Capstone Project

Tom Chang

Definition

Domain Background

The Knowledge Graph assists in a buyer's shopping journey to find their perfect on Ebay. A very popular use case is given a user query, the knowledge graph can recommend item aspects they may be interested in.

The purpose of this document is to explore how we can leverage the current Knowledge Graph data to further increase its usefulness. In particular, we want to graph to be able to "infer" knowledge where there are no current knowledge. One way is to build machine learning models to predict:

- Given the user's current query, what are some other "similar" queries the user may be interested in? This information may be used to recommend questions back to the user or enhance our current aspect recommendation by combining multiple queries to achieve the better aspect recommendation.

While word embeddings has been very popular in research and industries. One of the seminal paper is described here. Word embedding has been applied to vision applications as well (link). However, I am not aware of specific embedding application in the realm of knowledge graphs.

Problem Statement

The goal of the project is to create an word embeddings for queries such that similar queries will have high cosine similarity.

To accomplish this goal, we will apply word2vec algorithm on our queries. If the algorithm performs "well", queries that tend to appear near the same aspects will be represented in near proximity in the word embedding space, giving a low cosine similarity score. The specifics will be described in more details in algorithms and techniques section of the document.

Metrics

The best way to evaluate similarities between word embedding is to calculate the cosine similarity between like terms.

Word embeddings are notoriously hard to train because there is no singular metric one can use to “measure” the model. Part of this is because it is hard to deem what is similar? Is it via analogy, via relationships, etc… Furthermore, because we are dealing with human interpretation of words, reasonable people may come to different opinions. It may be more harmful to tune our model on a metric that may not be consistent or measure what we want.

Instead, I propose 2 ways of evaluating the performance of the word embeddings:

- Randomly select N words, and see the top M words closest by its cosine similarity.
- Visualize the embeddings via PCA visualization. While PCA visualization reduces the variance in the dataset, it is often use to get a good global perspective of the embedding space.

Analysis

Data Exploration

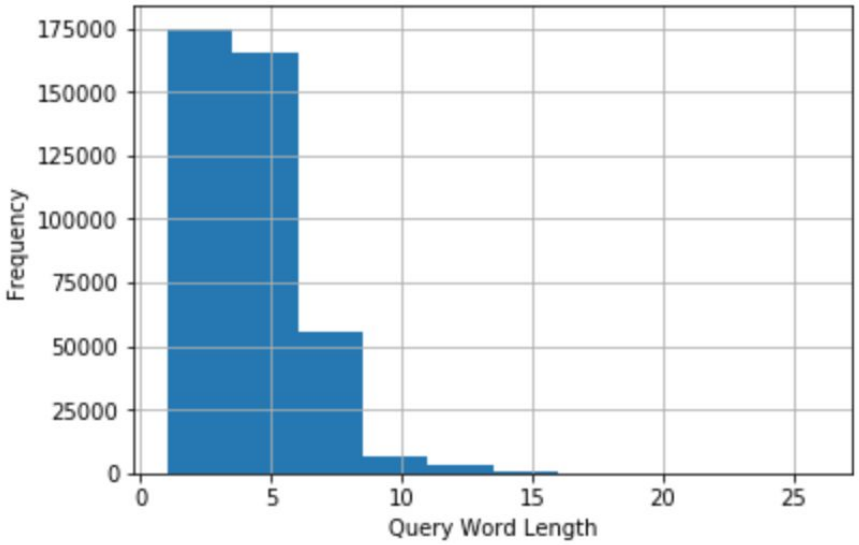
The data we use to training are collected via the user click logs. For a given user query, we know the aspects of “clicked” items.

To be precise, queries here are the longest fragment of the original search query. As an example, for the query “I want to look for red nike shoes”, the longest fragment will be red nike shoes. The dominant object will be will be nike shoes. One can loosely associate the dominant object as the main item the user is buying. There are currently **55,067,901** unique queries.

Similarly, an aspect is a tag value pair. In the nike shoe example, two aspects may be color:blue and size:9. We have **144,627** unique aspect values. I have also processed a list of all aspect names in the file kgAspectNames.txt attached in this project. There are **2,367** unique aspect values.

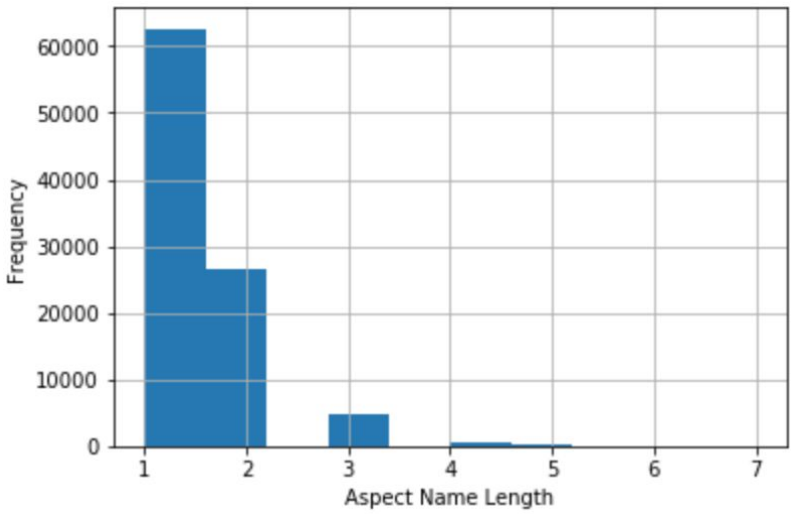
The query visualizations are listed below. The average length of the query is around 4. Interestingly, there is a query with length 26. I will be removing any query greater than 10 from the training data set. They are removed because they do not reflect normal typical user queries (aka dominant objects).

	count
count	405787.000000
mean	4.031899
std	1.799874
min	1.000000
25%	3.000000
50%	4.000000
75%	5.000000
max	26.000000



The visualization for aspects are below. The average aspect length is 1.45, which makes sense because the tag names corresponds aspect categories, such as brand, color, etc… There does not seem to be any clear outliers. The outlier data point with a length of 7 was actually valid.

	countTagName
count	94689.000000
mean	1.405443
std	0.633594
min	1.000000
25%	1.000000
50%	1.000000
75%	2.000000
max	7.000000



Algorithms & Techniques

To create the embeddings, we will utilize the word2vec algorithm. There are 2 flavors of this algorithm, Continuous Bag Of Word (CBOW) and Skip-Gram. In

skip-gram, the algorithm is given a target word, and tries to predict the neighboring words. In contrast, the CBOW is given the surrounding words, and tries to predict the word between the surrounding words. In either case, the objective function tries to minimize the negative likelihood of the sequence (description). The embedding is the last layer of the neural network.

One key challenge is to design the input to the word2Vec algorithm. The desire goal is for embedding of dominant objects that share clicked item aspects to have a large cosine similarity score. Furthermore, the fact the median query length is 3 may result in inferior embedding performance.

We can address these 2 concerns by leverage the insight that wordvec is a co-concurrence algorithm. Two words that appear in similar surrounding words will have a high cosine similarity. Given this, we can design our input as: Given a target query TQ, we insert the aspect between each token of TQ.

As an example:

Sugestion: Set##Alara Reborn maelstorm nexus Set##Alara Reborn

Sugestion: Padding to limit to n terms...

- query = maelstorm nexus
- aspects = {Color##Multicolored, Set##Alara Reborn, Type##Creature}
- Color##Multicolored maelstorm Color##Multicolored nexus Color##Multicolored
- Set##Alara Reborn maelstorm Set##Alara Reborn nexus Set##Alara Reborn
- Color##Multicolored maelstorm Color##Multicolored nexus Color##Multicolored

This model will be trained via TensorFlow.

The benchmark we compare out model with is based on the word2vec model, but the data is trained on English GoogleNews. The benchmark model is trained on a news-centric data, and it would be interesting to see how it performs on commerce queries.

Methodology

Data Preprocessing

The data preprocessing consists of creating the query aspect sentences.

To do this, we read in files where each line consist of user query and the click item aspects.

Given this, the pseudo code to create the sentence input is:

- For each line in the click log, containing query, item aspects
- Split the line into query and item aspects
- Split the query into singular tokens (aka ngram)
- For each query token, inject the aspect between the query token
- Skip line if query or aspect are empty

One of the challenge I had to overcome was the size of the input files. There were 85GB of text data. To overcome this, I leveraged Spark on the Google cloud clusters to create the training data for my word2vec algorithm.

```scala
/**
  * Method creates the training data for query similarity model.
  */
def extractQuerySimilarityEmbeddingTrainingData(conf: Config, spark: SparkSession): Unit = {
  val jobConfig = Util.getQuerySimilarityEmbeddingTrainingConfig(conf)

  val kgDataRawDF: org.apache.spark.sql.DataFrame = fromTSVDirToDF(
    spark,
    dir = jobConfig.inputDataDir,
    schema = KGQueryAspect.schema,
    bDropNullRows = true,
    numPartitions = jobConfig.numPartitions)
  kgDataRawDF.show(5)

  val queryAspectRDD = kgDataRawDF.select(kgDataRawDF("query"), kgDataRawDF("aspect")).rdd.distinct.map {
    row: Row => {
      val queryTokens: Seq[String] = row.get(0).asInstanceOf[String].split(" ")
      val aspect: String = row.get(1).toString.replaceAll("\\s", "_")

      aspect + " " + queryTokens.mkString(s" $aspect ") + " " + aspect
    }
  }
  val outputDir = s"${jobConfig.outputDir}${getNow()}/".replaceAll(" ", "_")
  queryAspectRDD.repartition(5).saveAsTextFile(outputDir)
}
```

For data cleaning purposes, I also drop any data which has null values.

Implementation

The training is done in tensorflow. The code is included in the project submission, and can be viewed in the file word2vec_query_similarity.py. The code was inspired by the Google Tensorflow documentation.

The pseudo code is:

- Build the vocabulary by read the query aspect training data we designed earlier.
- Generate the data for each mini batch of training. It is here we build the skip-gram model, whereby the window and num_skips are used.
- Define hyper parameters such as embedding size, skip_window, and size of each batch
- Start training model across multiple epochs
    - Pass the training data as place holders into each training iteration
    - Calculate the NCE loss and pass it to the GradientDescent optimizer
    - At each iteration, generate metaData for Tensorboard for training and debugging visualization

To improve the training models, I have experimented tuning:

- Hyper-parameters
    - Size of embeddings: started at 128, ended up at 256 as it gave relatively good performance without the model getting to big.
    - skip_window: corresponds to how many words to look to the right and left of the target word. I started at 1 and went up to 75% of my aspect tag (2). Eventually, I settled at 1.

- Different optimizers
    - I tried Tensorflow's AdamOptimizer, but it did not seem to improve the performance.

Results

Model Evaluation and Validation

The performance of the algorithm can be evaluated by looking at the cosine similarity of a term against other similar terms. The performance of the model, however, appears to be not very consistent.

Method 1: Top N Closest Candidates

As previously stated, I will randomly select N words from the embedding space, and see its closest neighbors, and determine if we capture some semantics.

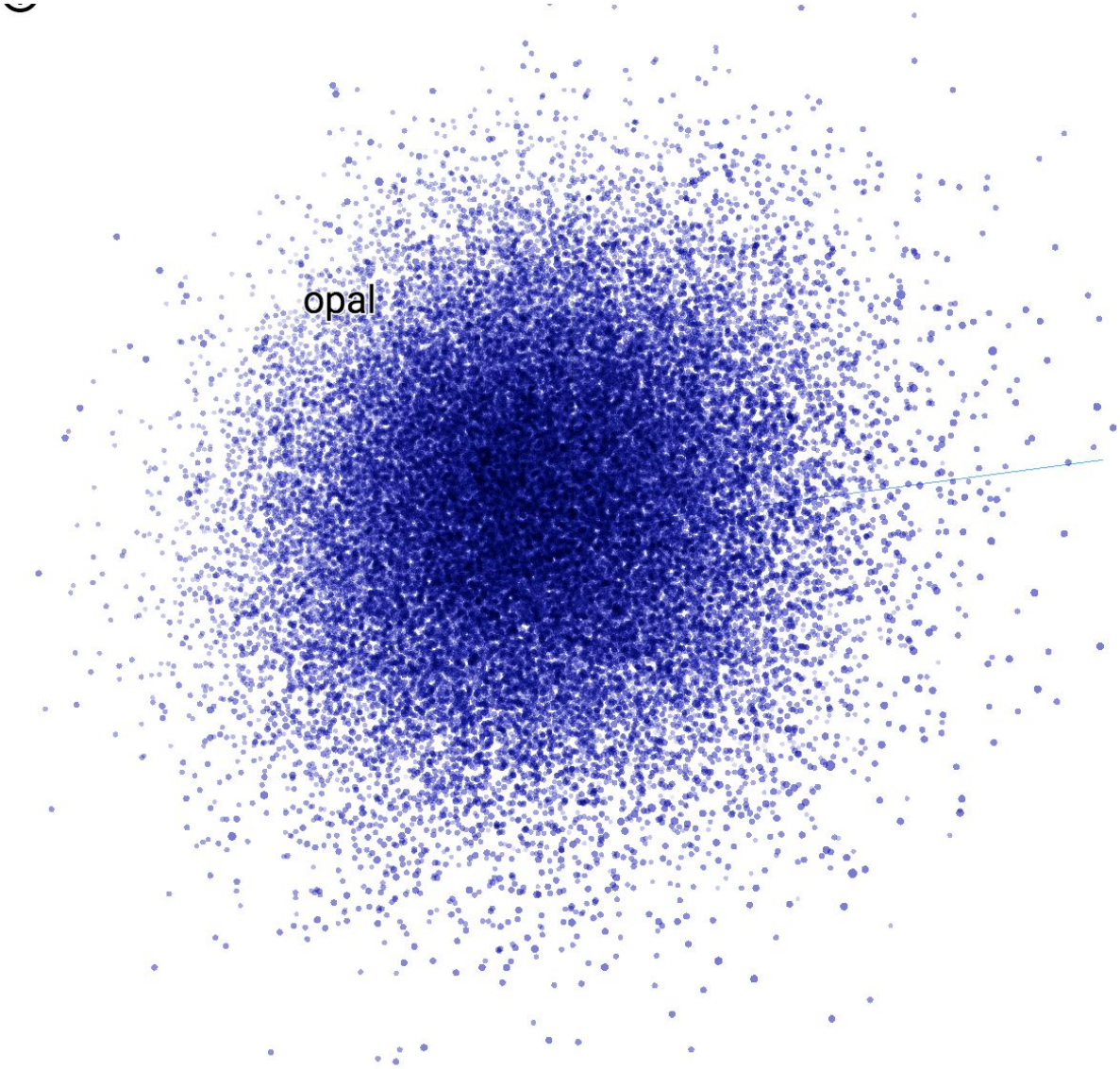
Nearest to Genre: Release_Year, Special_Attributes, Region_Code, Format, Rating, Sub-Genre, Record_Grading, Artist,
Nearest to Vehicle_Mileage: Model_Year, Brand_Type, Make, Fitment_Type, Warranty, Placement_on_Vehicle, Engine_Size, Transmission,
Nearest to nike: mens, size, womens, Nominal_Outside_Diameter_(OD), Ship_Type, Treatment_for, shirt, Button_Shape,
Nearest to Brand_Type: Fitment_Type, Placement_on_Vehicle, Warranty, Genuine_OEM, "Lift_Height, Bulb_Type, Lens_Color, Vehicle_Mileage,
Nearest to Character_Family: Shop_For, Character, Age_Level, Packaging, Year, Scale, Era, Material,
Nearest to Case_Size: Face_Color, Case_Color, Band_Color, Band_Material, Case_Material, Movement, Case_Finish, Display,
Nearest to dress: shirt, size, Wavelength, Nominal_Outside_Diameter_(OD), womens, Threat_Level_Rating, Button_Shape, Manifold_Type,
Nearest to Type: Bundle_Listing, Brand, Color, Country/Region_of_Manufacture, Material, Features, Warranty, Featured_Refinements,
Nearest to Band_Color: Band_Material, Case_Color, Face_Color, Case_Size, Case_Material, Movement, Case_Finish, Display,
Nearest to Brand: Color, Country/Region_of_Manufacture, Bundle_Listing, Type, Material, Gender, Features, Age_Group,
Nearest to Size_(Women's): Sleeve_Style, Size_Type, Pattern, Occasion, Sleeve_Length, Neckline,

Style, Bottoms_Size_(Women's),
Nearest to black: UNK, Wavelength, Nominal_Outside_Diameter_(OD), Treatment_for, End_A_Diameter/Tube_OD, Ship_Type, seller, white,
Nearest to womens: size, women, Ship_Type, Button_Shape, black, mens, Wavelength, Nominal_Outside_Diameter_(OD),
Nearest to watch: mens, Nominal_Outside_Diameter_(OD), Valve_Material, Button_Shape, Supply, Ship_Type, Audio_Outputs, Memory_(RAM)_Capacity,
Nearest to Publication_Year: Subject, Special_Attributes, Topic, Format, Language, Genre, Sub-Genre, Release_Year,
Nearest to 2: UNK, 3, 4, black, new, Wavelength, for, End_A_Diameter/Tube_OD,

Here, the first word correspond our random selection of words to evaluate. The color of the cell correspond to how well the randomly selected word is semantically similar to its nearest neighbor. Green means it captures well, yellow means some of the closest words matches semantically, and red means the embedding does not perform well.

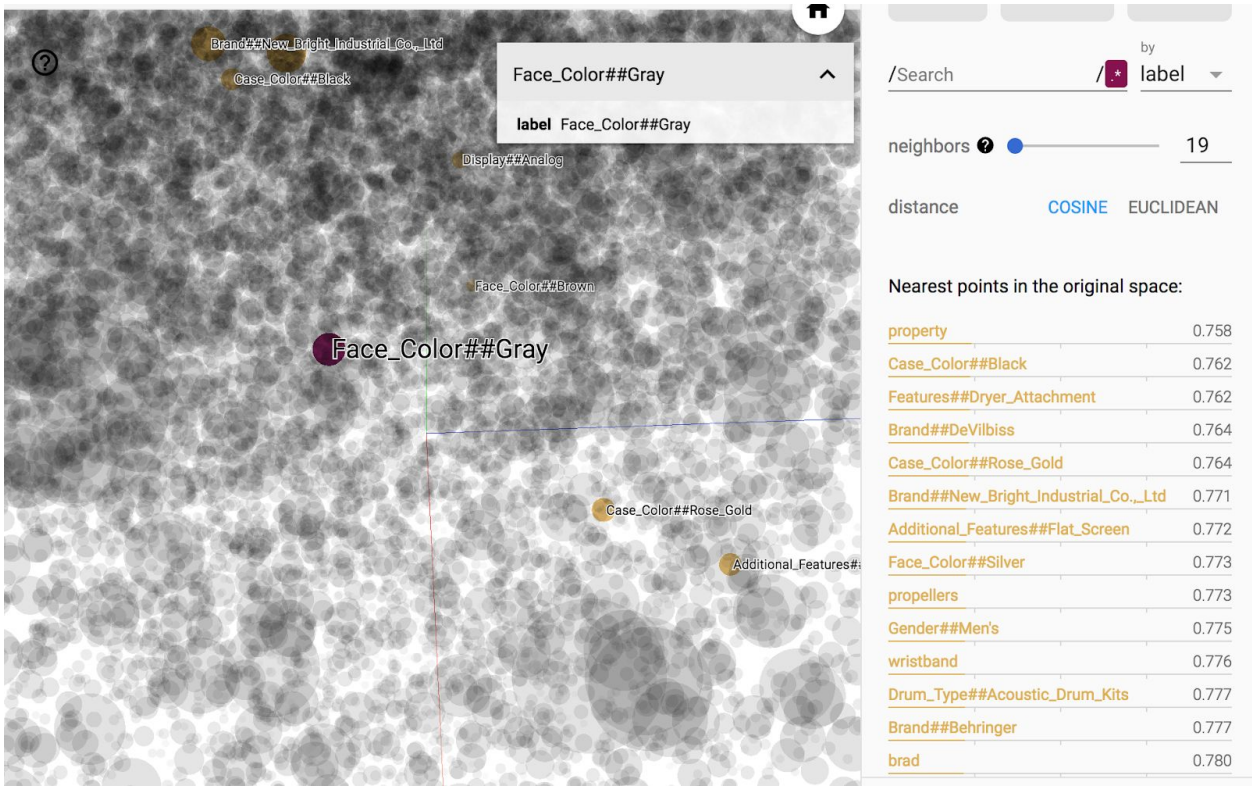
Method 2: Via PCA Visualization

I will be using the Tensorboard projector feature to see the embeddings in 3D space. It looks likes below. Each blue point corresponds to a word embedding of either an word or aspect in our training data. The plot uses PCA which chooses the axis to maximize the variance in the data.



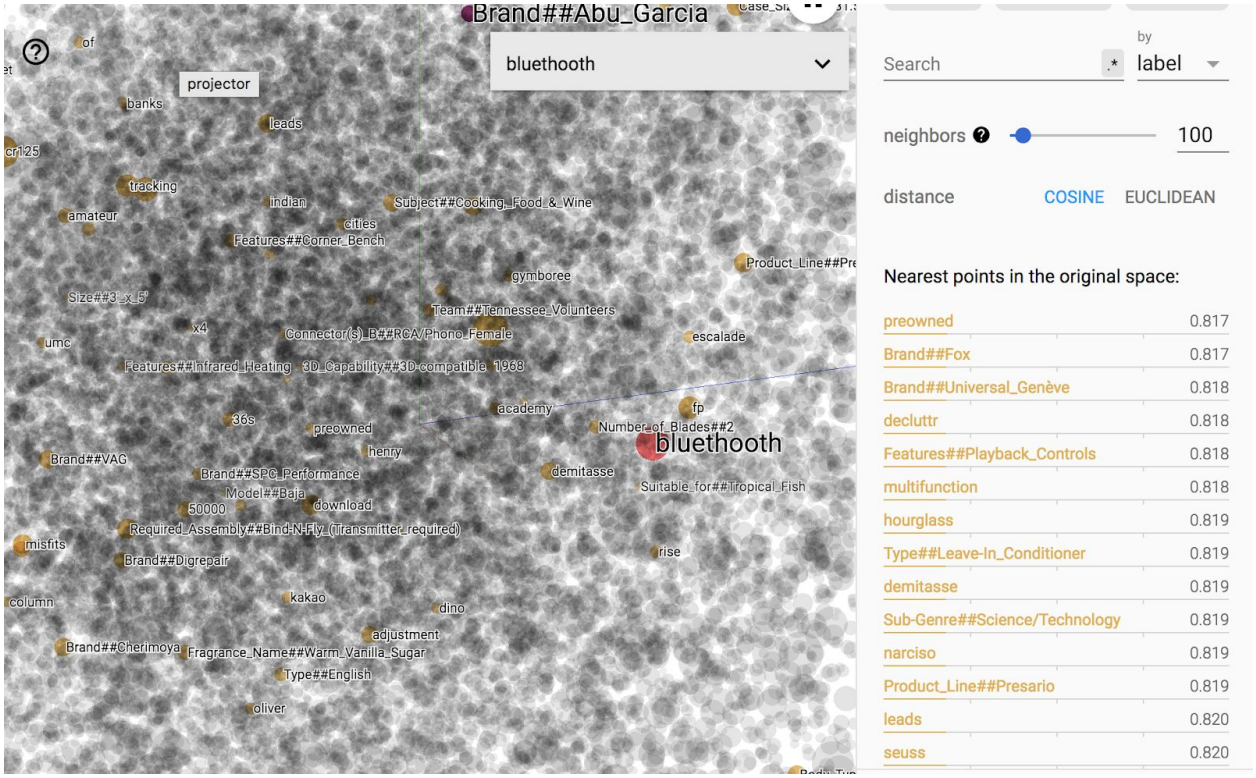
Good Example: color##gray

In the case of color###gray, the top 3 neighbors are property, case_color###black, and Feature###Dryer_attachment. Intuitively, these embeddings does have real life association with the color##gray. In fact, if one looks across the list in the diagram below, most of the terms do make sense.



Bad Example:bluetooth

Unfortunately, the performance for bluetooth is not very promising. The top categories based on cosine similarity hardly appear to be related to bluetooth.



Comparison to Benchmark:

Earlier, I mentioned that the benchmark would be a word2vec model trained on the English Google News. We will use the same terms that were randomly chosen queries in the previous section. The model can be accessed here.

nearest_to_genre	None
nearest_to_vehicle_mileage	None
nike	NIKE, Nike, usa, india, american
brand_type	None
character_family	None
case_size	None
dress	dresses, frock, attire, gown, cocktail dress
type	kind, types, sort, particular, characteristics
band_color	None
size_women	None

black	white, blacks, brown, blue, whites
womens	mens, women, Womens
mens	womens, mens, men, ladies, women
publication_year	None
2	3,4,1,5,6

Here, we can make a couple of observations:

- Even though the benchmark model returns multi-term queries, most of the multi term queries do not return any results. This may be because these queries were never seen in the Google News.
- For the queries that returned results, the results makes sense. However, it is not clear whether it performans well for my goal, which is given a query, what are the aspects that are interesting. Here, the benchmark model does not help in that goal.

Conclusion

While the project has been very educational, the performance for the word embedding to capture similar query aspects has been mixed. The performance of the embedding appear to depend on the word chosen.

The project has been educational because it helped reinforced the ML methodologies we have practiced throughout the course: data exploration, visualization, model tuning, and evaluation trade offs. This project has also pushed me to learn new technical tool sets and algorithms. I created a Spark ETL pipeline to the prepare the large volume of data needed for the training data. I explored and learned about Tensorflow in greater depth, and understanding the graph nature of its implementations. And finally, I worked about word embeddings. Word embedding can capture semantic meanings, but it can also pose challenge as shown from our analysis.

There are some hypothesis as to how to improve future performances. Perhaps the aspects are not discriminative enough. In other words, if an aspect appears in 90% of the queries, it's discriminative power may be compromised. Given this, I may want to clean the training data to include tags that are truly discriminative.

To conclude, this project has been very educational and the result shows some promise.