**Term Project: Heterogeneous Data**

Sushil Paneru (V00936990)

Rajneesh Gulati(V00949939)

## **Design Choice and data models**

1. **ETL**

As part of extract phase, we parsed the xml file and converted them to pandas dataframe. We then loaded the dataframe into a relational database. We also created a trie for efficient text match, created a knowledge graph for computing similar users and word2vec to compute the relevance of a post.

Since, we are concerned more about OLAP rather than circumventing deletion, addition anomaly, our objective was to denormalize the tables as much as we can to reduce the number of joins and speed up the performance. Sophisticated relational modelling was not required for our insights, instead we used trie and dataframe API to perform filter and group by operations to extract temporal insights. As part of relational modelling, we removed muti-valued columns (tags column in post table) and used trie instead, and also added indexes on columns to speed up search operations.

We have used graph, text and relational model to generate our insights; details about the use of data models is given as part of visualizations.

## **Visualization**

1. **Popularity of python libraries and their relative trend**

Hypothesis: With time, newer deep learning libraries related post are created more than scikit. Post related to scikit need not be created because older post can be reused.

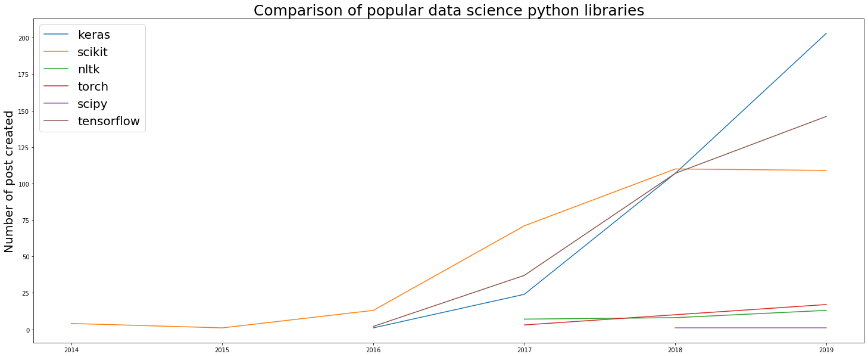


Fig: Popularity with respect to number of posts created

With the rise of deep learning in recent years, it can be seen that the number of posts created for framework like keras and tensorflow has steeply increased after 2017, whereas number of posts for scikit (machine learning library) has stayed the same. We assume this has happened because scikit is old compared to other libraries and most of the questions related to it already exists which is why new posts need not be created (**a user can refer to older post instead of creating a new one**) . This is supported by the visualization below.

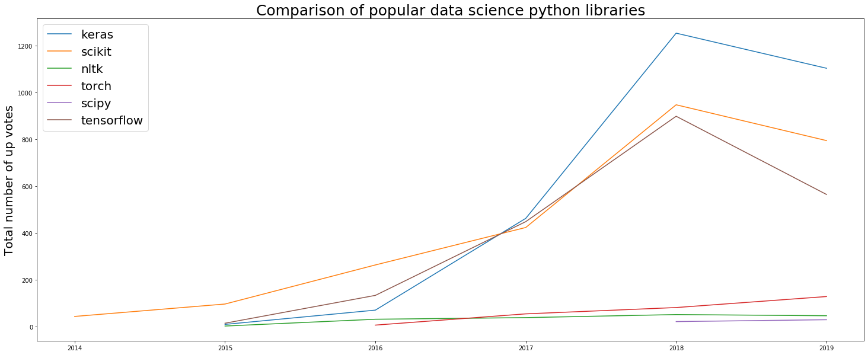
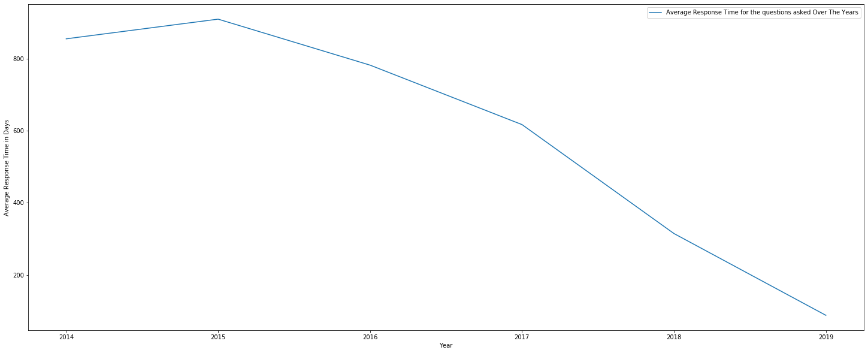


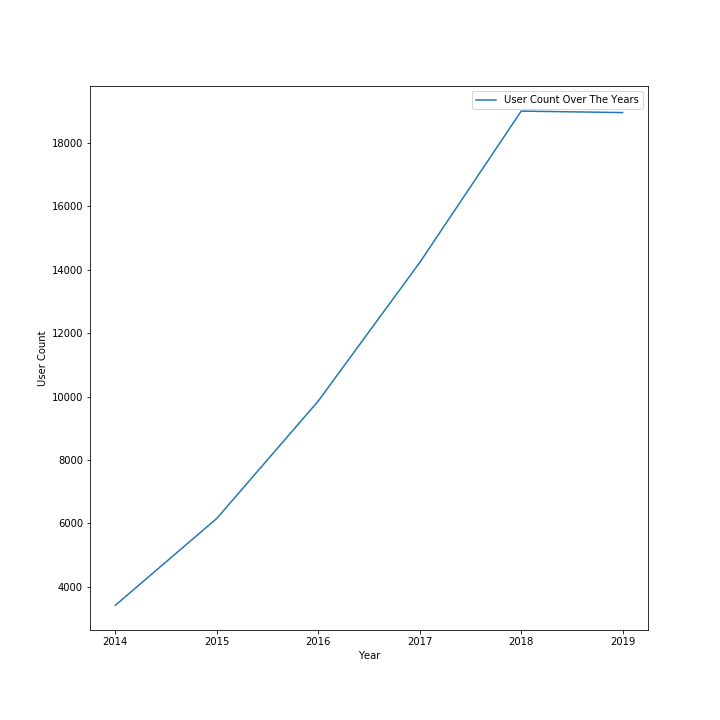
Fig: Number of up votes

If the dataset had non-aggregated value for views for each post, we would have concretely proved out the aforementioned hypothesis, but we don’t have it so we try to explain the hypothesis by counting the number of upvotes for a post. Even though newer framework’s post are created more in number during 2018 and 2019 (tensorflow vs scikit), upvotes for post related to scikit are relatively higher compared to number of posts created, which means people are still using stackexchange for scikit discussion but instead of creating new posts, they are referring to the older post.

1. **Average Response Time over the years**

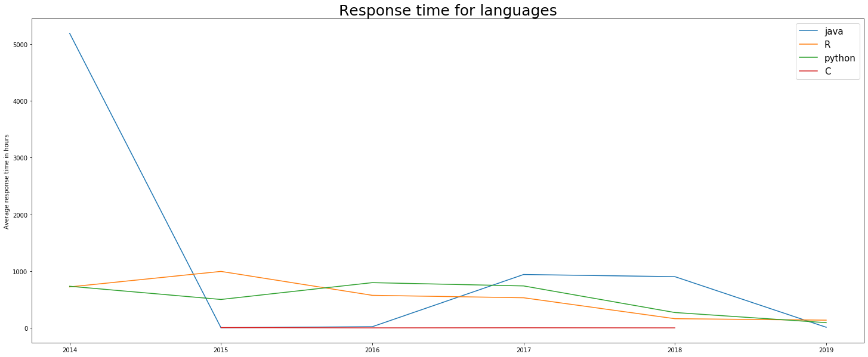
****Average response time has reduced over the years. To calculate this, we basically averaged the the hour difference between the creation time of post and creation time of the first answer. The reason that response time has decreased over time is because the number of users has increased due to which it is expected a latest post receives answer faster than previous years.

To support this, the **graph below** shows the number of users with respect to years. Number of users have **increased significantly.** With more users on the website, chances of a question getting answered increases.

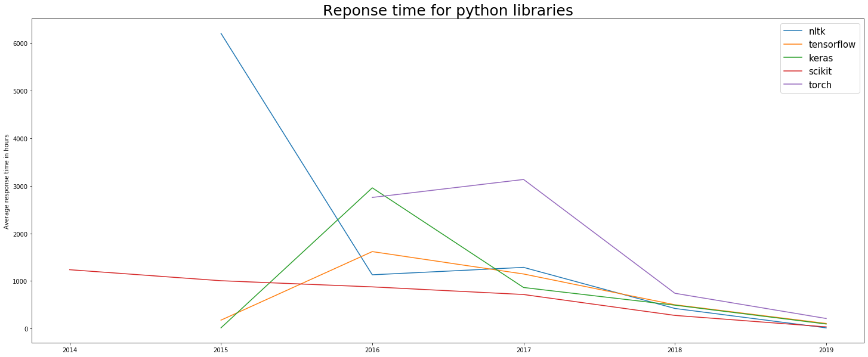


**Average Response Time for different languages and libraries**

Question: What kind of post has faster response and how the response time changed over the years?



This graph shows the average **response time for questions about different languages.** As the forum is about data science, so the question related to it R and python have better response time. Response time for **R reduced over the years, as people gained knowledge, smoothly whereas for python it increased in 2016-2017** which can attributed to release of new features/ new data science libraries (tensorflow, keras) and also adaptation of python 3.+. It is obvious that java has high response time because java is not preferred language for data science so very few people tend to use it and hence, higher response time. During 2015-16, the response time is almost 0, which we believe is because of the skew in the data. **Hence, post related to python and R tend to have lower response time.**



To support the above argument, libraries **like keras(release date 27th March, 2015[1]), tensorflow and nltk(not new but became mainstream because of data science as a field becoming so mainstream)** which were new in the market(2015 and 2016) and people didn’t have knowledge about them so the response time for the libraries is high which attributes to increase in response time of **Python between 2015-2016 which further proves the increase in average views between 2015-2016.** With time, people gained knowledge about these libraries and **response time decreased.** Similarly, **pytorch** was released in October 2016 which proves the **high response time**[3]. **Hence, old and stable libraries which most of the people have gained knowledge tend to have lower response time.**

**Data Model and Design Choice: Conjunction of Trie and Relational model**

We needed tags for comparison but tags column of post table had multiple values, and performing operations on such column is inefficient hence, we decided to discard the tags column and transform it. One choice was to create a new table to store the post and tag mapping and later on join on id columns. Other option would be to create a trie with mapping tagName -> list of postIds. As per our analysis, it is better to use trie instead of performing join on tables.

df\_t = df\_post[df\_post.index.isin(trie['tensorflow'])]

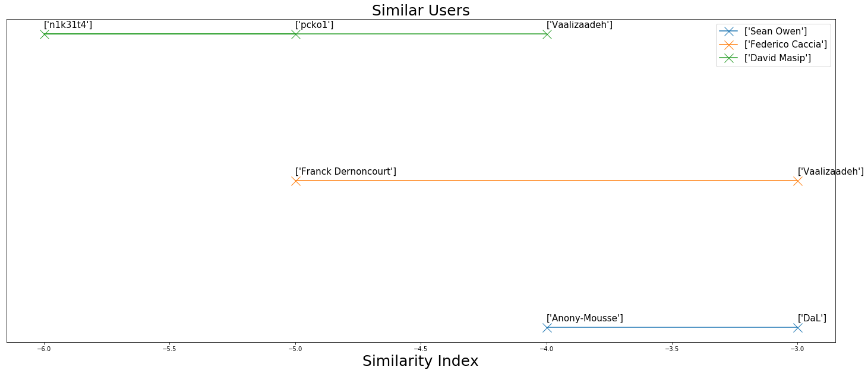
**Execution time: 0.00786 seconds**

select count(\*) from tag\_map, post where tag\_map.tagId = 'tensorflow' and tag\_map.Id = post.Id;

**Execution time: 0.014 seconds**

With the use of trie, we were able to get 2x improvement on speed of execution. **The reason trie is better than indexed search on relational table is that the depth of B-tree is proportional to size of dataset whereas depth of trie is proportional to length of key.**

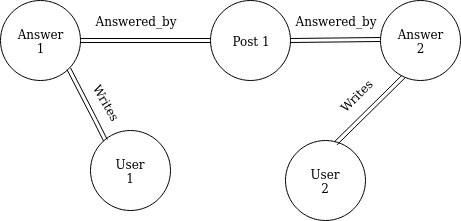
1. **Similar Users**



Popular social media features such as suggested users is implemented as part of this visualization. We assumed that **similar users tend to answer the same post** so we basically counted the number of times 2 users commented on a post, and sorted the tuple of users based on the number of times they commented on the same post i.e similarity index. We stored the suggested users for each user in a dictionary where key is the user id and value is the sorted list of suggested users (sorted based on number of times, they have commented on the same post). In the above figure, we visualized the suggested users for 3 users (Sean Owen, Federico Caccia and David masip). Lower the value of similarity index higher the similarity with suggested users. Since, this is a one dimensional data, we visualized it as a line with markers. Y-axis value for a point do not represent anything whereas x-axis value represent the proximity with the queried user.

**Design choices and data models: (Graph Model)**

We had multiple options to query users who commented on the same post. One would be to perform self join on post table on postId and parentId columns. Other option would be to model the data as a knowledge graph and perform a depth first search. The structure of the graph is shown below -



Even though we user node is connected to answer node, we have userId property present in answer node as well. The reason for doing so is to denormalize the structure and reduce the number of edge traversal during specific scenario. **We did not model graph on top of relational database because it takes more joins which is ineffective compared to graph database like Neo4j.**

**Relational Query (Self Join) :** select A.OwnerUserId, B.OwnerUserId, count(\*) as c from post A, post B where A.PostTypeId = 2 and B.PostTypeId = 2 and A.OwnerUserId != B.OwnerUserId and A.Id != B.Id and A.ParentId = B.ParentId GROUP BY A.OwnerUserId, B.OwnerUserId order by c DESC; (Index on PostTypeId, Id and parentId)

**Execution time: 0.238 sec**

**Graph Query:** MATCH (r1:Answer)-[:answered\_by]-(p:Post)-[:answered\_by]-(r2:Answer) WHERE r2.Id <> r1.Id and r2.OwnerUserId <> r1.OwnerUserId

RETURN r1.OwnerUserId, r2.OwnerUserId, COUNT(\*) as c order by c DESC;

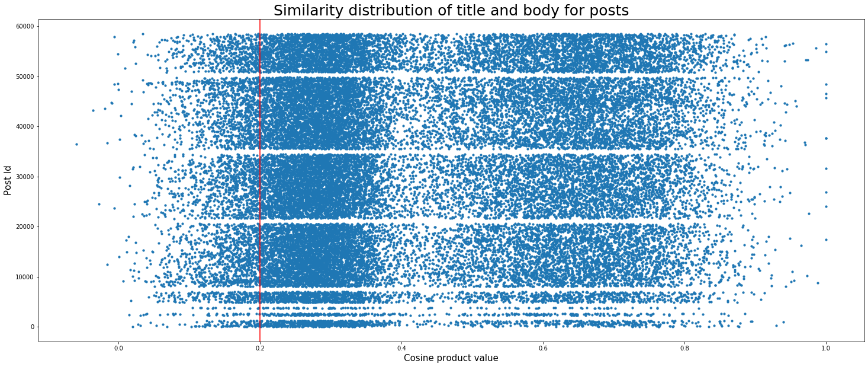
**Execution time: 0.149 sec**

The queries which intends to find common entity related to a same entity is best suited for graph model. Such queries in a relational model requires whole table scan and joins whereas in a graph model a single graph traversal is enough, which is why we were able to speed up the performance using graph model.

1. **Human Moderators on the website**

Websites like these require human moderators to keep an eye on the posts being made so that the posts are relevant (title and body matches or not) and are not **offtopic.** Hiring and managing the mods can be a tedious task. Also, new mods take time to get used to the rules. Here, we used a pre-trained **word2vec model to calculate similarity of title and body to determine if human moderator is required for a particular post or not.**

In the graph below we tried to predict how well the title of the post is well written or not and if a human moderator is required to intervene. Predictions are based on **cosine similarity** which calculated for the **titles of the posts**.If cosine similarity is less than 0.2 we tag that post as off topic.This filters out and reduces the number of posts for the moderators to check as **offtopic.** As clearly seen the density of posts **not offtopic is high** which accounts for the accuracy of the model.



We calculated the vector for title and body. We added each word’s vector of title/body to calculate the representative vector for title/body. After getting the title/body vector, we calculated the cosine product of title and body vector to find relevance between them. **If the value of similarity index is less than 0.2, a moderator has to be notified.**

**Design choices and data models: (Word2Vec)**

We did not train our own model, instead we used google’s trained model. The reason being that our dataset was not dense so training our own model would not have been accurate. Also, we are concerned more about similarity of title’s text and body’s text rather than biases, so we believe it is better to use google’s pre-trained model.

## **ADHERENCE TO RUBRIC**

1. We have created a story around trend of post for python’s popular libraries and response time of post related to different topics. We have demonstrated a way to calculate similar users for a given user. Also, we have explained a mechanism if a moderator of stackexchange is to be notified about relevance and quality of post.
2. As part of making the queries performant for an insight, we analysed different models and selected the one which suits the best. We have used graph model for finding users who posted on same post ( suits better than relational model which requires joins ). We used word2vec to find similarity between title and body to find the quality and relevance of a post. We used relational model to generate temporal insights by performing group by operations and adding indexes.Also, we used trie in conjunction to relational model to make queries performant.
3. Trie in conjunction with pandas/sqlite has been used to answer the question like ‘What kind of post have low response time and how the trend of libraries changed in stackexchange’. Graph model along with a dictionary has been used to find the similarity of users. Word2vec has been used to find the quality and relevance of posts and determine if intervention of moderator is required for a given post.
4. Several ideas taken from research papers relevant to topic introduced in class are -
5. As per paper [4], we used the cosine similarity (dot product) of representative vectors to calculate similarity between the title and body of the post.
6. The paper “**Nanosecond Indexing of Graph Data With Hash Maps and VLists**” has demonstrated the usefulness of indexes on graph data to speed up the performance. We have also added indexes on nodes (Id) to speedup the performance of queries.
7. Trie and relational database has been used in conjunction to speed up the queries.

**References**

[1] <https://en.wikipedia.org/wiki/Keras>

[2] <https://stackify.com/java-2018/>

[3] <https://en.wikipedia.org/wiki/PyTorch>

[4] C. Xia, T. He, W. Li, Z. Qin and Z. Zou, "Similarity Analysis of Law Documents Based on Word2vec," *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion*