&text=TF%2DIDF%20(term%20frequency%2D,document%20search%20and%20information%20retrieval.
https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/
https://towardsdatascience.com/natural-language-processing-count-vectorization-with-scikit-learn-e7804269bb5e
https://www.kaggle.com/adamschroeder/countvectorizer-tfidfvectorizer-predict-comments
https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e
https://towardsdatascience.com/implementing-multi-class-text-classification-with-doc2vec-df7c3812824d
https://medium.com/@mishra.thedeepak/doc2vec-simple-implementation-example-df2afbbfbad5
https://medium.com/betacom/hyperparameters-tuning-tf-idf-and-doc2vec-models-73dd418b4d
https://kavita-ganesan.com/tfidftransformer-tfidfvectorizer-usage-differences/#.YJtAzbVKhhF
https://github.com/RaRe-Technologies/movie-plots-by-genre/blob/master/ipynb_with_output/Document%20classification%20with%20word%20embeddings%20tutorial%20-%20with%20output.ipynb
https://cs.stanford.edu/~quocle/paragraph_vector.pdf

Machine Learning:

https://www.analyticssteps.com/blogs/what-light-gbm-algorithm-how-use-it
https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html
https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568
https://towardsdatascience.com/3-things-you-need-to-know-before-you-train-test-split-869dfabb7e50
https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6
https://levelup.gitconnected.com/classifying-reddit-posts-with-natural-language-processing-and-random-forest-classifier-af2d8fa77bd3
https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5
https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74
https://medium.com/@vibhuti.siddhpura/machine-learning-algorithms-introduction-fb86623c5218
https://towardsdatascience.com/how-i-was-using-naive-bayes-incorrectly-till-now-part-1-4ed2a7e2212b
https://nlp.stanford.edu/IR-book/html/htmledition/the-bernoulli-model-1.html

EDA:

https://towardsdatascience.com/a-complete-exploratory-data-analysis-and-visualization-for-text-data-29fb1b96fb6a
https://jakevdp.github.io/PythonDataScienceHandbook/04.14-visualization-with-seaborn.html
https://www.kaggle.com/codename007/start-from-here-quest-complete-eda-fe

Stack Exchange:

https://datascience.stackexchange.com/
https://archive.org/details/stackexchange
https://www.cs.cornell.edu/~arb/papers/stack-exchange-jocn-2019.pdf
http://cs229.stanford.edu/proj2013/SchusterZhuCheng-PredictingTagsforStackOverflowQuestions.pdf
https://stackoverflow.blog/2011/08/09/improved-tagging/#:~:text=But%20how%20do%20you%20determine,what%20you've%20typed%20so%E2%80%A6&text=%2DFounder%20(Former)-,Every%20Stack%20Exchange%20question%20is%20required%20to%20have%20at%20least,%2C%20order%2C%20and%20find%20questions.
https://medium.datadriveninvestor.com/predicting-tags-for-the-questions-in-stack-overflow-29438367261e
https://stackoverflow.com/help/how-to-ask

11. APPENDIX

8.1 Stack Exchange Data Science Data Schema