**GIF DATA CHALLENGE – Swathi Annamalai**

1. **"Warmup" metrics:**

**Compute the aggregate "share rate": Ratio of shares to searches**

SELECT DISTINCT count(eventname)

FROM ios\_events

where eventname like '%share%'

SELECT DISTINCT count(eventname)

FROM ios\_events

where eventname like '%search%'

Share Rate = No. of total shares/ no. of total searches \* 100

= 22512088/34262927 \* 100

**Share Rate = 65.7039%**

**Compute the share rate for the top 1000 most-searched terms**

SELECT DISTINCT count(eventname)

FROM ios\_events

where tags IN (select TOP 1000 tags

from ios\_events

where tags <> '' and eventname like '%share%'

GROUP BY tags

order by count(tags) desc

)

***Result: 27005617***

SELECT DISTINCT count(eventname)

FROM ios\_events

where tags IN (select TOP 1000 tags

from ios\_events

where tags <> '' and eventname like '%search%'

GROUP BY tags

order by count(tags) desc

)

***Result: 27022603***

**SHARE RATE: 27005617/27022603 = 0.9994 \* 100 = 99.94%**

**How many unique search terms are there? What does the distribution look like (consider the count per search term; plot or describe the results)?**

**Unique search terms in the table:**

SELECT COUNT(DISTINCT (tags) ) as search\_unique\_count

FROM ios\_events

WHERE tags <> ''

**RESULT: 2729142**

**Distribution of unique search terms in the** **table**: Please refer to code - **Visualizations\_GIFChallenge.py**

**Explanation:** The distribution resembles a PROBABILITY DENSITY FUNCTION (PDF) graph. The histogram plotted shows properties of underlying data which is plotted on “tags” and their counts.

frequency distribution.html, Basic-bar.html, Searches-Count.html 🡪 are the Plotly graphs for this data

**Frequency distribution.html** 🡪 It represents a skewed distribution when sorted by most used search tags with decreasing order of usage. You can hover your mouse over the graph – you can see the count vs tags and specifically on different points of the graph, you can look at the search tag and their respective count.

**Basic-bar.html** 🡪 gives a histogram view of the same data, this time the data is not sorted in any order unlike the previous graph. You can see the PDF distribution underlying in this dataset too. Spikes are seen in this graph which indicates higher usage of certain tags. Excited, Goodmorning, Wow, Yes – many of such positive terms seem to be used more frequently which are indicated by spikes in the graph.

**Searches-Count.html** 🡪 It gives a more spread out and clearer view of the graph above – it can be considered as a sub-plot of the graph above since it accommodates 100 tags. If there are certain terms or certain category of terms we want to look at in order to measure sentiment of users, we can create subplots in Plotly using above approach to study particular category of GIFs. We can also expand to using bubble plots on categories which can denote number of terms under each category and bubble size will indicate number of terms within the category. Bigger the bubble, more number of tags under that category and vice versa.

**Unique GIFs in the table**

SELECT COUNT(DISTINCT (riffid)) as gif\_unique\_count

FROM ios\_events

WHERE eventname = 'share'

**RESULT: 481394**

**Code:** Refer to **Visualizations\_TenorChallenge.py**

A similar PDF distribution can be seen from plotting unique GIFs from this dataset. A few plots will appear upon running the Python code. Also refer to basic-bar-gif.html plot where GIFs that have a usage between 50 to 100 (less frequently used GIFs) and riffid is plotted on X-Axis; count on Y-axis.

1. **Advanced Metrics**

**Code: Please refer to R code – SessionRecords\_GIFChallenge.R**

**Concept:**

The session sequence number starts at 1 for each visitor and is incremented whenever the time interval is

greater than 10seconds This is done as follows:

1. Compute the time lag, in mins, from prior record
2. Set session flag as True or False when: True = there is no prior record for visitor OR False = lag to prior record is > 10secs/ 5 seconds
3. Perform Cumulative sum of session flags to get session sequence number
4. **Experimental design (hint: A/B testing):**

- Suppose you've built a new model for the search engine that you'd like to test against the current model.

\* How would you measure the effectiveness of the new model?

\* What additional data would you need in the dataset to compare the effectiveness of the new vs. current models?

\* How would you determine the statistical significance of the results?

**Measure effectiveness of the new model -**

**User Testing:**

I tested on a few terms in the Tenor website (current) and particularly tested on one term "indian" which gave additional search suggestions such as "indian man", etc. Then used "indian man" as the search term and noticed that the retrieved GIF's having keywords such as "indiana"/ "Indiana Jones" which is not accurate to "indian man". Also saw the search return GIF's with some "bollywood heroines" which again doesn't conform to the search term "indian man".

If the search was "indian" - it can display Indian man or woman, but the search for "indian man" should be more accurate.

On the other hand, upon searching "indiana" or "indiana jones", the search results were more accurate.

Using a "User Testing Group" to search and regression test out the search results, an example as shown above, can be one of the ways to measure the effectiveness of the new model - comparing results returned from old model to the new model.

**Applying A/B Testing to measure effectiveness of new model:**

In A/B testing we look to improve conversion rate - measuring the number of users converting to lead users (i.e., sharing more GIFs rather than just searching for GIFs).

Instead of creating 2 separate versions of the same page and understanding user clicks, then calculate conversion rate for new model v/s old model, we could provide for the following:

1. Based on commonly used keywords such as "Friday" (is most searched for on Fridays) or “Happy”, identify such pages which are popular and these are the pages we would want higher conversion rate. Identify and setup the new model version and layout for a set of popular keywords or phrases.
2. Choose test to run across the pages - Tests will include capturing "clicks", "shares", "searches" and calculate % shares/ % searches as a part of these tests. Capturing these metrics can be achieved by exporting to a dashboard where we measure the % of searches v/s % of shares. These metrics should be run for both old and new search models in order to understand which model works better. If A variant of the model has higher click through or share %, then it's better to stick to older model than the newer model. It just proves the customers are more comfortable with the A version of website compared to B variant.
3. Conversion funnel - This describes journey of a customer through stages of search, clicks and a final share. Click-through rate represents top level of the funnel. This measure is affected by layout in website, search results returned, improved suggestions based on typing within search bar. Highest click-through rate when achieved leads to ad optimization. That is when ads are hidden within GIF's, the rate at which GIFs are shared helps us measure the ad-optimization parameter.
4. Measure these changes and declare success if variant group (B) outperforms its forecast while original group does not.
5. We can look to modify certain "tags" and compare to see the %share on those particular tags between the old and new model.

**Additional data needed to compare the models:**

We can look to have a "feedback" section someplace for the product where we can record feedback from user base. Based on keywords from this "feedback" field, we can look to join this keyword to match up to existing keywords and find out missing emoticons. This could be helpful in comparing what's missing in new model compared to the old search model and hence improvise the development of new GIFs based on customer needs.

**Statistical significance of results:**

1. Causal Analysis - We can run CausalAnalysis package using R. We can perform SEO eventname by date. In case we are updating "tags" in the new search model and the tags got updated in the 100th day, I want to examine user behavior in next 35days. I can plot and report the results from using this function. The plot will display 3 stacked graphs - original, pointwise and cumulative. Original graph is data points from original dataset. Pointwise is estimated impact on a day to day basis after making the tag change. Cumulative shows overall impact of making the change - it would display an increasing or decreasing trend.
2. Statistical Hypothesis Testing - We make a hypothesis saying that Variation B will perform better than original page A, then metrics from both pages are captured and compared to determine if Variation B is statistically significant improvement over the Original Page A.
3. Variance - we might have 2 out of 100 conversions compared to having 1 out of 10 conversions in a sample. Hence, variance measures how far a random sample can differ from true mean.
4. Confidence Interval - Original model may have conversion rate of 20.3% +/- 1.0%. The conversion rate is the mean and thus confidence interval spans between 19.3% to 21.3%. The 1% is a margin for error. 20.3% +/1 1.0% at 95% confidence is our actual rate of conversion. We can use this statistic to measure performance of new search model.

1. **Predictive models:**

There is sufficient data in the dataset to build a crude "search suggestions" model: Given a specific search term, return a list of related search terms. Describe how you might build such a model (describe the data you would use, techniques, inferences, etc.; you do not need to implement this).

In order to learn the user behavior and user searches, we can build "customer intent segments" - what is the intention of the customer that’s making him search for those specific GIFs on your website. We can do this by using "**K-means clustering**" unsupervised learning to create this model.

K-means clustering is a method of partitioning data into "k" subsets, where each element is assigned to closest cluster based on distance of that element from centroid of the cluster.

1. We first use "RSiteCatalyst" to get the list of distinct search keywords "tags" from our dataset into a dataframe.
2. To apply k-means clustering to TEXT data, we first need to convert text to numeric data using document-term matrix. This can be achieved in R using RTextTools package. In document-term matrix, each row is a "tag" and each column is 1/0 bit representation showing the tag present in natural search term
3. Find popular words - Initially, in order to get a view and apply this method on a small subset of data, we can use findFreqTerms to find minimum frequency terms in the dataset setting it to freq = 10/ 50 - the number of times a tag occurs in the dataset.
4. Determine number of clusters -There are 2729142 unique tags in the dataset - it would be difficult to pick a number for "k" clusters. Rather we can use the "elbow method" for such a dataset which uses automated approach for picking k. For every kmeans object returned, there is a tot.withinss that provides the total of the squared distance metric for each cluster.

*#accumulator for cost results*cost\_df **<-** data.frame()

*#run kmeans for all clusters up to 100***for**(i **in** 1**:**100){  
 *#Run kmeans for each level of i, allowing up to 100 iterations for convergence* kmeans**<-** kmeans(x**=**dtm, centers**=**i, iter.max**=**100)

*#Combine cluster number and cost together, write to df* cost\_df**<-** rbind(cost\_df, cbind(i, kmeans**$**tot.withinss))

}  
**names**(cost\_df) **<- c**("cluster", "cost")

The cost\_df dataframe accumulates results from each run, which is then plotted using ggplot2. This plot will help us understand breakpoints in our cost and helps us decide on where to stop adding clusters. At certain points in the graph, you will see the graph start to flatten out which indicates stopping point for adding any more clusters. Adding more clusters only makes the entire technique ineffective and inaccurate.

Clustering here serves to discover underlying patterns and divide search tags into separate groups. Hence, following clustering we can apply Classification in order to take these features and predict classes. For Classification, we look to map keywords to users’ needs and state of mind. Here, we can apply Naïve Bayes and Support Vector Machine classifiers to return list of search terms given a search tag.

Naïve Bayes - We need to classify search terms into different labels. Pythons scikit-learn package can be used for this purpose. In order to train this multinomial Naïve Bayes classifier, I first need to get training data containing tags (80% of the original dataset) and labels. Labels are states where keywords would be classified. In this dataset, "category" will be the labels. We have 25 distinct categories such as emoji, audio, uploads, packs, profiles etc.

On running the classifier through this training dataset and labels, we can measure Accuracy and Confusion Matrix in order to test the effectiveness of this model. Following this, we can run the rest of the 20% of the dataset which is the validation dataset and compute accuracy and Confusion Matrix to confirm validity of this classifier on this dataset. We can also use Precision-Recall here.

Similarly we can apply SVM using Python sckit-learn - SVM is non-probabilistic classifier. It learns non-linear models.

Naive Bayes treats features as independent, whereas SVM looks at the interactions between them to a certain degree, as long as you’re using a non-linear kernel.

Confusion Matrix is the most appropriate metric for calculating validity of the classifier as it helps us understand the Actual vs Predicted - how many search terms were actually classified vs predicted.

Deciding on the best classifier for this dataset is done by looking at tradeoff between Bias and Variance. An ideal model should be able to capture regularities in training data and generalize well to unseen data. Would be hard to find both in a classifier, but we can look to achieve balance in the two.

Another technology we can put to use in order to solve this problem is using **Neo4j's GraphDB** in order to build a search model.

GraphDB helps us design a link based analysis and build patterns within data fed to the GraphDB. We can feed in different columns and column values in order to build a link analysis graph and study the data.

1. Neo4j provides a database model and language that supports this kind of a modeling. Upon feeding the dataset to Neo4j, it assigns a variety of rich metadata to content for rapid search and retrieval. We can change the underlying data and structure or add new data and re-run the model in order to obtain better search results and link results.
2. GraphDB operates on node-relationship model where node is annotated by name and type of concept it represents; Relationship is the path that define relation between two nodes connected. Traversal along the path (relationship) can be two-way traversal depending on how the relationship is coded.
3. Transforming this model to Tenors ios\_events dataset:
   1. Node 1 - USER: represents a unique user in the dataset. Node 2: KEYBOARDID.

Node 1 ---------- Node 2 (relationship here is USES) = User Uses KeyboardId

1. Node 1 - USER; NODE 3: Timestamp; relationship is Logged at --> User Logged at Timestamp
2. In this way, we can define a parent Node = Category --> which will have 25 child nodes as there are 25 distinct categories in this dataset.
3. Define another parent Node = Eventname which will have 2 child nodes = share and textsearch.
4. Following these nodes, we can then connect specific "tags" under category and eventname and see what other related tags are connected to the same category and eventname. We can pass in chunks of data with "tags" and "eventname+"category" data to this GraphDB - we can iterate over batches of data and the Graph will keep growing and find related search terms. Ex: Happy is associated with Smile; Cute is associated with Beautiful; sisters associated with blushing, etc.
5. We can break out graphs depending on category or even associate event/category/tag data to a set of users for a week’s time. We can apply querying at different levels in order to understand different associations hidden in the dataset.