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

by Laura Elliott

Springboard Data Science Career Track
Capstone 2 Project / 2021

# Problem Definition

- Many questions are submitted to Stack Exchange daily
- Each question requires at least one tag to provide context for and to categorize the question
- Current tagging process involves 2 steps:
  - A recommender engine identifies relevant tags
  - User manually selects tags from recommender's choices
- Several specific user rules for the manual tagging
- Tag selection can be challenging for inexperienced users
- Customer experience can be affected

# Project Objectives

- Investigate relationships between the question text and their tags on the Stack Exchange Data Science platform
- Build an algorithm to predict the best tags automatically
  - Input should be the question text only
- Reduce the need for human intervention
- Improve the overall efficiency of the question answering process
- Ultimately, improve the customer experience

# Data Acquisition

- Download from Stack Exchange data dump site
  - Archive from May 31 2020
  - https://archive.org/details/stackexchange
- 2 separate zip files – meta and data
- 8 separate xml files in each zip file
  - Badges, Comments, Post History, Post Links, Posts, Tags, Users, Votes
- Popular Use Cases include:
  - Automated tag prediction
  - Automated question answering
  - Prediction of future question status (upvote, downvote, closed)
  - Prediction of length of time to answer
- Meta – Tags.xml used for initial evaluation of the tags and their distribution
- Data – Posts.xml contained all the information required
  - 79 Mbyte file – too large to load in a standard text editor
  - XMLValidator Buddy was used to preview the file and understand the structure

# Raw Data Attributes

- Posts.xml file contents
- Work performed in the following Jupyter notebook
  - CP2_01_Convert XML to DF.ipynb
- Xml.dom.minidom package used to load and parse the file in Python
- Dataframe created in pandas - posts_df: >51K rows / 15 columns

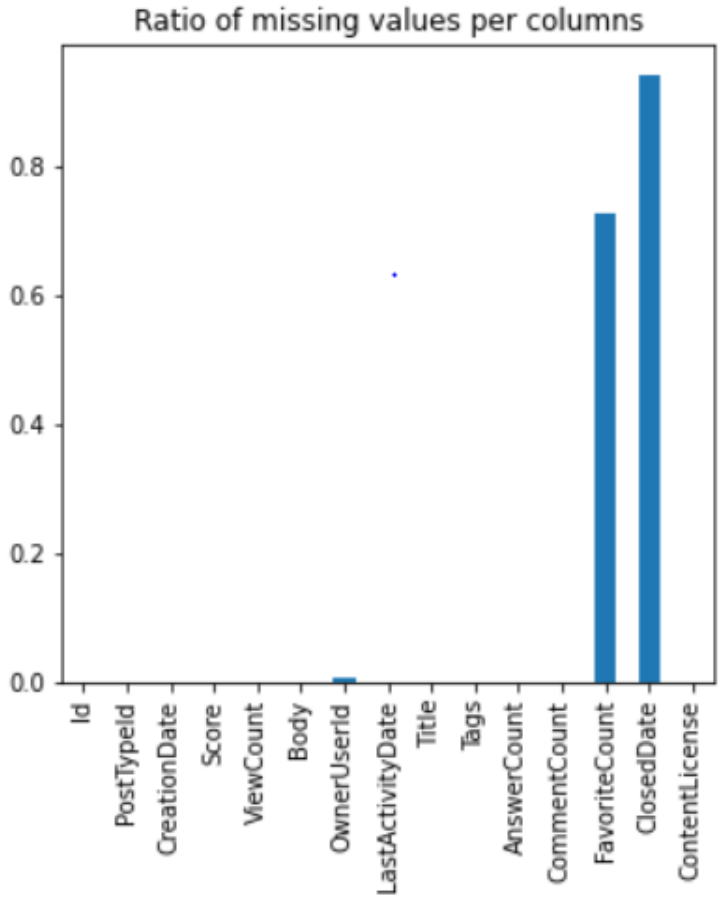
| *Features Name* | *Data Type* | *Description* |
|---|---|---|
| Id | int | unique post identifier |
| PostTypeId | tinyint | 1 = question; 2 = answer |
| CreationDate | datetime | Date of post |
| Score | int | sum of upvotes - downvotes |
| ViewCount | int | count of number of views |
| Body | nvarchar | 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 |

# Data Wrangling

- Work performed in the following Jupyter notebook
  - CP2_02_Load DF Clean and Analyze.ipynb
- The following libraries were used
  - pandas, pickle, numpy, collections
  - Matplotlib for visualizations
- Blank fields replaced with NaN
  - This was an artifact from the XML extraction process
- Filter out the answer rows leaving only the questions ~25K rows
  - PostTypeID flag = '1'

# Check for Missing Values

- Missing values only in non-pertinent data columns




```
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
```

# Removing Duplicate Rows

- 10 duplicate questions were identified
  - Same OwnerUserId, Same Body, One of the Pairs Closed

| 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 | |

  - Second entries with closed dates were removed
- 24353 rows remaining

# Text Preprocessing Steps

- Work performed in the following Jupyter notebook
  - CP2_03_Clean_Body_Text.ipynb
- Significant text preprocessing performed
  - Beautiful Soup – removal of code snippets and html formatting
  - Contractions – expansion
  - FastText – language detection to check for languages other than English
  - Re – special character removal
  - WordNet Lemmatizer – lemmatization
  - Spacy - Named Entity Recognition
  - Convert to lowercase
  - NLTK – stop word removal, part-of-speech (POS) tagging
  - Remove single numerals

# Before & After Examples

```
HighestRank ********************************************************************************************************************************************************

Question id: 410
Rank : 3
Question Tags    : machine-learning neural-network deep-learning optimization hyperparameter
Question Title   : Choosing a learning rate
Question Body    :

<p>I'm currently working on implementing Stochastic Gradient Descent, <code>SGD</code>, for neural nets using back-propagation, and while I understand its
purpose I have some questions about how to choose values for the learning rate.</p>  <ul> <li>Is the learning rate related to the shape of the error gradient,
as it dictates the rate of descent?</li> <li>If so, how do you use this information to inform your decision about a value?</li> <li>If it's not what sort of
values should I choose, and how should I choose them?</li> <li>It seems like you would want small values to avoid overshooting, but how do you choose one such
that you don't get stuck in local minima or take to long to descend?</li> <li>Does it make sense to have a constant learning rate, or should I use some metric
to alter its value as I get nearer a minimum in the gradient?</li> </ul>  <p>In short: How do I choose the learning rate for SGD?</p>


HighestRank ********************************************************************************************************************************************************

Question id: 410
Rank : 3
Question Tags    : machine-learning neural-network deep-learning optimization hyperparameter
Question Title   : Choosing a learning rate
Question Body    :

currently work implement stochastic gradient descent neural net use backpropagation understand purpose question choose value learn rate learn rate relate shape
error gradient dictate rate descent use information inform decision value sort value choose choose seem like would want small value avoid overshoot choose one
get stick local minimum take long descend make sense constant learn rate use metric alter value get near minimum gradient short choose learn rate sgd
********************************************************************************************************************************************************
```

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

# Before & After Examples

```
HighestRank ************************************************************************************************************************************************************

Question id: 5226
Rank : 4
Question Tags    : machine-learning python scikit-learn random-forest decision-trees
Question Title   : strings as features in decision tree/random forest
Question Body    :

<p>I am doing some problems on an application of decision tree/random forest. I am trying to fit a problem which has numbers as well as strings (such as country
name) as features. Now the library, <a href="http://scikit-learn.org" rel="noreferrer">scikit-learn</a> takes only numbers as parameters, but I want to inject
the strings as well as they carry a significant amount of knowledge.</p>  <p>How do I handle such a scenario?</p>  <p>I can convert a string to numbers by some
mechanism such as hashing in Python. But I would like to know the best practice on how strings are handled in decision tree problems.</p>
************************************************************************************************************************************************************


HighestRank ************************************************************************************************************************************************************

Question id: 5226
Rank : 4
Question Tags    : machine-learning python scikit-learn random-forest decision-trees
Question Title   : strings as features in decision tree/random forest
Question Body    :

problem application decision treerandom forest try fit problem number well string country name feature library scikitlearn take number parameter want inject
string well carry significant amount knowledge handle scenario convert string number mechanism hash python would like know best practice string handle decision
tree problem
************************************************************************************************************************************************************
```

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

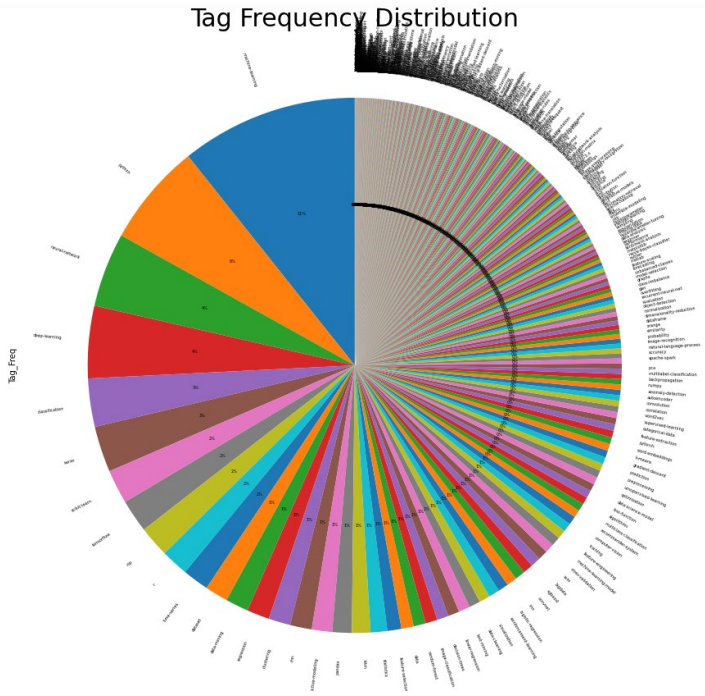
# Before & After Examples

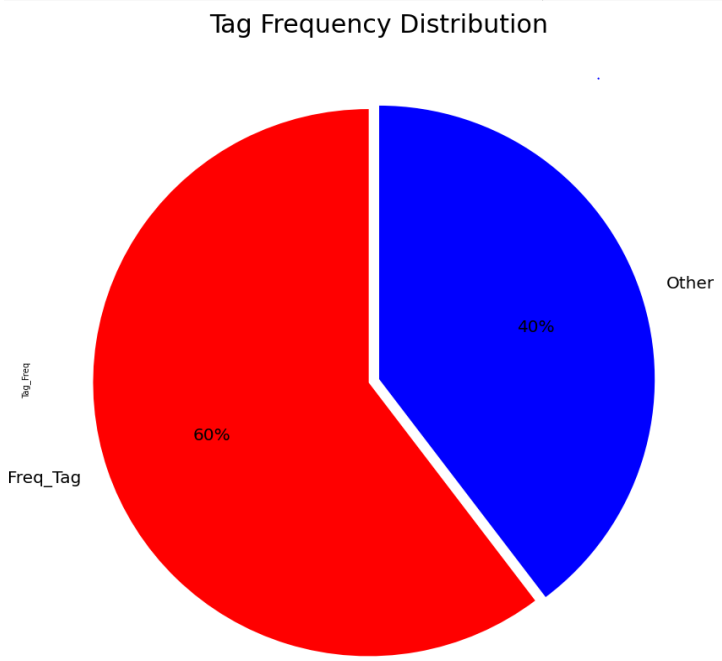
```
LowestRank ************************************************************************************************************************************************************************

Question id: 73375
Rank: 24348
Question Tags    : time

Question Title   : Getting a constant accuracy for a ping-like command

Question Body    :

<p>very simple and naïve question here, I'm trying to mesure some RTTs with a reasonnable accuracy. The Ubuntu ping command provides a pretty good mesure with a
10µs accuracy. but only when the total RTT time is under 10ms. </p>  <p>Actually, it seems that the output time is constantly given with 3 digits (e.g. 6.34 ms
; 17.3 ms ; 137 ms).</p>  <p>I would like to keep the 10µs accuracy whatever the output value.</p>  <p>Does anyone knows if such an option is available with the
command ping, or if there exist another tool which will allow me to get what I want.</p>  <p>Thanks in advance, and have a great day.</p>  <p>PS : I'm not a
native english speaker so i may have made some grammatical mistakes, sincere apologies for that.</p>
************************************************************************************************************************************************************************

 LowestRank ************************************************************************************************************************************************************************

 Question id: 73375
 Rank: 24348
 Question Tags    : time

 Question Title   : Getting a constant accuracy for a ping-like command

 Question Body    :

 simple naive question try mesure rtts reasonnable accuracy ubuntu ping command provide pretty good mesure 10 accuracy total rtt time 10ms actually seem output
 time constantly give 3 digit eg 634 173 137 would like keep 10 accuracy whatever output value anyone know option available command ping exist another tool allow
 get want thanks advance great day ps native english speaker may make grammatical mistake sincere apology
 ************************************************************************************************************************************************************************
```

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

# Exploratory Data Analysis

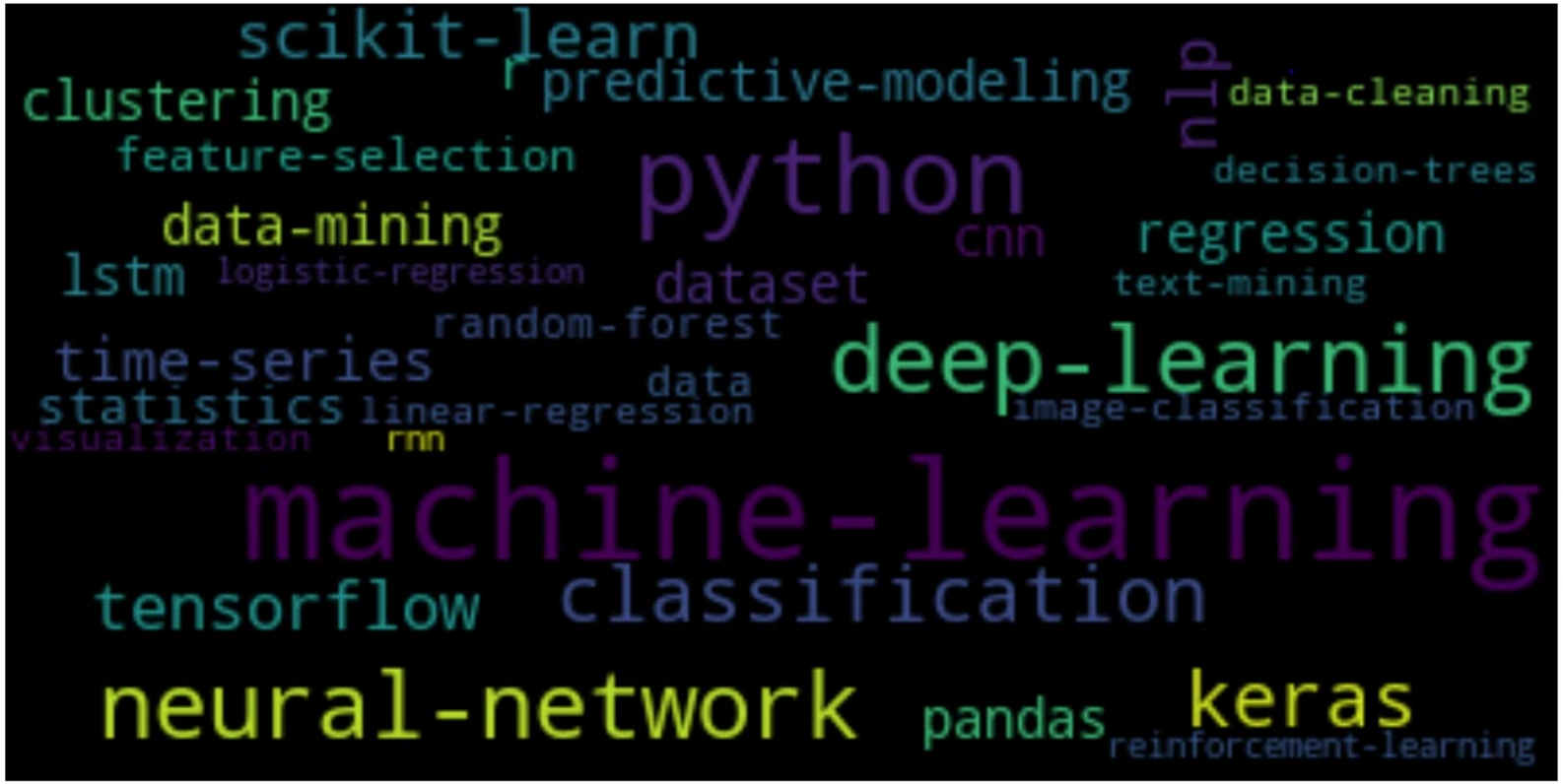
- Examine tag frequency to determine on which tags to focus our prediction power
- Work performed in CP2_02_Load DF Clean and Analyze.ipynb
  - 590 Unique Tags (up to 5 tags per question)
  - Top 32 most frequent tags encompass 60% of data



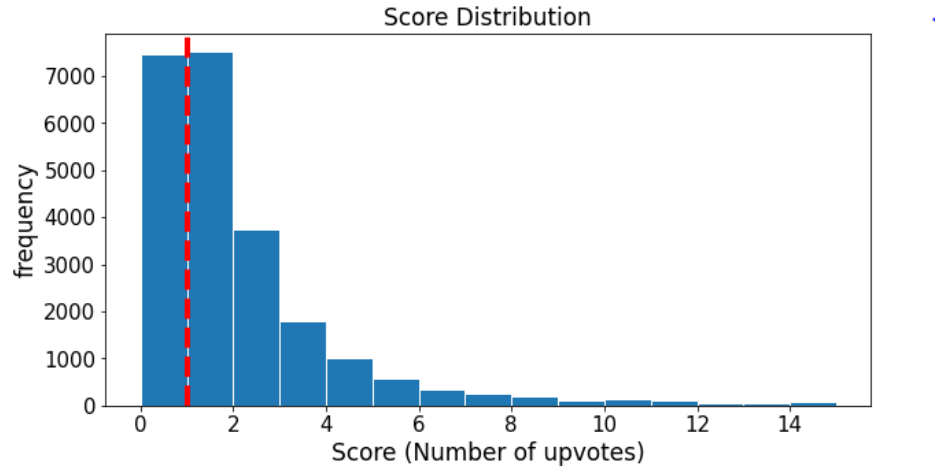



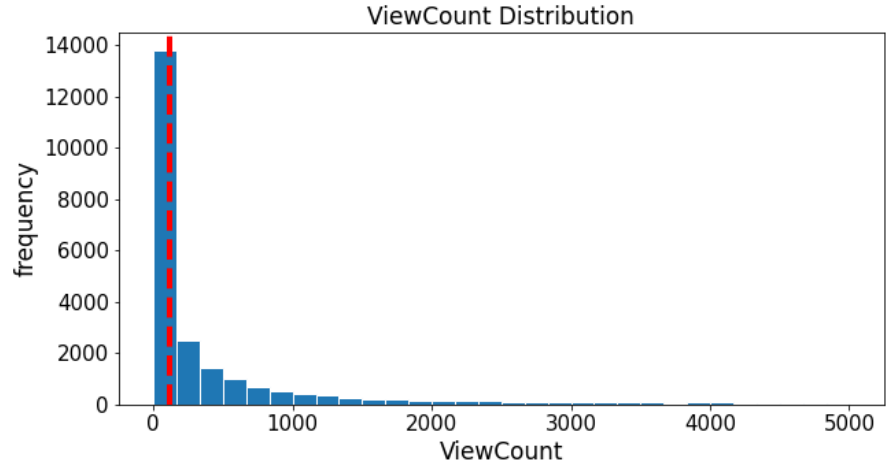
# Exploratory Data Analysis

- Examine tag frequency, cont'd
  - Wordcloud graphic showing the majority of most frequent tags are related to deep learning topics and related software / libraries

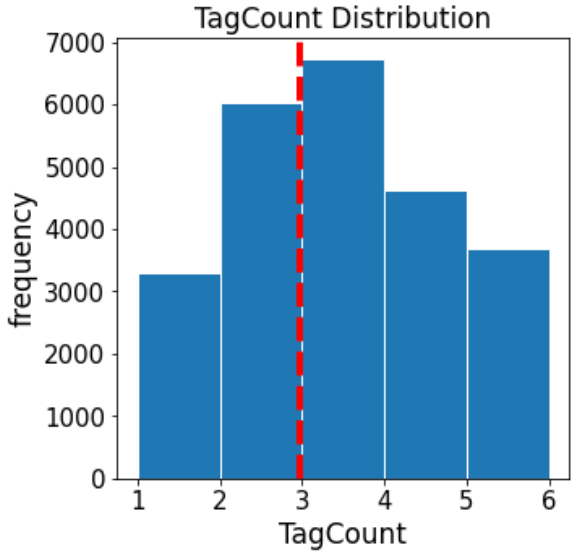


# Exploratory Data Analysis



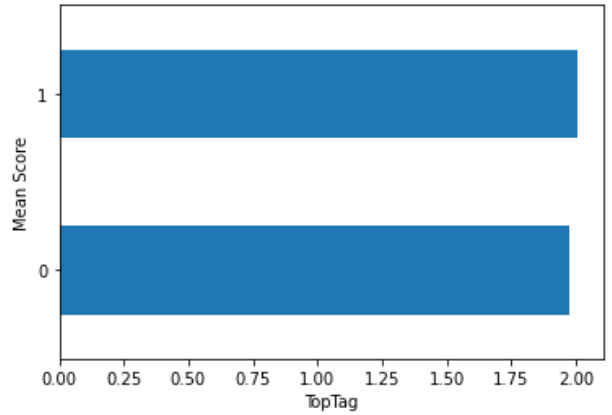

- Frequency distribution of other attributes
  - Score (poisson distribution)
  - ViewCount (poisson distribution)
  - # of Tags Per Question (normal distribution)

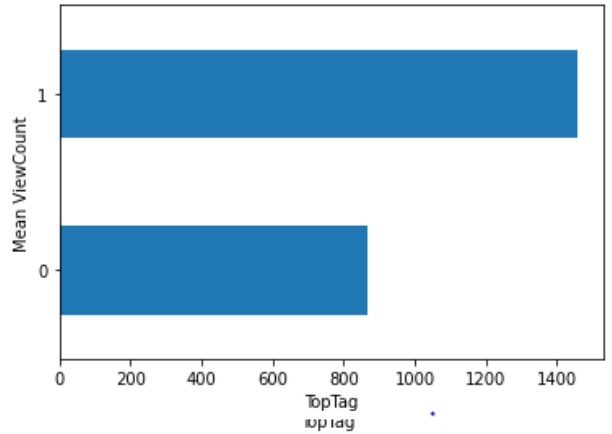


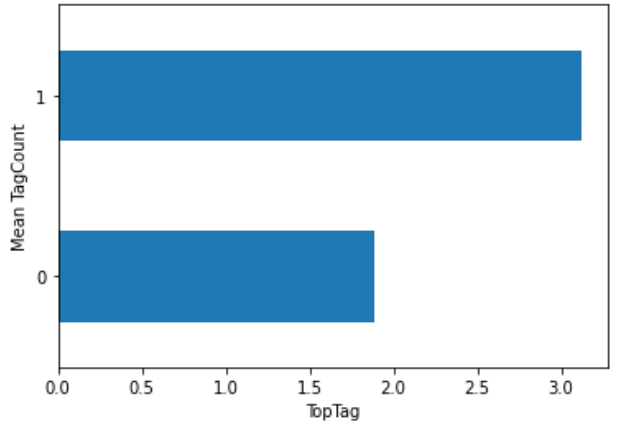

| | Score | ViewCount | TagCount |
|---|---|---|---|
| count | 24353.000000 | 24353.000000 | 24353.000000 |
| mean | 2.005872 | 1387.329651 | 2.974705 |
| std | 5.433220 | 6895.850492 | 1.257866 |
| min | -6.000000 | 2.000000 | 1.000000 |
| 25% | 0.000000 | 36.000000 | 2.000000 |
| 50% | 1.000000 | 114.000000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

# Exploratory Data Analysis

- Identify significant relationships between the top tags and other attributes
- Questions split into 2 groups and means calculated for each group
  - Those with tags on the TopTagsList (1)
  - Those with tags not on the Top TagsList (0)
- ViewCounts and TagCounts in TopTagsList questions are almost twice the mean value of those that are not
- Tags were further high-graded, and re-ranked; Final Rank order was determined by Tag Frequency + ViewCount + TagCount

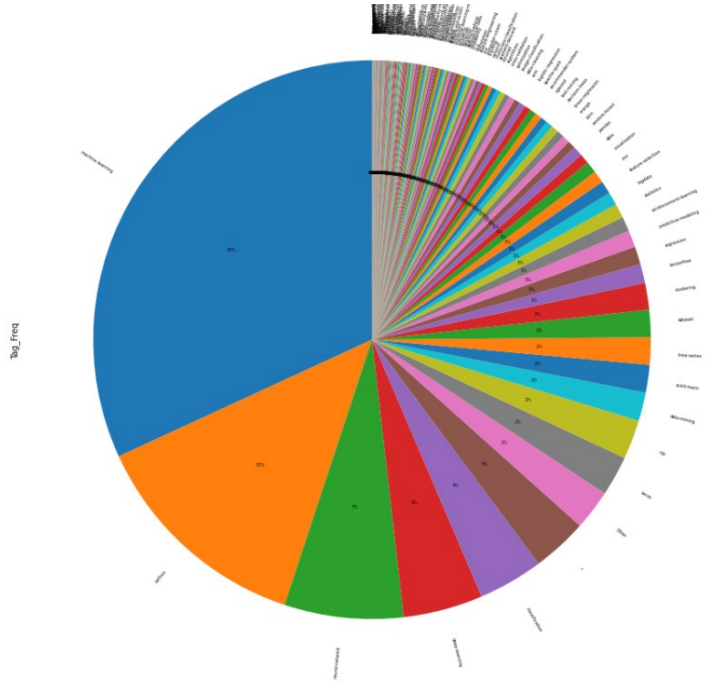


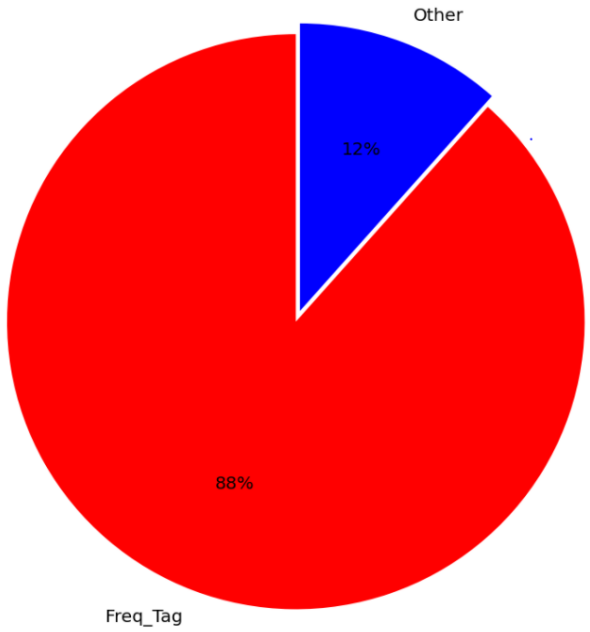



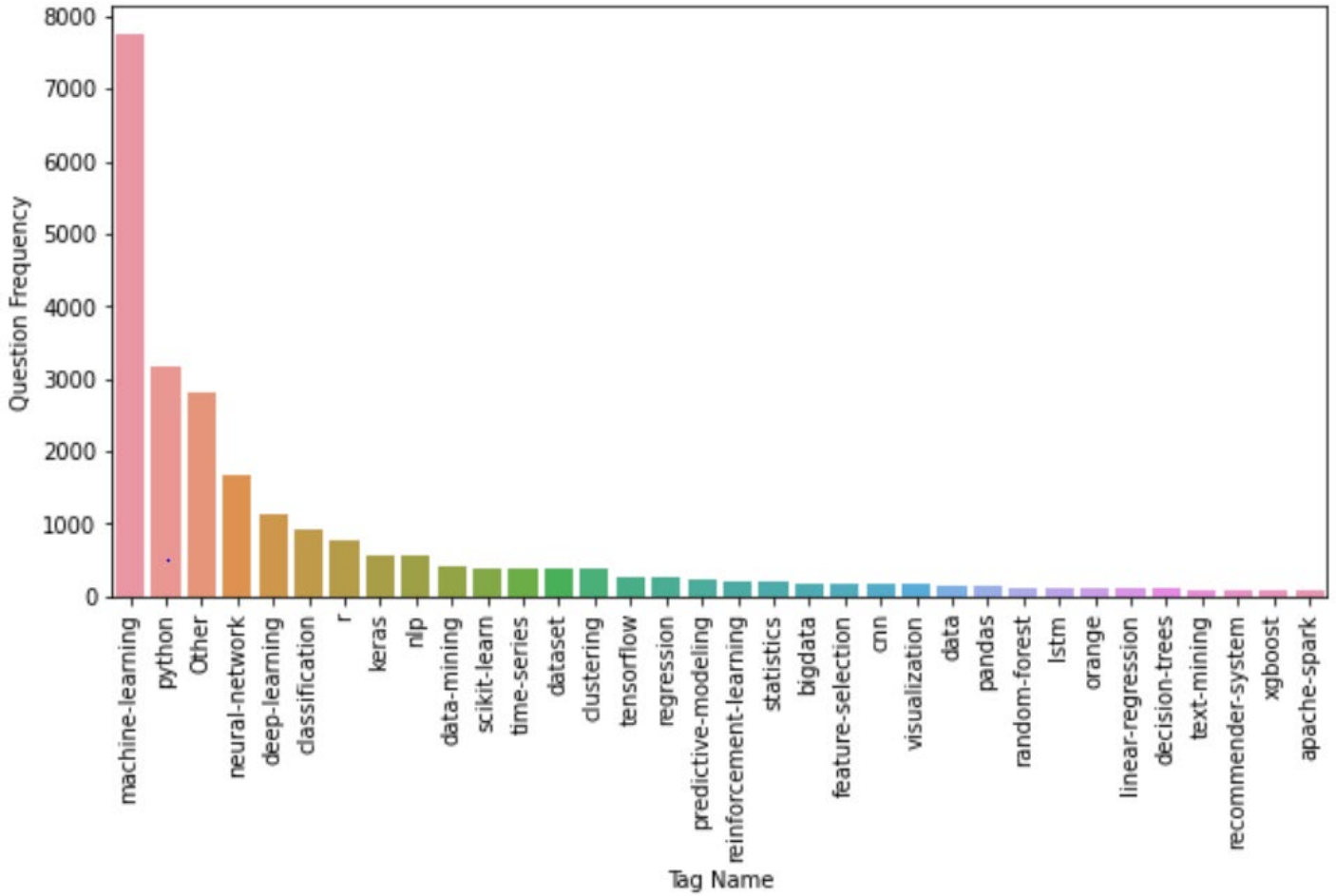
# Exploratory Data Analysis

- Use of all tags increasing noise in the data
- Limited tags to the primary tag for each question
  - 349 total primary tags
  - Top 33 ranked tags account for 88% of the data
  - Rest of tags were renamed to 'Other'





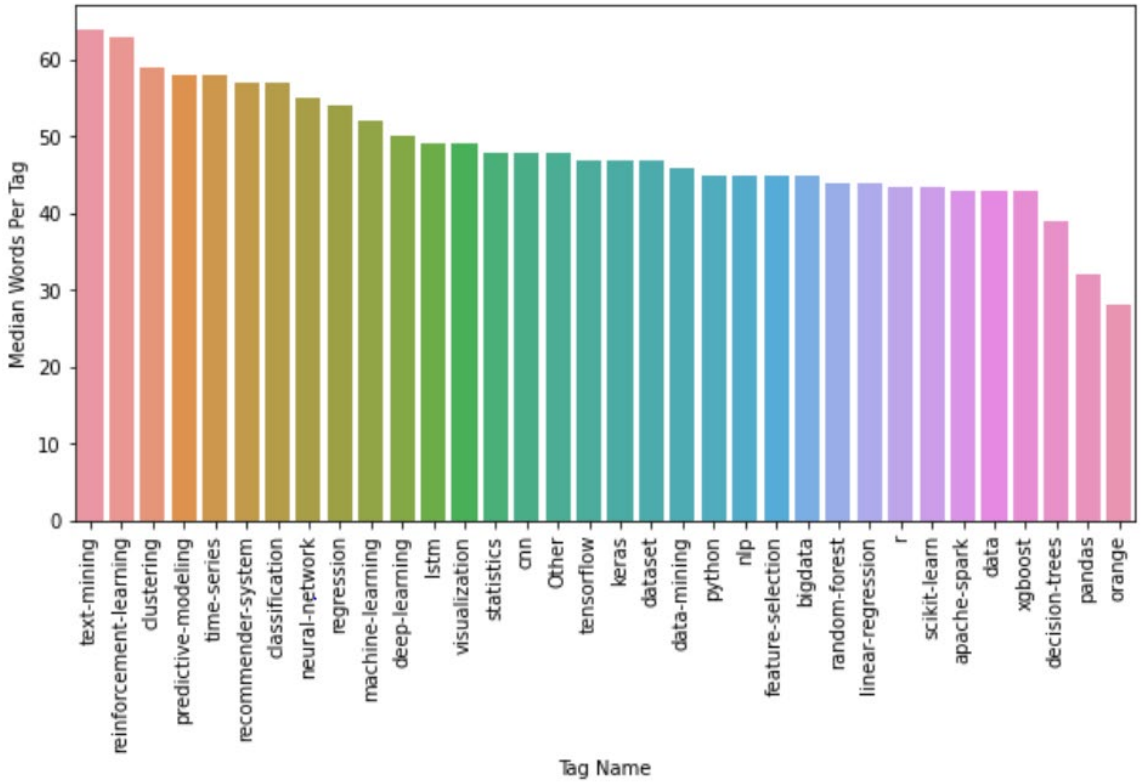
# Exploratory Data Analysis




Bar chart showing the frequency distribution of the ranked ordered first 33 tags. The distribution is highly imbalanced with the 'machine-learning' tag accounting for the majority of questions.

# Exploratory Data Analysis

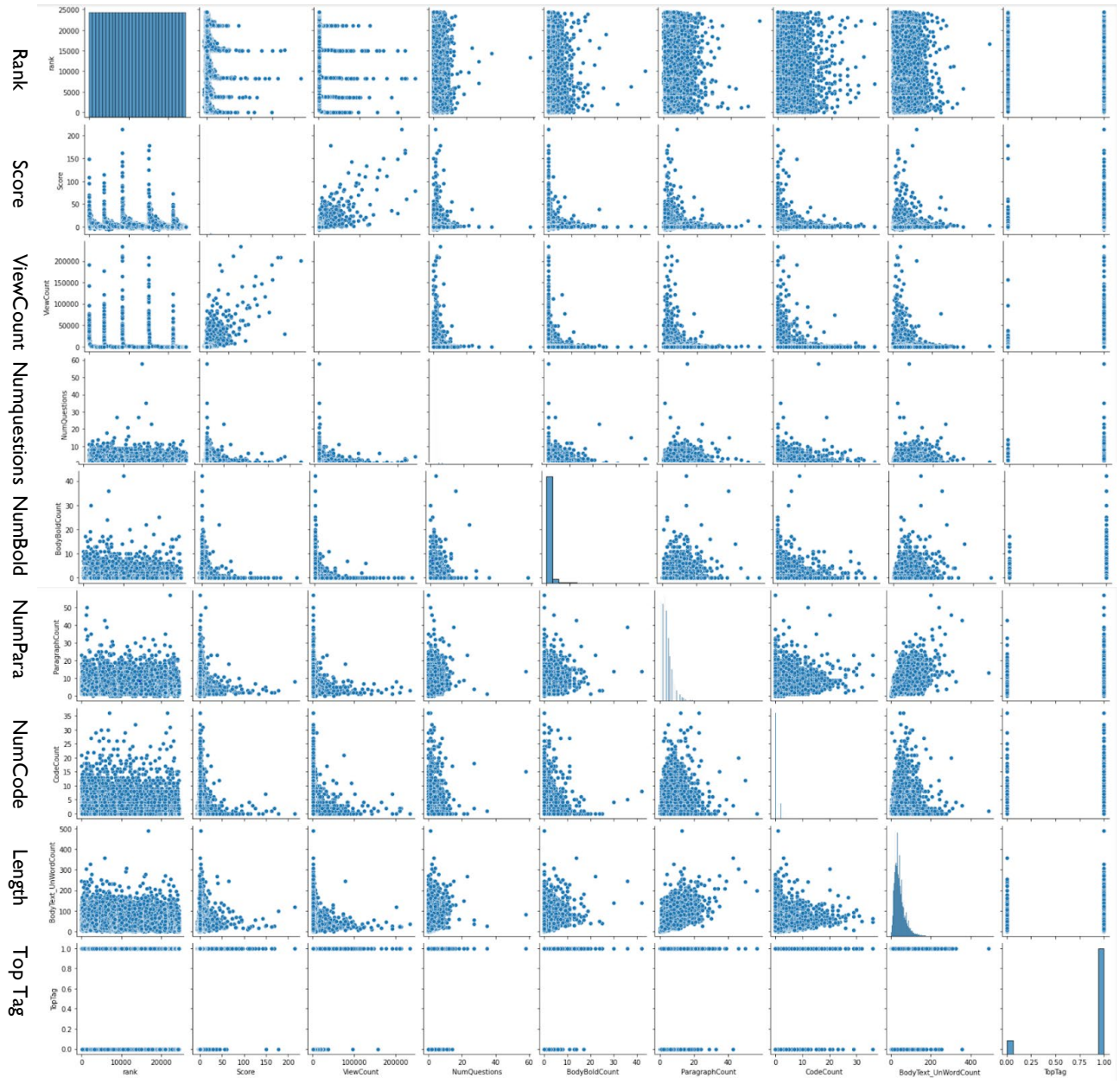
- Features created prior to text cleaning
- Work performed in CP2_03_Clean_Body_Text.ipynb & CP2_04_Analyze_Features.ipynb
  - Question length
  - # of paragraphs
  - # of question marks
  - # of code snippets
  - # of bold text




Bar chart showing the average question length by tag.

# Exploratory Data Analysis

- Seaborn pairplot
- Score and ViewCount are positively correlated
- Most other relationships are negatively correlated
- This may speak to different and unique styles of the question askers



# Text Transformation

- Work performed in CP2_05_Transformation.ipynb
- 3 different methods evaluated
  - Count Vectorization – scikit learn
    - 'Bag of Words' – simple count of words with no attention paid to order
  - TF-IDF – scikit learn
    - Calculation of word frequencies
    - Term Frequency – Inverse Document Frequency
    - Evaluates how relevant a word is to a document within a collection
  - Doc2Vec – Gensim
    - Most sophisticated method – extension of Word2Vec using Neural Network
    - Unsupervised - learns fixed-length feature vectors for each document
    - Detects relationship among words and understands the semantics of the text based on the context of each word around the sentence it is surrounded with

# Data Preparation / Evaluation

- Split into 70/30 train / test
- Further split train into 70/30 dev / val
- Scoring metric – ROC AUC
  - 'Receiver Operating Characteristic'
    - Summary of multiple confusion matrices (to evaluate all multi classes)
  - 'Area Under the Curve'
    - Measure of how much model is able to discriminate between classes

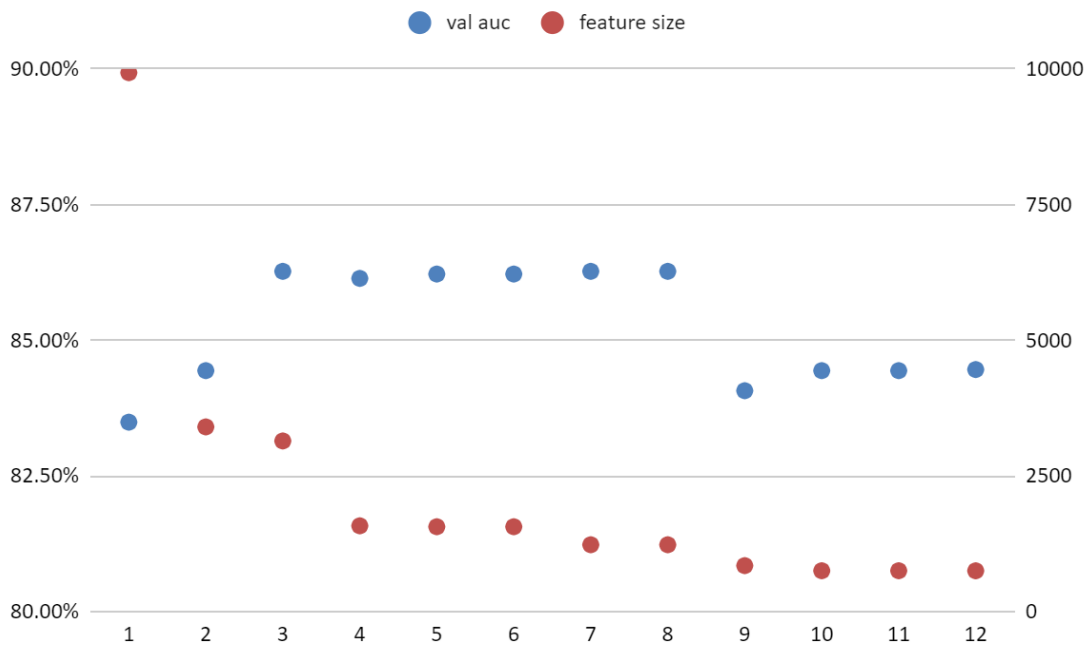
# Count Vectorization

- Default parameter run does not remove any words
- Resulted in dict size of 42953 unique words from development dataset
- Important Parameters
  - Min_df - removal of words that appear too infrequently
  - Max_df – removal of words that appear too frequently
  - N-Grams – inclusion of strings of words

# Count Vectorization

- Results of hypertuning using Random Forest base model
- 12 parameter combinations

| 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 |



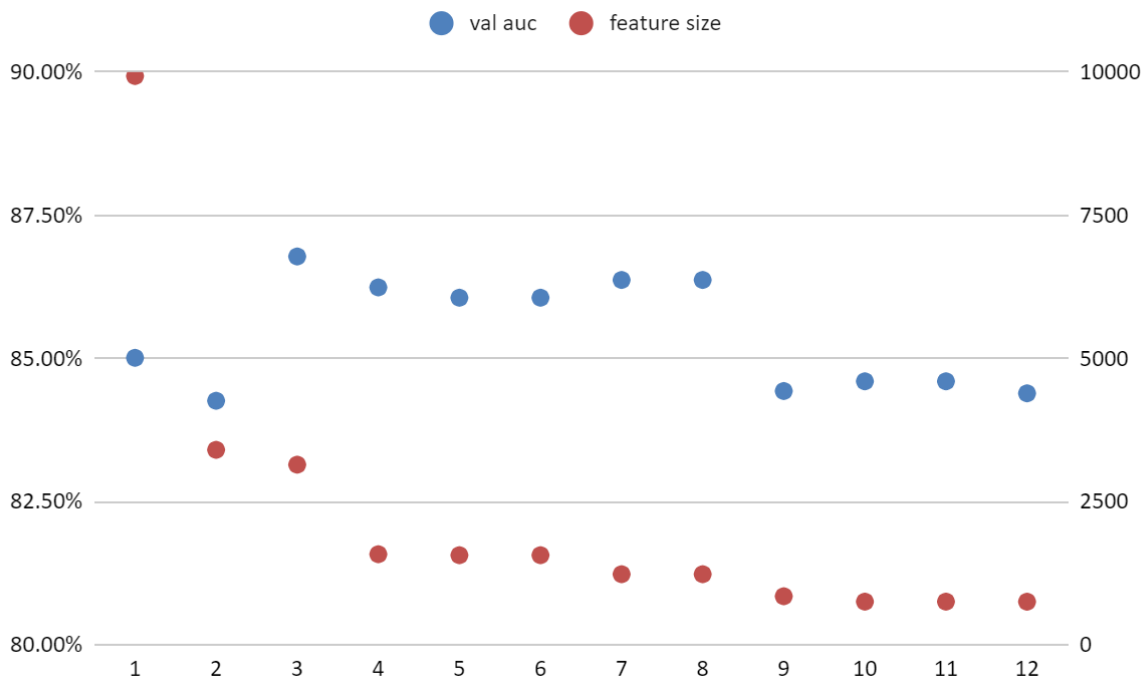
# TF-IDF

- Default parameter run does not remove any words
- Resulted in dict size of 42953 unique words from development dataset
- Important Parameters – Same as Count Vectorization
  - Min_df - removal of words that appear too infrequently
  - Max_df – removal of words that appear too frequently
  - N-Grams – inclusion of strings of words

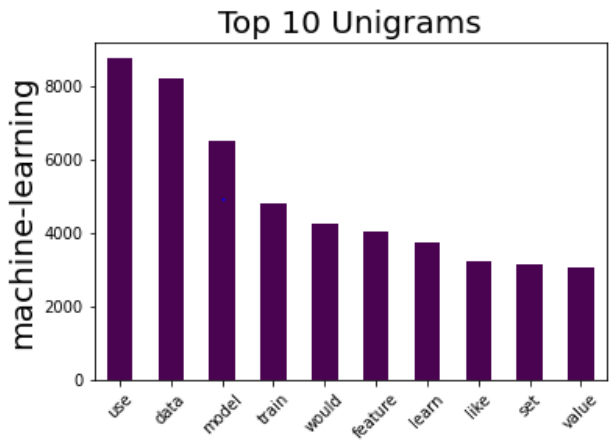
# TF-IDF

- Results of hypertuning using Random Forest base model
- 12 parameter combinations

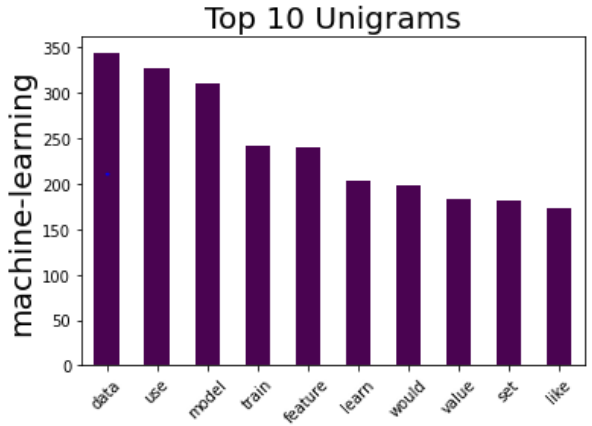
| Param Set # | min_df | max_df | ngram_range | feature size | train auc | val auc |
|---|---|---|---|---|---|---|
| 1 | 10 | 1000 | (1, 3) | 9931 | 1.00 | 0.8501 |
| 2 | 10 | 1000 | (1, 1) | 3403 | 1.00 | 0.8426 |
| 3 | 0.001 | 0.999 | (1, 1) | 3144 | 1.00 | 0.8678 |
| 4 | 0.005 | 0.99 | (1, 3) | 1580 | 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 |

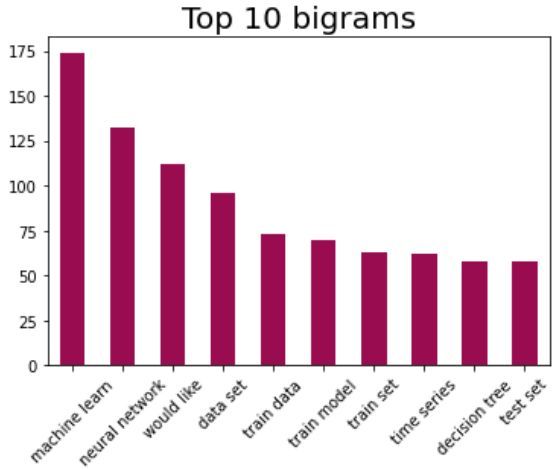


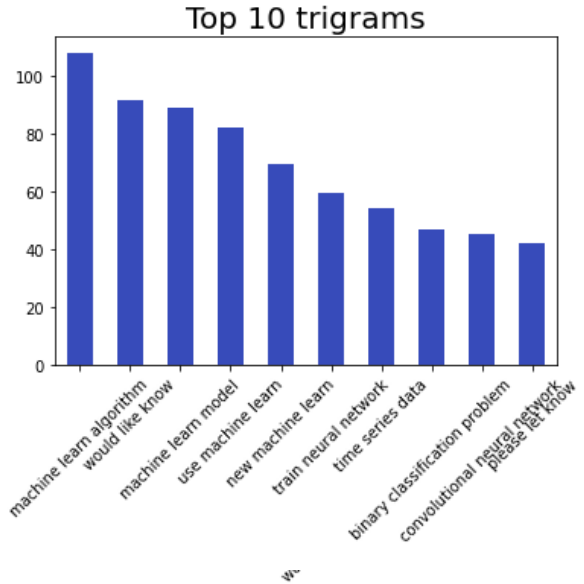
# Most Frequent Words by Tag - CV

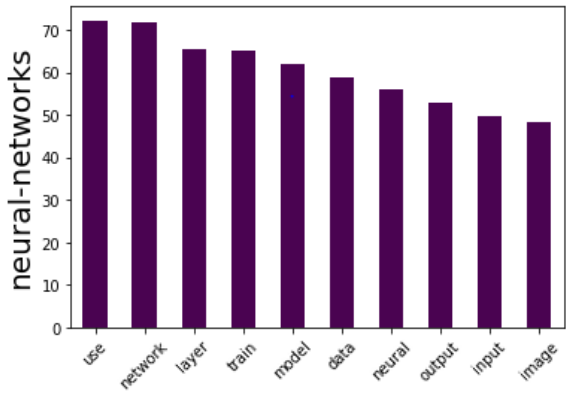


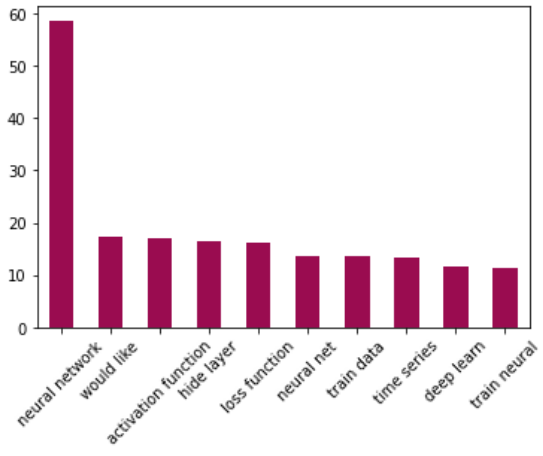
# Most Frequent Words by Tag - TFIDF

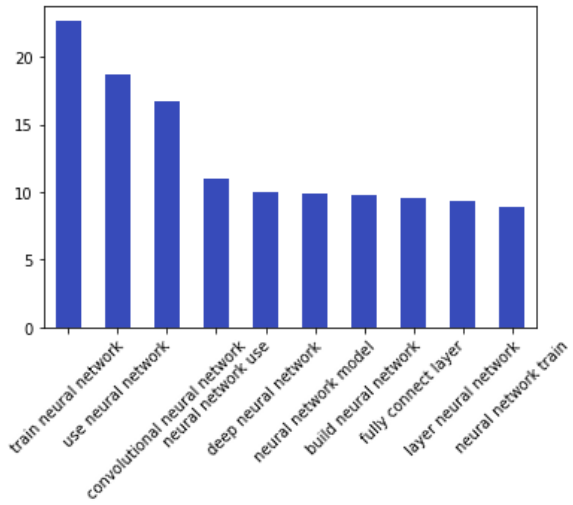


# Doc2Vec

- Different Pre-Processing Steps Prior to Training
  - Convert each data split to tagged document
  - Build the model with desired parameters
  - Build the vocabulary with the tagged documents
- Important parameters
  - Distributed memory setting – dm
    - 1= PV-DM – preserves word order
    - 0 = PV-DBOW – does not preserve word order
  - Vector Size – [10,25,50,100,200,500]
  - Window - max distance between the current and predicted word within a sentence – [1,2,3,4,5]
  - Hierarchical softmax – hs = 1 or 0
  - Epochs – iterations over corpus – [10,25,50,100]

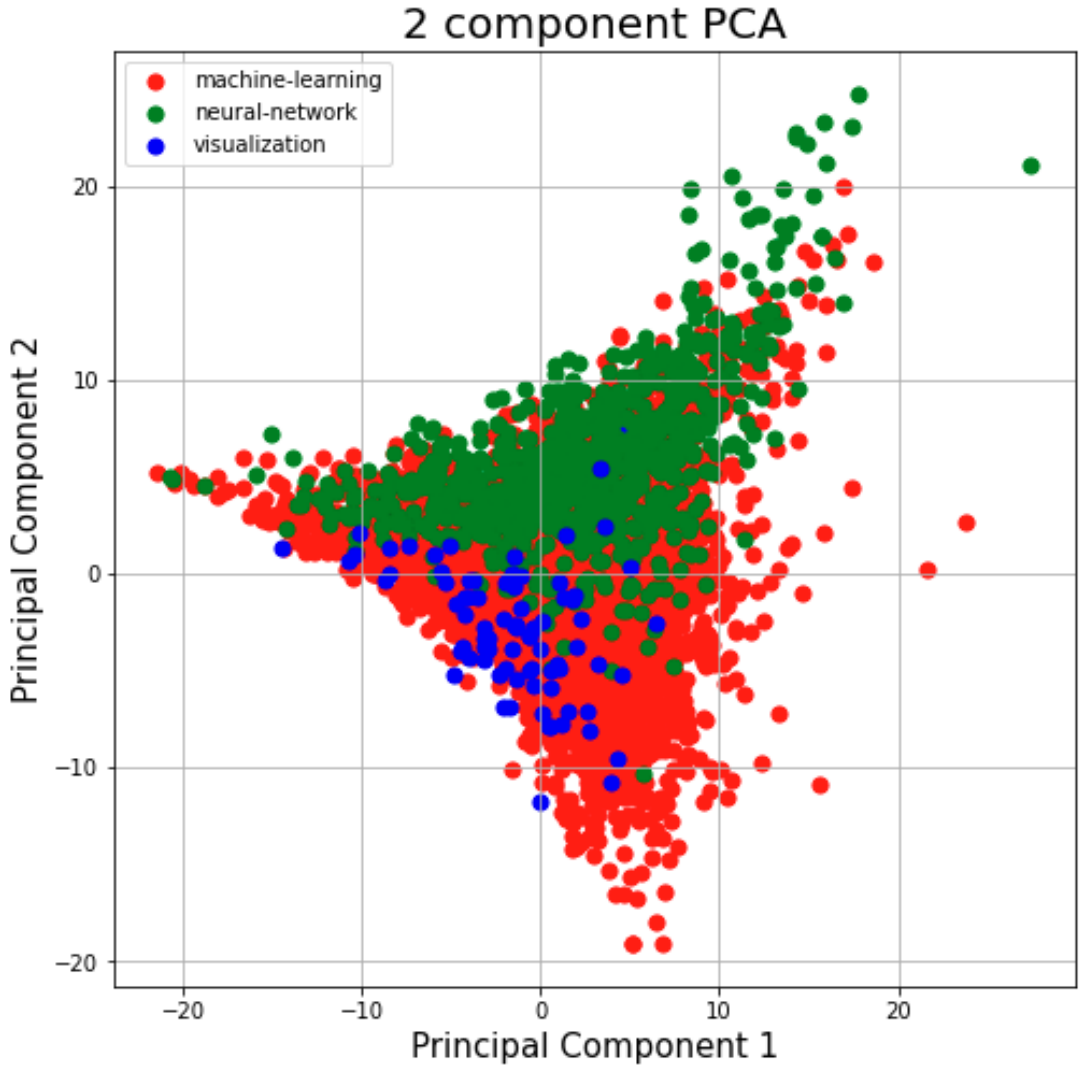
# Doc2Vec

| 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 |

- Results of hypertuning using Random Forest base model
- Hypertuning employed itertools to loop over parameter settings and score cross validation data set
- Top 10 highest scoring parameter combinations
- PV-DBOW implementation (dm = 0), scored better
- Model #2 used - larger vector size was predicted to regularize better than the first model

# Doc2Vec Results Visualization

- 2-Component PCA (principal component analysis) was calculated and plotted for 3 of the tags
- Shows nice segregation of the vector results for these tags



# Modelling

- Baseline Modelling MultiClass Classification Type
- Objective function is cross entropy
  - measure of the difference between two or more probability distributions for a given set of events
- 3 Different model types evaluated
  - Random Forest
    - Ensemble decision tree / bagging technique
  - Naïve Bayes
    - Probabilistic classifier based on Bayes Theorem
    - Gives the likelihood of the event occurring
  - LightGBM
    - Ensemble decision tree / gradient boosting technique

# Random Forest Results

- Work performed in P2_06_RandomForestModelling.ipynb
- Best transformation is TF-IDF with 1000 Estimators
- Highly Overfitted

| 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 |

- GridSearchCV Hypertuning: Train 0.9760 / Val .0.8836

# Bernoulli Naïve Bayes Results

- Work performed in CP2_08_BernoulliNaiveModelling.ipynb
- Best transformation is Doc2Vec
- Still Overfitted

| 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 |

- GridSearchCV Hypertuning: Parameters Used Were Best

# Light GBM Modelling

- Work performed in CP2_07_LightGBMModelling.ipynb
- 12 different parameters considered
  - 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, n_estimator
- Brute force method used to identify best parameters – grid method with 1000 random combination of parameter values
  - Same method applied to each text transformation type

# Light GBM Results

- Best transformation is Count Vectorizer
- LGBM CV and TFIDF methods have the lowest overfit of all models evaluated
- LGBM Doc2Vec has lower AUC scores and is highly overfitted

| 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 |

# Candidate Model Selection

- The following LGBM CV and TF-IDF parameter sets were chosen based on highest ROC-AUC scores and lowest overfit percentages

| 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% |

# Conclusions

- New method of automated tagging of the primary tag on Stack Exchange Data Science is viable
- Candidate model results using LGBM Model
  - ROC- AUC 85-88%
  - Overfit Range 2-5%
- Both accuracy and stability are important
- Success of 3 vectorization processes dependent on the model
  - Random Forest – TF-IDF
  - Bernoulli Naïve Bayes – Doc2Vec
  - LGBM – Count Vectorization
- Random Forest and Bernoulli Naive Bayes techniques produce lower ROC-AUC scores and are more highly overfitted

# Next Steps

- Because parameters are sensitive to model used:
  - Re-run text transformation hypertuning process with LGBM
  - Further hypertuning of the Doc2Vec method to reduce overfit
- Build model that can predict multiple tags now that we've succeeded at predicting the first most popular tag
- Create new NER dictionary of Data Science terms to improve the results of the Named Entity Recognition process