# **FINAL REPORT:**

# **StackOverflow - Tag Classification**

## **Problem Statement:**

An existing problem for StackOverflow is that just like any open web forum or developer community platform, there are millions of questions posted without any moderation or manual intervention of categorizing questions posted by the developer community, which makes it very hard for the community in general to explore which words contribute the most to which topics, and which topics contribute the most to which documents (questions on StackOverflow, in this case).

StackOverflow is a question-and-answer website for professional and programming enthusiasts.

* Every document is a mixture of topics
* Every topic is a mixture of words

Based on the problem statement, it is a complicated task to tag questions to a certain topic name, since more than often, the questions posted would be very open ended and every user has a different interpretation, semantic and syntax of what and how to post a question.

An effective data-oriented solution needs to be devised by developing models that identify and tag questions to a specific topic which could be used as a basis for future tag recommendation, when another developer posts a question on a related topic.

This would help keep the platform organized and make it a more collaborative platform for users to question, and answer on topics that they are interested in.

## **Criteria for Success:**

Identify and classify questions with most popular tags (top 10 tags) with high accuracy, precision and any other score/ metric that applies to the nature of this problem (depending on the type of classification problem we have in hand – in this case, it is Multi label classification model as I will explain below, so we will choose Jaccard index score)

## **Data**

[Kaggle data](https://www.kaggle.com/stackoverflow/stacksample)

**File descriptions**

* **Questions.csv**
  + Questions contains the title, body, creation date, closed date (if applicable), score, and owner ID for all non-deleted Stack Overflow questions whose Id is a multiple of 10.
  + **Format** - CSV file that has 7 columns
* **Answers.csv** 
  + Answers contains the body, creation date, score, and owner ID for each of the answers to these questions. The ParentId column links back to the Questions table.
  + **Format** - CSV file that has 6 columns
* **Tags.csv**
  + CSV file that has 2 columns - TagId and feature tags that need to be mapped to the question text

## **Data Wrangling**

**Main steps:**

* Data Extraction
* Data Cleansing

The aim of this Data Wrangling exercise was to perform text processing on ‘Title’, ‘Q\_Body’ and ‘A\_Body’ columns of the dataframe which is going to be cleaned, pre-processed and broken down into independent categorical features (columns) before performing modeling on the ‘Q\_Body’ column data as against the ‘Tag’ column which would be the dependent feature for this dataset.

**Essential steps** performed were:

* **Merge training and test data set** 
  + Total question entries **– 2014K**
* **Missing data manipulation and duplicates**
  + Since there are limited columns in the dataframe, finding reason for why the data was missing and handling it using techniques other than removing those entries didn’t make sense. Hence, null data handling was done by simply removing null entries of 'target' column of the dataframe.
* **Converting all letters to lower or upper case**
  + For the sake of consistency of text in the dataframe.
* **Remove html tags**
  + In this case, only html tags i.e. markdowns that are presented for each question when a developer posts questions, so removing them would be helpful.
* **Removing punctuations**
  + This step allowed me to remove this set of symbols [!”#$%&’()\*+,-./:;<=>?@[]^\_`{|}~] which essentially would be not very helpful for the classification of text in the Question columns.
* **Removing stop words, sparse terms, and particular words**
  + “Stop words” are the most common words in a language like “the”, “a”, “on”, “is”, “all”. These words do not carry important meaning and are usually removed from texts. It is possible to remove stop words using Natural Language Toolkit (NLTK), a suite of libraries and programs for symbolic and statistical natural language processing.
* **Stemming using NLTK library**
  + Stemming is a process of reducing words to their word stem, base or root form (for example, books — book, looked — look). I applied this step to the dataframe using Porter stemming algorithm that removes common morphological and inflexional endings from words.
    - <https://pythonprogramming.net/stemming-nltk-tutorial/>
* **Other steps included**
  + **Removing white spaces and newline characters**
  + **Removing entries with small text length (wouldn’t aid the analysis of the project)**
    - Judgement was to get rid of entries with question title length < 10 characters.

**Data Wrangling Output data**

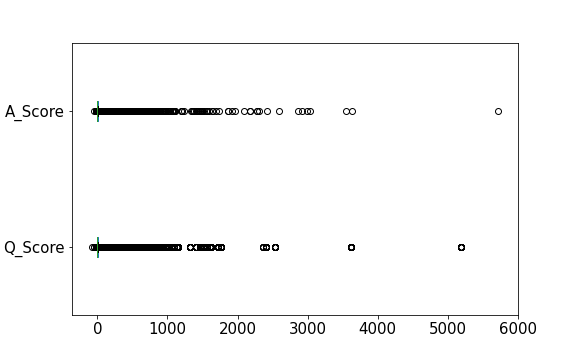
After data extraction and cleansing, the final dataset contains 1455K – 28% of the dataset was trimmed as a part of the Data wrangling exercise. This cleaned dataframe will be further used for EDA, pre-processing and modeling steps

## **Exploratory Data Analysis**

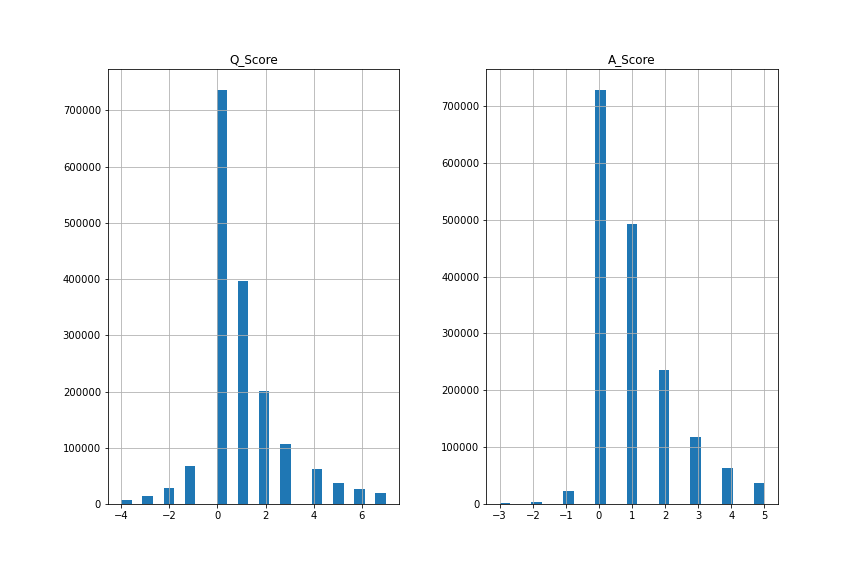
Conducted EDA on the cleaned dataset

**Inferences**

* Answer and Question Scores distribution – number of votes submitted for answer and question respectively

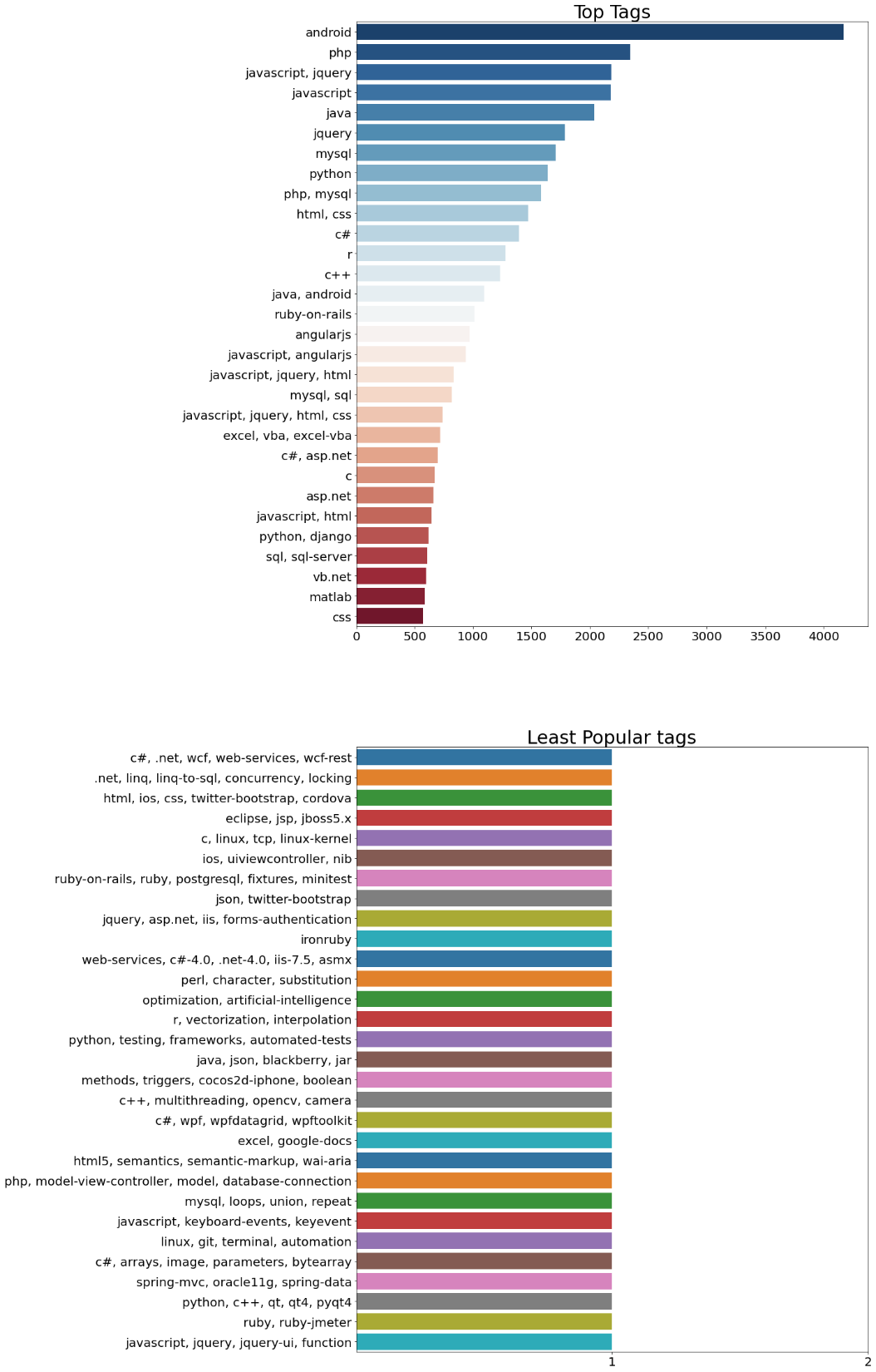


**Image-1: Non-Scaled Scores**

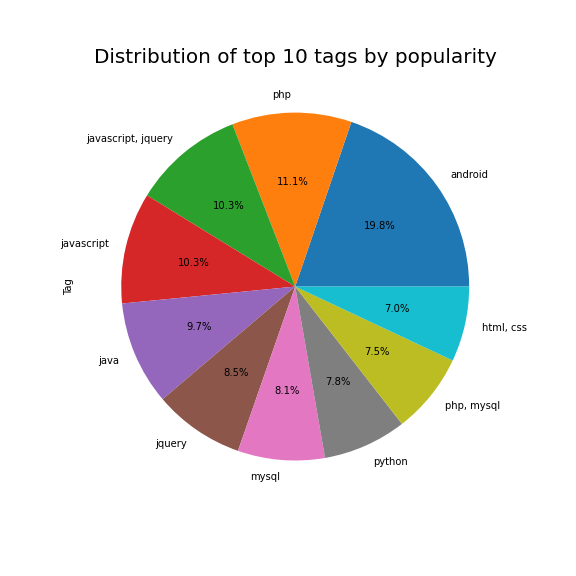


**Image-2: Scaled Scores**

* As we see in the image, the merged dataset consists of multiple tag(s) that were associated to question. Here is the list of most popular and least popular tags

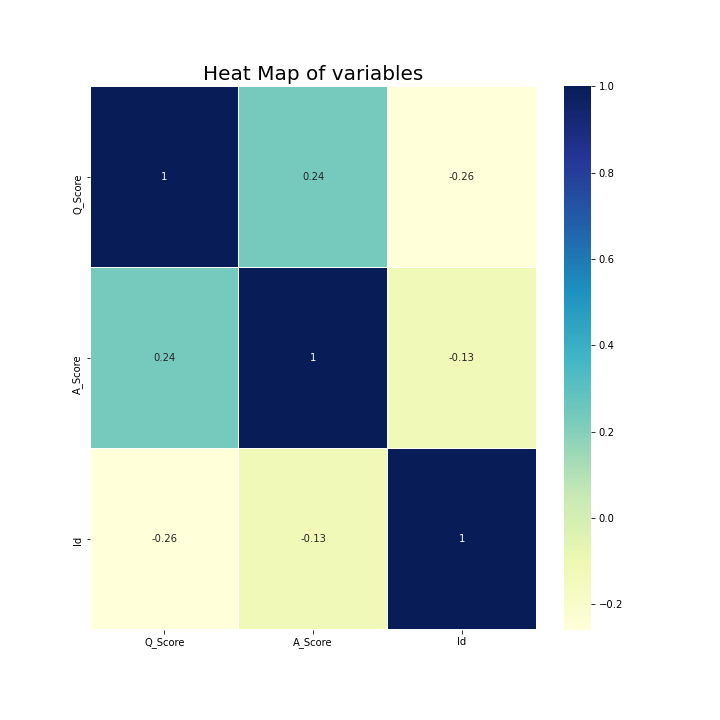


**Image-3: Most popular & Least popular tags**



**Image-4: Distribution of top 10 tags by popularity**

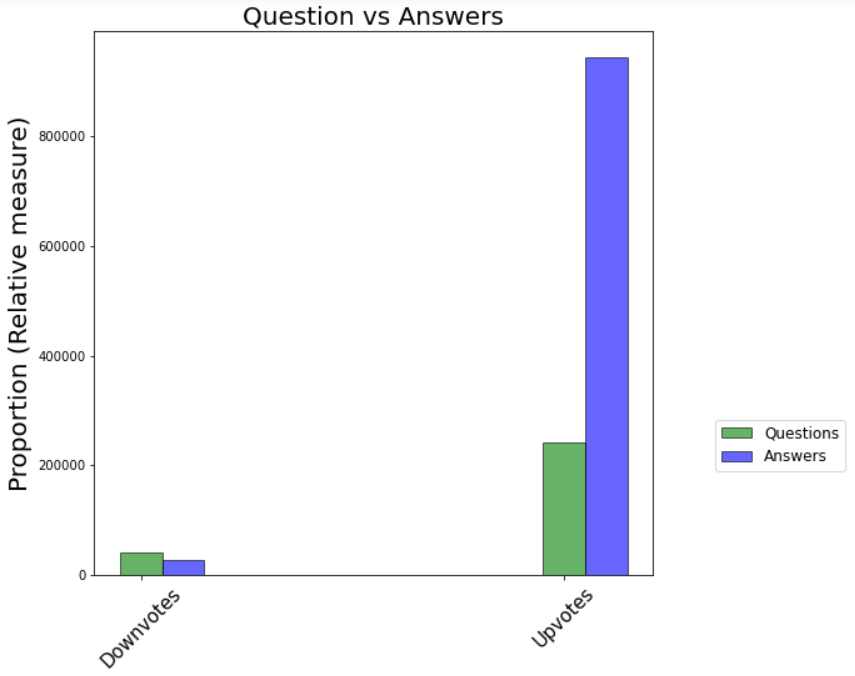
**Conclusion -** Android is No.1, in terms of developer community support based on questions posted



**Image-5: Heat Map of Numeric variables**

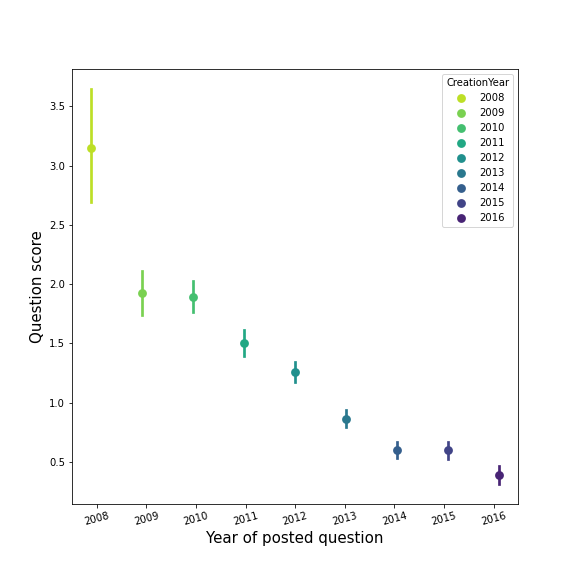
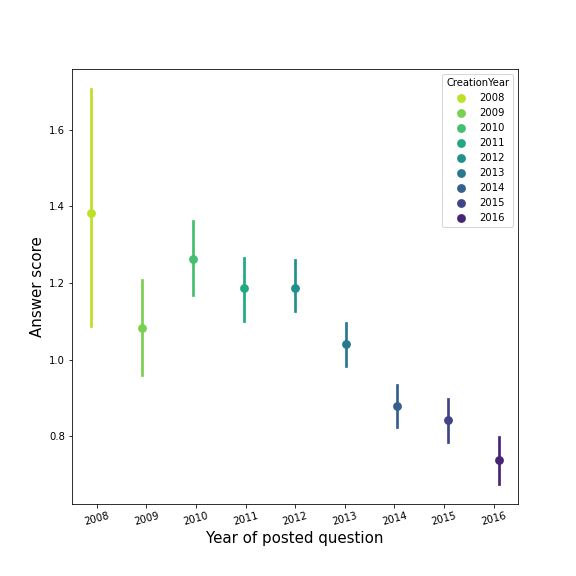
**Conclusion -** There is no strong correlation between question-and-answer scores

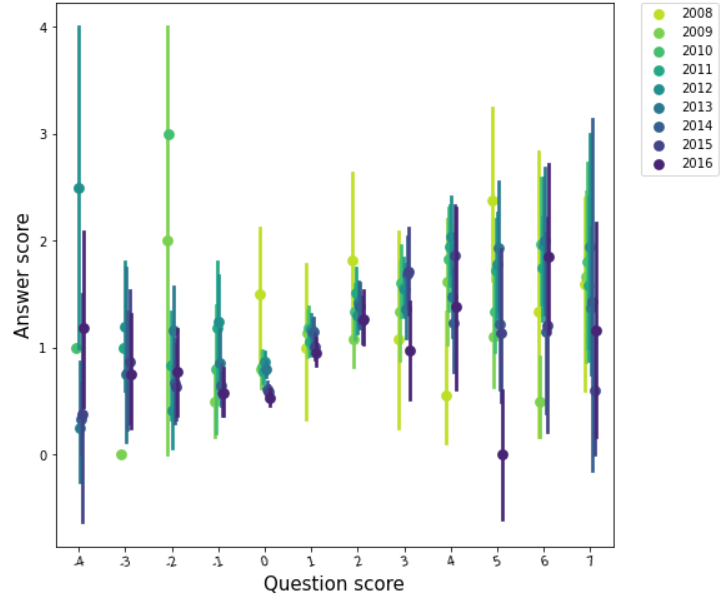
* There is a higher % of downvotes to questions, than to answers.



**Image-6: Relative proportion of Upvotes and Downvotes by Q&A**

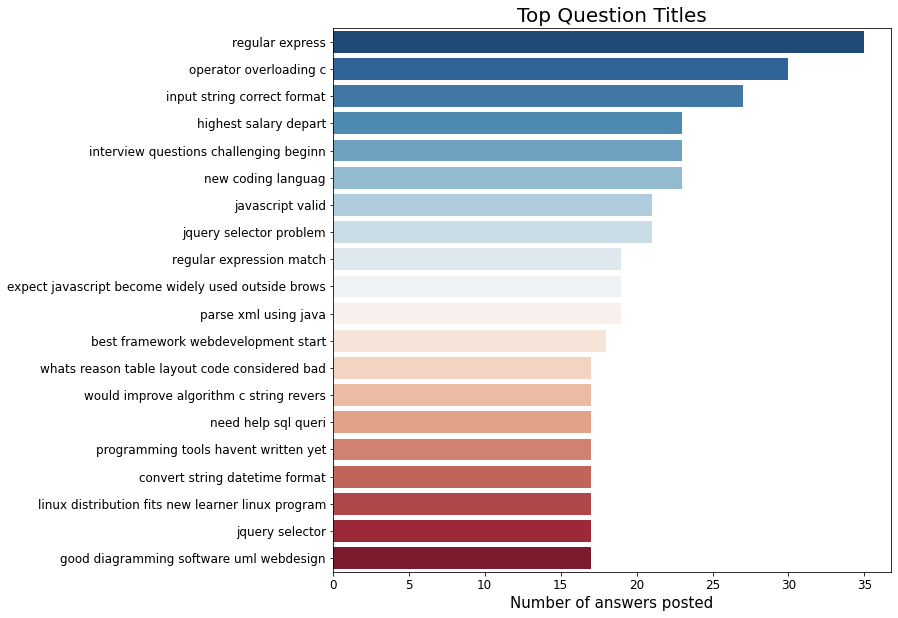
* Figures below represent
  + Question scores by year of posted question
  + Answer scores by year of posted question
  + Question and Answer score comparison

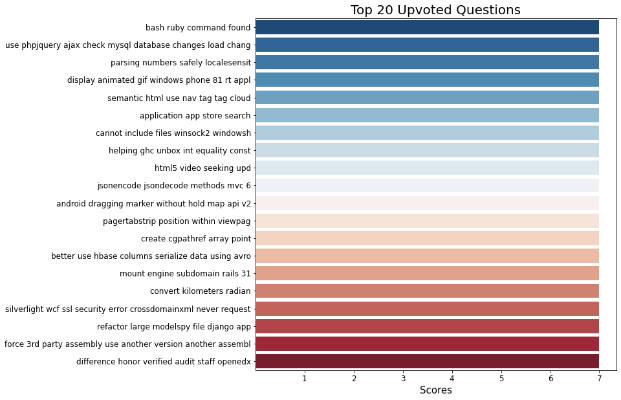
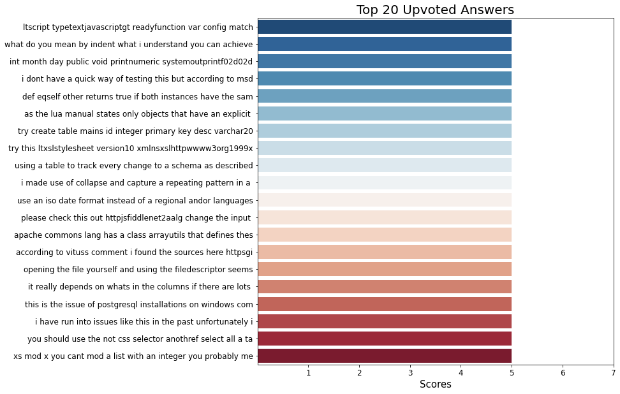


**Image-7: Relative proportion of Upvotes and Downvotes by Q&A**

**Conclusion -** Seems like the average scores for both question and answers have reduced over time. Could be possibly due to a different scoring mechanism which could have possibly changed later OR the initial enthusiasm of yesteryears where developer community used to more actively upvote on questions and answers which eventually started reducing over time

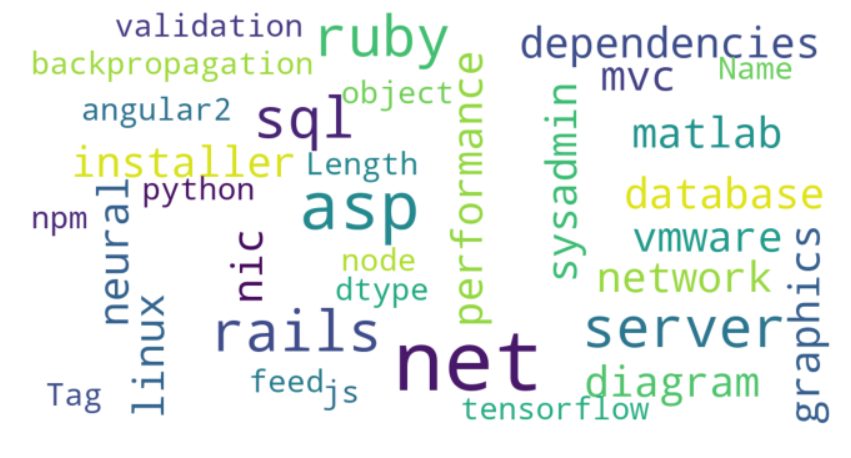


**Image-8: Top Question Titles (based on # of answers posted)**

**Image-9: Top 20 Upvoted Questions and Upvoted Answers**

* **Word clouds** 
  + Word cloud for 'Tags'
  + Word cloud for 'Questions (Title AND Body)'
  + Word cloud for 'Answers'



**Tags**



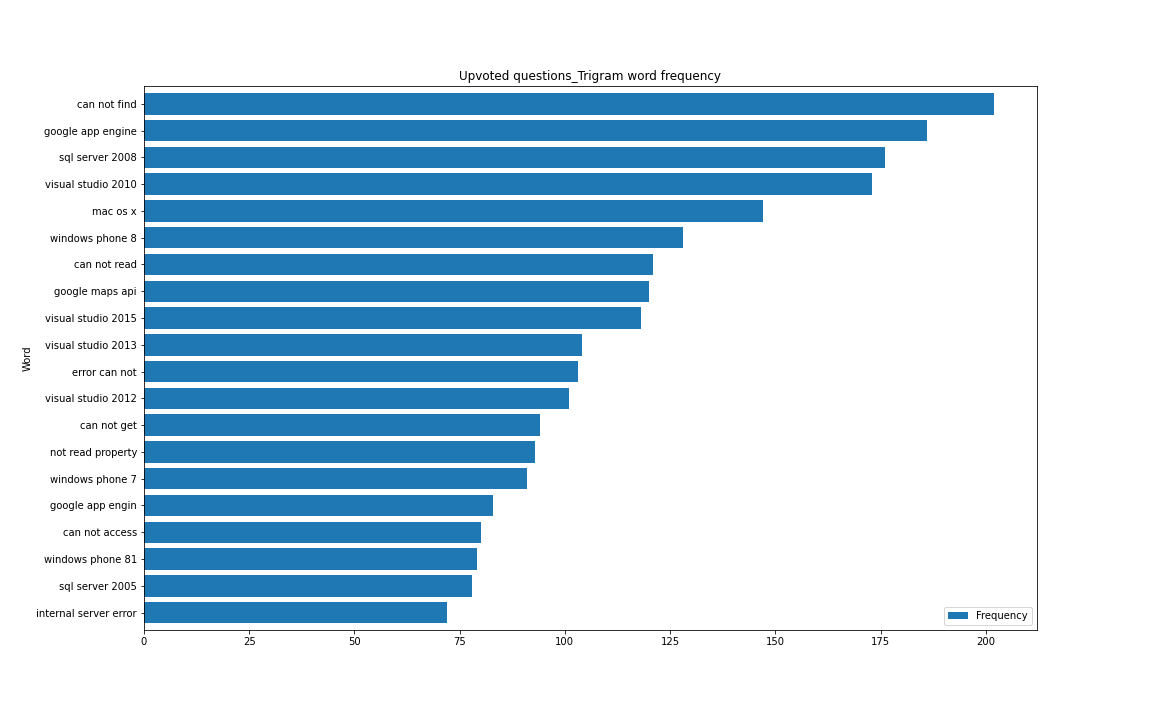
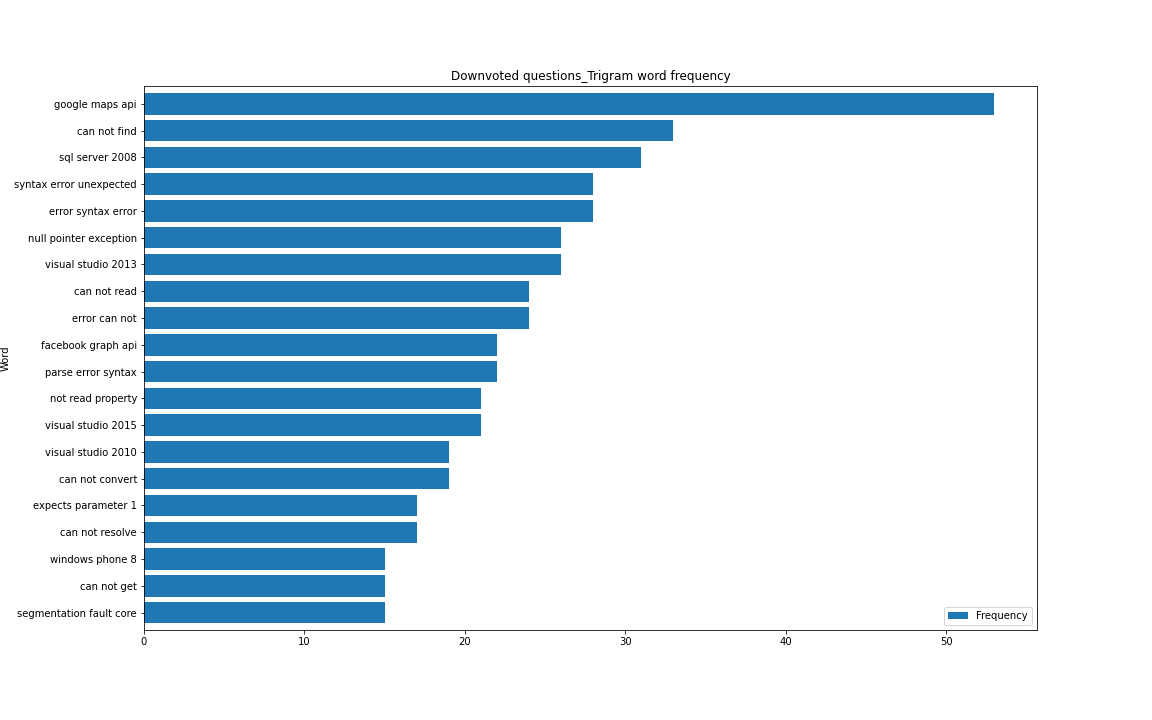
**Question Body and Title**



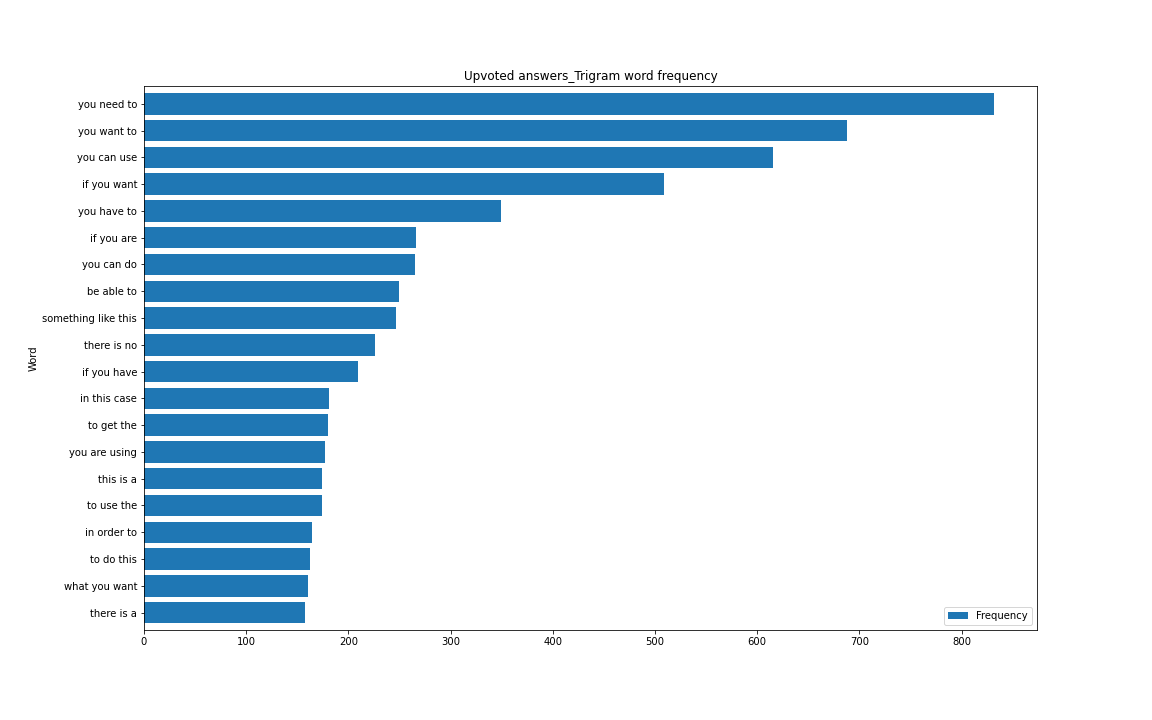
**Answer Body**

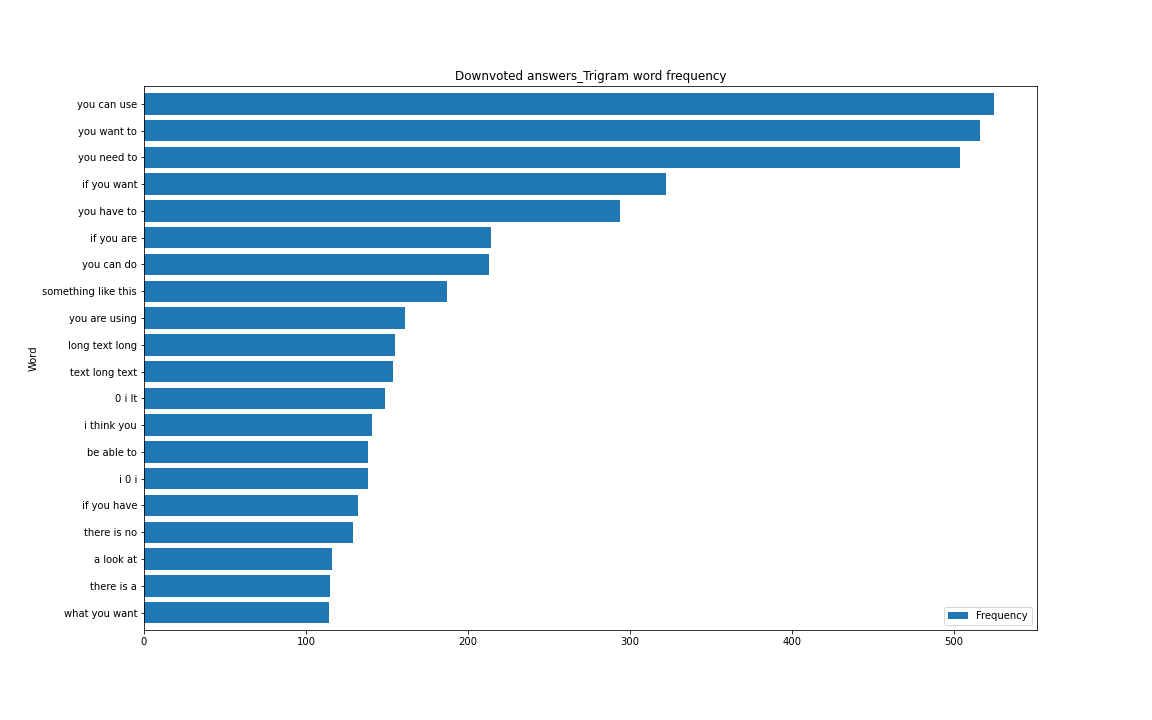
**Image-10: Word Clouds of textual columns**

* One step was to plot trigrams of words across the data set – for both Upvoted and Downvoted questions and answers
  + To do this, 2 dataframes were created that were split from the cleaned dataframe classified by upvoted questions (i.e. score > 0) and downvoted questions (i.e. score < 0)
  + Then, the most frequent occurrences of trigrams were determined, by plotting a horizontal bar chart – See image 11 (for upvoted and downvoted questions) and 12 (for upvoted and downvoted answers)

****

**Image-11 –Trigram of word frequency (for Upvoted and Downvoted questions)**





**Image-12 – Trigram of word frequency (for Upvoted and Downvoted answers)**

## **Pre-processing and Training data development**

Preprocessing, training data development and Modeling

Objective is to perform pre-processing and build a model that will be able to identify questions posted by developer community which have top 10 tags based on binary output – 0 , 1 where 0 means top 10 tag not present, 1 means top 10 tag present. This is especially important for organizations that have a list of 10 top technologies or languages (tags) used across the board and build a learning community where they are looking to identify questions and provide learning bootcamps with a rich library of articles around those languages to their associates

* C#, Java, PHP, JavaScript, C++, .Net, ASP.Net, HTML, IOS, Ajax

Steps performed to achieve are as follows:

* Pre-processing
* Training data development
* Modeling

**Steps involved:**

* Standardizing the magnitude of numeric features using a scaler function
  + Standardization of numeric feature – ‘A\_Score\_scaled’ and ‘Q\_Score\_scaled’ in the dataset using MaxAbsScaler
* Performing TFIDF on categorical feature – ‘Q\_Body’ to fit and transform data
  + Tfidf involves ignoring common words which was applied on the dataset breaking down the column into 200 categorical features
* Picking only records from dataframe that have either of the **top 10 tags** 
  + The pre-processed column now has **181159** records
* Converting categorical variables – ‘Tags’ into numeric variables using one hot encoding and combining to the tfidf vectorized columns
  + This generated 5 columns (Target variable) with values 0 OR 1
    - **Note**: 5 columns are generated,
      * **Reason**: Each question that is returned after weeding out records with questions that didn’t have top 10 tags – had maximum 5 tags
  + Since there are 5 columns as a part of target variables, this is a multi label classification problem that we have in hand
* Splitting the standardized and vectorized dataset (after performing above 2 steps) into test (30%) and training (70%) datasets
  + 126811 entries in training dataset
  + 54348 entries in test dataset

## **Model Selection (Algorithms & Machine Learning)**

This is a multi-label classification problem, in supervised learning. Here I applied the following classification models on the training dataset after breaking the training dataset consisting of 126K StackOverflow questions into training and validation data

To solve multi-label classification problem, we can decompose it into multiple independent binary classification problems (one per category) using “one-to-rest” strategy, where we will build multiple independent classifiers and, for an unseen instance, choose the class for which the confidence is maximized.

The main assumption here is that the labels are mutually exclusive. Models applied are:

* tree.DecisionTreeClassifier()
* KNeighborsClassifier(),
* MLPClassifier()
* LogisticRegression(),
* RandomForestClassifier()
* GradientBoostingClassifier()

### **Model Evaluation**

* Used metrics like Jaccard Score, Hamming loss which are highly preferred evaluation metrics for multi label classification models
  + <https://en.wikipedia.org/wiki/Multi-label_classification#Statistics_and_evaluation_metrics>
* Hamming loss: the fraction of the wrong labels to the total number of labels
* Jaccard index score, also called Intersection over Union in the multi-label setting, is defined as the number of correctly predicted labels divided by the union of predicted and true labels

### **Model comparison**

**Scores**

* DecisionTreeClassifier() Classifier (Clf: OneVsRestClassifier)
  + Jaccard score: 27.03508909647539
  + Hamming loss: 9.772123951954162
* KNeighborsClassifier() Classifier (Clf: OneVsRestClassifier)
  + Jaccard score: 24.000334616028105
  + Hamming loss: 9.551344389833627
* MLPClassifier() Classifier (Clf: OneVsRestClassifier)
  + Jaccard score: 24.49668524016757
  + Hamming loss: 9.758456645727652
* LogisticRegression() Classifier (Clf: OneVsRestClassifier)
  + Jaccard score: 0.05601194921583271
  + Hamming loss: 8.44166425736589
* RandomForestClassifier() Classifier (Clf: OneVsRestClassifier)
  + Jaccard score: 8.797956743832192
  + Hamming loss: 8.822246169211764
* ExtraTreeClassifier() Classifier (Clf: OneVsRestClassifier)
  + Jaccard score: 27.546250834675362
  + Hamming loss: 9.696428102084266
* Seems like OnevsRestClassifier applied on ExtraTreeClassifier() classifier model performs best followed by DecisionTreeClassifier() and MLPClassifier() models, so we will perform hyperparameter tuning using GridSearchCV() on the ExtraTreeClasifier() and MLPClassifier()
* LogisticRegression and GradientBoostingClassifer are the worst and shouldn't be even considered
* We can keep DecisionTreeClassifier() out of the scope since Decision tree model is more susceptible to overfitting and performance erosion as number of trees increase, we will perform GridSearchCV and hyperparameter tuning on the other 2 models.

**Conclusion** - Based on the Jaccard score and reasoning above, MLPClassifier() and ExtraTreeClassifier() are chosen and in the next section, I will try to perform GridSearchCV and hyperparameter tuning on these 2 models. Also, the following metrics will be calculated

* Training time
* Prediction time
* Jaccard score (index)
* Hamming loss
* F1 score
* Precision score
* Accuracy

### **Hyperparameter Tuning**

Applying Grid search CV for hyperparameter Tuning:

* Additionally, hyperparameter tuning was done on the best performing shortlisted model(s) – in this case, MLPClassifier() and ExtraTreeClassifier() to enhance performance.
* Also, Cross validation using Grid search was applied to pick up random pairs of training and validation data by splitting the training set into k smaller sets, where a model is trained using k-1 of the folds as training data and the model is validated on the remaining part.
* **MLPClassifier()**
  + Training time is: 55.17 seconds
  + Prediction time is: 0.16 seconds
  + Jaccard score: 0.34
  + Hamming loss: 8.45
  + F1 score: 0.006
  + Precision score: 0.373
  + Accuracy: 0.577
* **ExtraTreeClassifier()**
  + Training time is: 174.32 seconds
  + Prediction time is: 2.98 seconds
  + Jaccard score: 0.80
  + Hamming loss: 8.45
  + F1 score: 0.016
  + Precision score: 0.441
  + Accuracy: 0.577
* Accuracy for **MLPClassifier()** model based on GridSearchCV and hyperparameter tuning applied comes out to 0.577 which is same as **ExtraTreesClassifier()**, however for this purpose since the Jaccard score is so high ~0.80 for **ExtraTreeClassifier()**, we will clearly choose that as the final model to see how well the scores are after applying the model on unseen dataset.
* It was observed that Accuracy and Precision scores are decent but not that great maybe is a good thing since there is no sense of overfitting done.
* F1 score is significantly low – close to 0 and is insignificant for the purpose of
* Jaccard score for **ExtraTreesClassifer()** is very good and we will apply this model on the unseen test dataset to see what the final scores look like.

**Conclusion** – We will choose Jaccard index score as the final metric to evaluate performance of the model on unseen dataset

Also, you can see in confusion matrix,

[[21826 166]

[15924 131]]

* There are still a lot of misclassified tags (OR questions not classified as top 10 tags) i.e. false negatives which is the nature of such problems since there is always a confusion around Stackoverflow users posting tags for questions without being sure of the tag, which is the challenge that Stackoverflow still faces today. However, the Jaccard score is great and since this is a multi-label classification model, we will go ahead and save the model to reapply on unseen test datasets

### **Save the model**

After optimizing the ExtraTreeClassifier() model, the model was saved for future application

### **Prediction on unseen test data**

The scores below are pretty, which is conclusive that model selection was good and the classification done using ExtraTreeClassifer() model is a good indication of a significantly accurate prediction on unseen test data set consisting of 54353 stackoverflow questions

This model was applied on the test data set to achieve the following scores:

* Training time is: 87.53 seconds
* Prediction time is: 4 seconds
* Jaccard score: 89.80
* Hamming loss: 0.869
* F1 score: 0.946
* Precision score: 0.999
* Accuracy: 0.957

**Conclusion** – All scores have drastically improved.

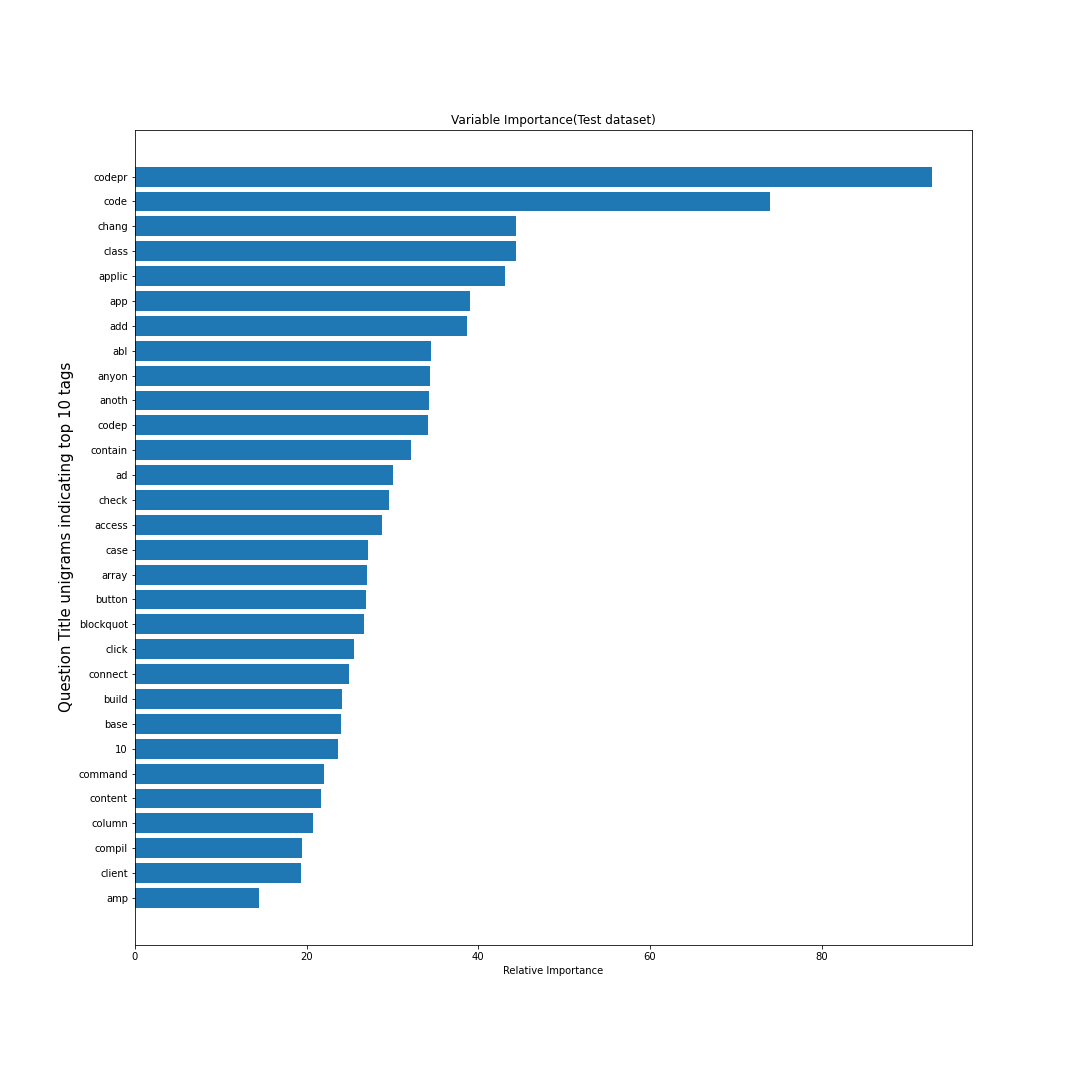
* The prediction time on unseen test data went slightly up from 4.6 seconds to 6 seconds which is not a dealbreaker
* As you can see in Confusion Matrix,

[[31165 20]

[ 2344 20824]]

* There are a fewer number of misclassified top 10 tags across questions (%), i.e. false negatives
* There was a significant improvement in F1 score.
* Accuracy – 0.957 is around the same as the one we derived on validation data set
* Jaccard scores reduced a bit – 0.89, however it is still a great score

### **Feature Importance**

When we have a list of questions that have titles like the ones below, there is a high likelihood that they belong to the top 10 tags. So, when an organization that is aiming to build somewhat like a learning experience platform for their developers who are expert in those top 10 languages, they can focus on extracting questions across different forums and narrow down to the ones that are classified as ‘1’ based on application of the final model on the dataset (Extracted from different developer communities)

**Image-13 – Question title unigrams indicating top 10 tags**

### **Future Research**

* Apply a different training, test data split instead to further improve scores across different classification evaluation metrics.
* Hyperparameter tuning can be performed on MLPClassifier model but limited scope of this project to only tuning ExtraTreeClassifier.
* Perform hyperparameter tuning on additional parameters than just criterion, max\_depth, max\_features, n\_estimators which could additionally improve the scores, especially the true positive score
* Apply Cross Validation using GridSearchCV across all models to be able to conclude on the best model.
* Choose more than 200 max features aka categorical features for modeling step.
  + For this project scope, I chose 200 features due to CPU constraints causing overcommit memory issues.
  + Choosing 500-1000 max features would ensure modeling is performed on a dataset without getting rid of few important features that could have possibly been trimmed in this effort
* Use word embeddings to analyze semantic and syntactic similarity, relation with other words for better classification and model results.
* Target another business problem like predicting tags associated to questions that belong to unseen test dataset.