Advertising Brain for Programmatic strategies

Project Overview:

The world of online advertising has witnessed tectonic shifts in the last decade. The evolution of digital and mobile advertising has disrupted the traditional advertising space confined to print and television and has introduced new execution strategies to target Audiences that are driven by personalization and are more relevant compared to traditional "spray and pray" techniques used for mass targeting.

Programmatic media buying, marketing and advertising is the algorithmic purchase and sale of advertising space in real time. During this process, software and data driven approaches are used to automate the buying, placement, and optimisation of media inventory via a bidding system.

Programmatic play allows the "owner/brand" to tailor a specific message and creative to the right person, at the right time in the right context – using audience insight from the brand (the customers you want to target) around the kind of audience they want to target.

Underlying all the disruption, in Advertising Industry, is the flood of data-- big data-- and superior data processing techniques, *Machine Learning and Deep Learning*, that help organize data into audience segments.

NLP and text processing plays a pivotal role mining the data and automating the datapipe that consists of heterogeneous, structured & unstructured, data sources -- like location data, Point of sales (POS) data, online transaction data, social media data, etc.

The focus of the project would be mine social media data, a rich source of behavioural data, to create programmatic targeting strategies that can be used by the media buyer to bid/buy digital media

Programmatic Targeting Strategies:

Programmatic buyers/traders use following strategies buy/bid for digital media.

- Segment based targeting.
    - *Segmenting a broad target **market** into subsets of consumers who have common interests and designing strategies to target them.*

- Keyword targeting
    - *Bidding for Keywords that resonates with particular demography, gender or segment of Audience.*

- Retargeting
    - *Target consumers based on their previous Internet actions to help companies reach target audiences who don't convert right away*

Data Classification & IAB Segments:

The Interactive Advertising Bureau (IAB) has developed and standardized Data Segments & Techniques Lexicon that provides a framework to help understand how all of the data components work together to form the critical audience segments that enhance advertising value.

The first tier is a broad level category defined as a targeting depth of either: category/portal, site section, or page. Tier 2 categories and greater are additional categories nested under Tier 1 categories. Both tier 1 and tier 2 categories are formerly established for the IQG Program so that content classification can be consistent across the industry.

The links provide more details on IAB, Data sources and data classification.

- https://www.iab.com/guidelines/iab-quality-assurance-guidelines-qag-taxonomy/
- https://www.iab.com/news/data-segments-techniques-a-new-lexicon/
- https://www.iab.com/guidelines/social-data-demystification-best-practice-2/

Problem statement:

Behavioural data, *though difficult to mine*, promises rich "information gain" if mined correctly. Social media is a rich source of "Behavioural data".

Can we use Twitter data stream, *Twitter is public & free*, to mine behavioural traits and classify conversations into various IAB segments to create media buying strategies.

The results of behavioral mining should predict

- Trending level 1 & level 2 IAB segments that a marketer/media buyer can bid.
- Trending Keywords within each segments that can be used by media buyer to bid for right keywords.
- Correlation amongst different IAB segments -- so that media buyer can target audiences in different segments a user is likely to visit in his internet journey.

Metrics

We measure the ***accuracy*** and ***mean cross-entropy*** loss after stipulated number of steps and ensure the loss tends/converges to zero as we increase the number of steps.

We used ***mean cross-entropy*** loss as the ln() function in cross-entropy takes into account the closeness of a prediction and is a more granular way to compute error.

II. ANALYSIS:

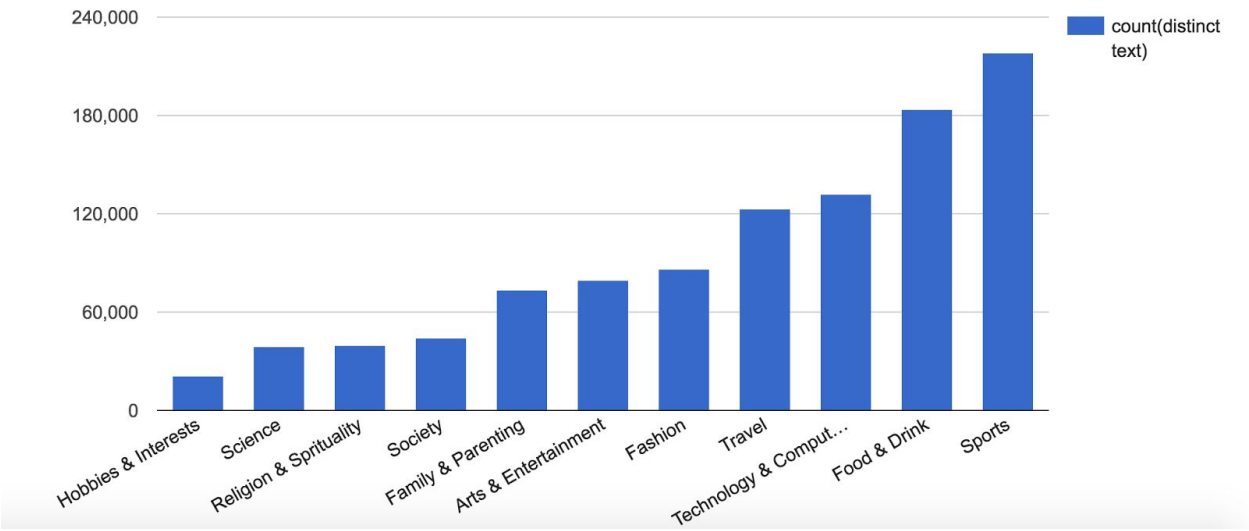
Data Exploration:

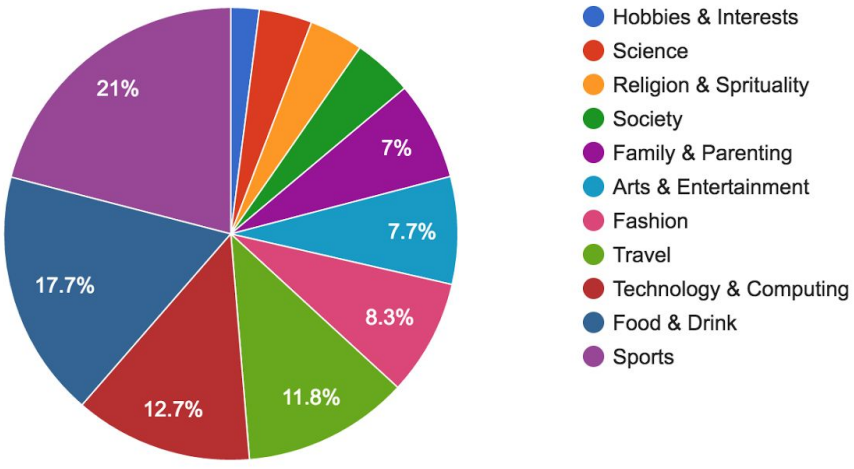
Twitter provides REST APIs you can use to interact with their service and fetch real time tweets. Real time tweets are very random in nature. The initial train tweets obtained for the classification were highly irrelevant to the problem domain as they were too noisy in terms of the meaning of the tweet. It was then realised after a series of observations that Twitter data is too dynamic and fast paced that the tweets of interest are generally lost in the humongous number of tweets generated every second.
After the above observation, we developed a novel approach of identifying category specific keywords or in other words domain knowledge for categories that would have a greater impact on fetching those tweets that would be less noisy and more relevant in terms of training data set.
Hence below is sample of category specific keywords we identified from tweets to build a dictionary that would eventually be used as a base in identifying right set of tweets

Category	Sub Category	Keywords
Travel	Cruise	cruisetravel,cruisewithme,CruiseWithTribute,DinnerDanceCruise,dreamcruise,njcruisenights,Ponant_Cruises,rivercruisediva,rivercruises,ScenicCruise
Arts & Entertainment	Movies	movie,cinema
Food & Drink	Dining	"dining happy","dining food","dining tasty","dining delicious","dining date"

Exploratory Visualization:

Tweet density across different categories





From the above visualization we realised that not all categories trend regularly at any given point of time and there are few categories like technology, Art, entertainment that trend at regular intervals.

This insight helped us prepare the training data in a way that covers wide variety of possibilities for categories like technology, Art, entertainment etc so that the precision/accuracy for these categories is relatively high.

Algorithm and Techniques

The core model revolves around classifying unstructured text, *Tweets*, into relevant IAB segments and hence the very first task was the selection of a classification Algorithm that scores optimal on the following selection criterion

1. ***F1 Score*** *-- which is a measure of accuracy. https://en.wikipedia.org/wiki/F1_score discusses it furth*
2. ***Training time****: Time required to train the classifier.*
3. ***Prediction time:*** *Time taken to predict -- test data*

The project evaluated following Algorithms.

- *Naive Bayes,*
- *Support Vector Machine*
- *Decision Trees*
- *Convolutional Neural networks*

The evaluation test set consisted of 10k Tweets which were divided into 70/30 training and test set split using sklearn-train_test_split function.

The table summarizes the comparison results

Algorithm	**Training set** (Number of Tweets)	**Test set** (Number of Tweets)	**F1-score**	**Training time** (in minutes)	**Prediction time** (in second)
Naive Bayes	7000	3000	0.55	3	2
SVM	7000	3000	0.65	20	3
Decision Tree	7000	3000	0.68	3	3
CNN	7000	3000	0.75	15	3

The algorithm that performed above and beyond the other classification Algorithms considered was Convolutional Neural network. We expect the accuracy of CNN to further increase when training is performed on larger dataset, *described below in the Benchmark section*.

CNN has several hyper-parameters that were tuned and are worth mentioning:

loss function: Although the project used default loss function, provided by TensorFlow, it's also possible to use absolute error or a couple other options that are more resilient to outliers. In this case we removed the significant outliers from the dataset up front, and my exhaustive grid search yielded the best results with least squares.

learning_rate: this represents how much one trusts the change of each additional weak learner. However, since I tuned with the n_estimators parameter I didn't mess with this much (a lower learning rate with more learners should behave similarly to a higher learning rate with fewer learners).

***n_estimators*:** This is just the number of boosting steps to go through, could also be though of as the number of weak learners used in sequence.

***max_depth*:** how deep any given weak learner can go, which can help avoid overfitting.

***min_samples_split*:** how big a leaf in a given tree is allowed to be.

For initial evaluation default parameters were used to see how the model did out of the box, then during actual training, when CNN was trained 3 million Tweets, the parameters were tuned to see which combination yielded the best performing learner.

Benchmark:

Model Benchmark:

The model was trained for 11 IAB categories/labels and a training set consisting of 3 million+ tweets distributed across these categories, that were used as training set. We measured **accuracy** and **loss** after stipulated number of steps to ensure the loss tends/converges to zero as we increase the number of steps.

The table summarizes IAB categories/labels and number of tweets per category.

IAB category/Label	Number of Tweets
Arts & Entertainment	230432
Family & Parenting	235768
Fashion	258808
Food & Drink	283022
Hobbies & Interests	179218
Religion & Spirituality	274056
Science	281992
Society	258812
Sports	340703
Technology & Computing	259477
Travel	288630

We need to define few terms that are necessary to benchmark our model and are commonly in benchmarking any Neural network related models.

1. **Epoch:** one pass of all the training examples
2. **Batch size :** the number of training examples in one pass. The higher the batch size, the more memory space you'll need.
3. **Step:** Number of tweets used in a training iteration.

For our use case we used following numbers for above parameters

Epoch	Batch Size
2000	64

We also benchmarked the model's performance with respect word embeddings available specifically for twitter data. The twitter word embeddings can be found at [stanford Global Vectors for Word Representation]

The GloVe model was trained on the global word-word co-occurrence matrix, which tabulates how frequently words co-occur with one another in a given corpus.

The benchmarking results are detailed below, for the set of parameters described.

Steps	Loss	Accuracy
2000	2.21507	0.562134
2600	1.92649	0.623792
3400	1.65499	0.674398
4000	1.49775	0.718656
4600	1.37581	0.73929
5400	1.2461	0.766755
6200	1.14148	0.790692
7000	1.04623	0.803317
8400	0.911524	0.835262
12000	0.69295	0.86411
14400	**0.613263**	**0.878898**

- The cross entropy loss reduced to 0.61 when 14400 steps were used compared to 2.21 for 2000 steps.
- The accuracy, on test data, increased to 0.87.

On-target rate Benchmark:

On-target rate is defined as the percentage of total campaign impressions served to the intended audiences.This is direct measure of the effectiveness of an Advertising campaign.

We can benchmark the On-target rate of IAB segments and Keywords, predicted by our Advertising brain, against the BAU IAB segments and Keywords that Media buyers use for purchasing the media.

Demand side platforms (DSPs) are used by media buyers for bidding for Keywords and IAB segments. DSPs provide a past one month summary of segments and keywords that performed best and we can test our segments and Keywords against those.

The steps for calculating On-target rate are detailed below.

1. Input the Keywords and IAB segments, predicted by our Advertising brain, in a DSP like DoubleClick Campaign Manager (DBM).
2. Calculate the On-target rate generated by the segments & Keywords from our tool.

An On-target rate between 25%- 35% is considered as phenomenal and hence if we are able to achieve this, or surpass this rate, than we have surpassed the industry standards.

III Methodology:

Data Preprocessing:

The project employs a stub, *included as the part of the project deliverable*, that can extract Tweets in real time from Twitter stream.

Twitter data is quite tricky to handle as it has a 140 character limit. So any user would tend to convey his/her idea in those 140 characters. This may result in shortened word forms and emojis being used in the tweet.

The stub extracts following fields from Twitter stream

1. **Twitter Id:** unique Id used by Twitter to identify accounts.
2. **Text message**: The actual text message tweeted -- by a user.
3. **Tweet creation time**: Timestamp when the Tweet was created/Tweeted.
4. **Tweet Hashtag:** Hashtag,if any, associated with the tweet.
5. **Tweet location**: Geo location from where the tweet was tweeted.

All these fields are extracted and stored as comma separated fields in a csv file, one line per record, that can be analyzed further by machine learning toolkit -- also developed as the part of the experiment.

Implementation:

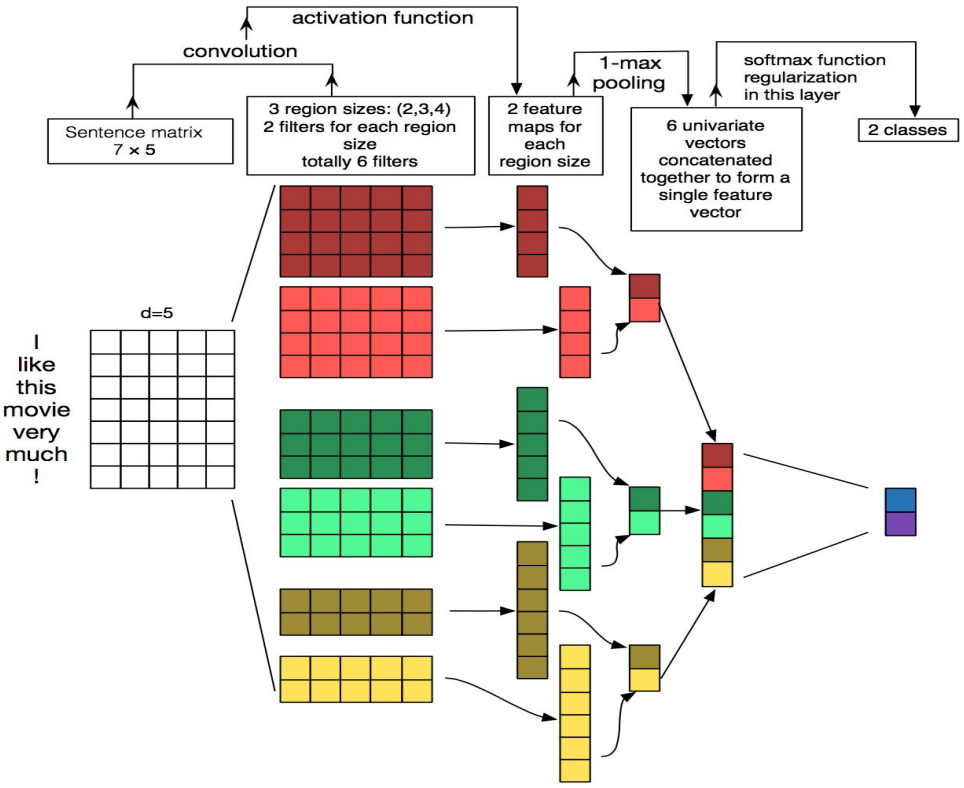
The project leverages on TensorFlow library for creating/building the Convolutional Neural Network (CNN).

CNNs are basically just several layers of convolutions with *nonlinear activation functions* like ReLU or tanh applied to the results. In a traditional feedforward neural network we connect each input neuron to each output neuron in the next layer. That's also called a fully connected layer, or affine layer. In CNNs we don't do that. Instead, we use convolutions over the input layer to compute the output. This results in local connections, where each region of the input is connected to a neuron in the output. Each layer applies different filters and combines their results. During the training phase, **a CNN automatically learns the values of its filters** based on the task you want to perform.

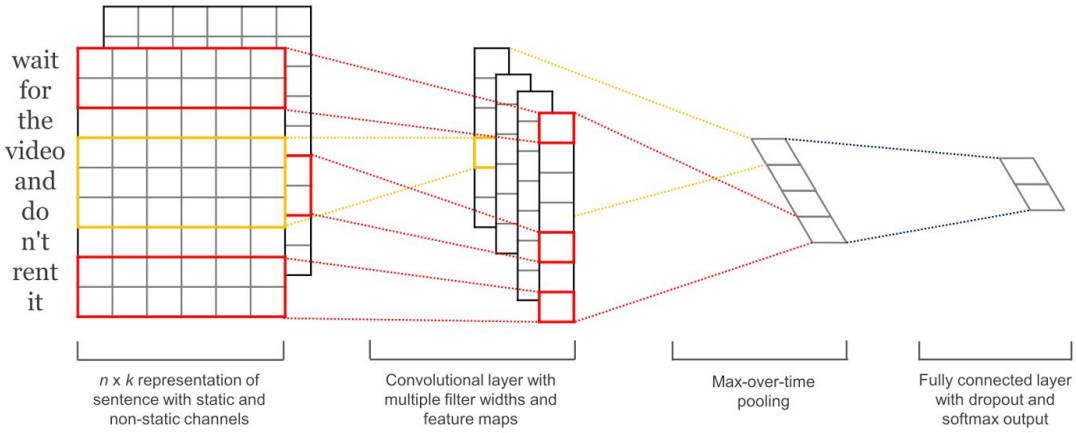
The training input, *for our use-case*, consists of Tweets representing token of words as vectors. These vectors are word embeddings that can be thought of a contextual vocabulary used by the neural network to disambiguate the context in which a given word is appearing in a sentence. For a 10 word sentence using a 100-dimensional embedding we would have a 10×100 matrix as our input. http://www.fredericgodin.com/software/ has provides more details on word embeddings and can be used for referance.

The diagram, below, depicts a high level CNN architecture employed for the project.
The input are the tweets represented as a matrix. Each row of the matrix corresponds to one token, typically a word in a tweet. That is, each row is vector that represents a word. Typically, these vectors are word embeddings (low-dimensional representations) of Twitter GloVe (Global Vectors for Word Representation), but they could also be one-hot vectors that index the word into a vocabulary. For a 10 word tweet using a 100-dimensional embedding we would have a 10×100 matrix as our input.

we typically use filters that slide over full rows of the matrix (words). Thus, the "width" of our filters is usually the same as the width of the input matrix. The height, or region size, may vary, but sliding windows over 2-5 words at a time is typical. Putting all the above together, a Convolutional Neural Network for a tweet "*I like this movie very much*" may look like this



Here we depict three filter region sizes: 2, 3 and 4, each of which has 2 filters. Every filter performs convolution on the sentence matrix and generates (variable-length) feature maps. Then 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded. Thus a univariate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer. The final softmax layer then receives this feature vector as input and uses it to classify the tweet.



CNN HYPERPARAMETERS

STRIDE SIZE

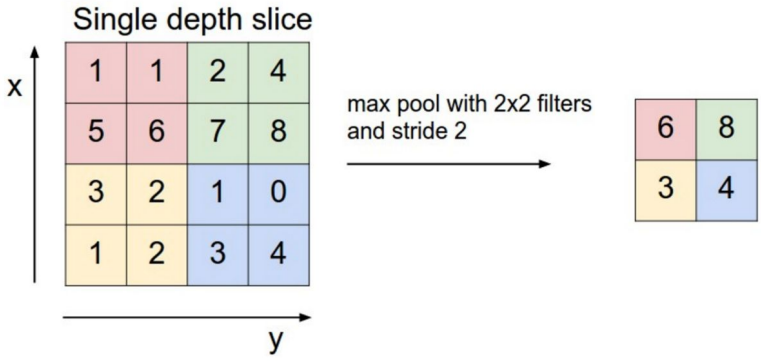
Another hyperparameter for your convolutions is the stride size, defining by how much you want to shift your filter at each step.

In our experiment we used filters of various sizes - "2,3,4"

Total number of filters used 128

POOLING LAYERS

A key aspect of Convolutional Neural Networks are pooling layers, typically applied after the convolutional layers. Pooling layers subsample their input. The most common way to do pooling it to apply a max operation to the result of each filter.



Max pooling in CNN. Source: http://cs231n.github.io/convolutional-networks/#pool

One property of pooling is that it provides a fixed size output matrix, which typically is required for classification. For example, if you have 1,000 filters and you apply max pooling to each, you will get a 1000-dimensional output, regardless of the size of your filters, or the size of your input. This allows you to use variable size sentences, and variable size filters, but always get the same output dimensions to feed into a classifier.

Pooling also reduces the output dimensionality but keeps the most salient information.

CHANNELS

Channels are different “views” of your input data. We could imagine having various channels as well: You could have a separate channels for different word embeddings (word2vec and GloVe for example), or you could have a channel for the same sentence represented in different languages, or phrased in different ways.

Refinement

The model's performance with respect word embeddings available specifically for twitter data increased by good 6.3% from 81.5% to 87.8%. This is because word vectors capture a lot bigger meanings of the words on which they are pretrained.

Also we increased the filters size upto 4 so as to learn good representations automatically, without needing to represent the whole vocabulary

IV RESULTS

Training and Evaluation

Training parameters

batch_size: 64

Number of training epochs: 2000

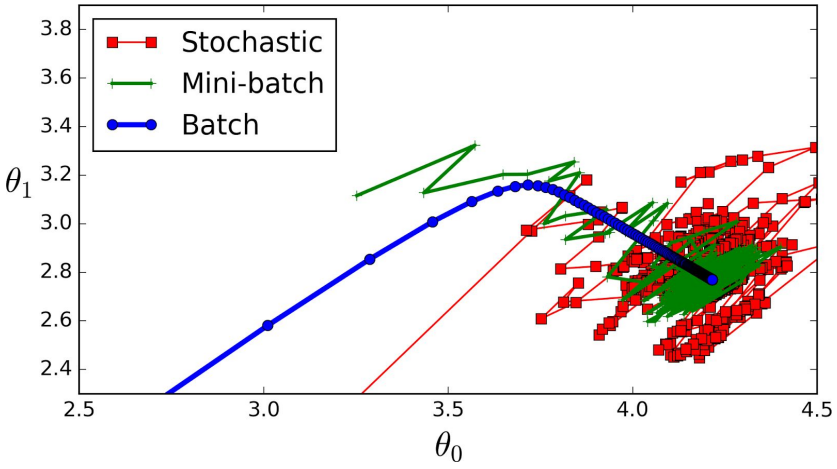
Evaluate model on dev set after this many steps: 200

Batch size Advantages:

- It requires less memory. Since you train network using less number of samples the overall training procedure requires less memory. It's especially important in CNN as we are not able to fit dataset in memory.
- Typically networks trains faster with mini-batches. That's because we update weights after each propagation. If we used all samples during propagation we would make only 1 update for the network's parameter.

Disadvantages:

- The smaller the batch the less accurate estimate of the gradient. In the figure below you can see that mini-batch (green color) gradient's direction fluctuates compare to the full batch (blue color).



The model's final accuracy is ~87.8% and generalises well across different categories as it learned the category specific keywords for various categories.

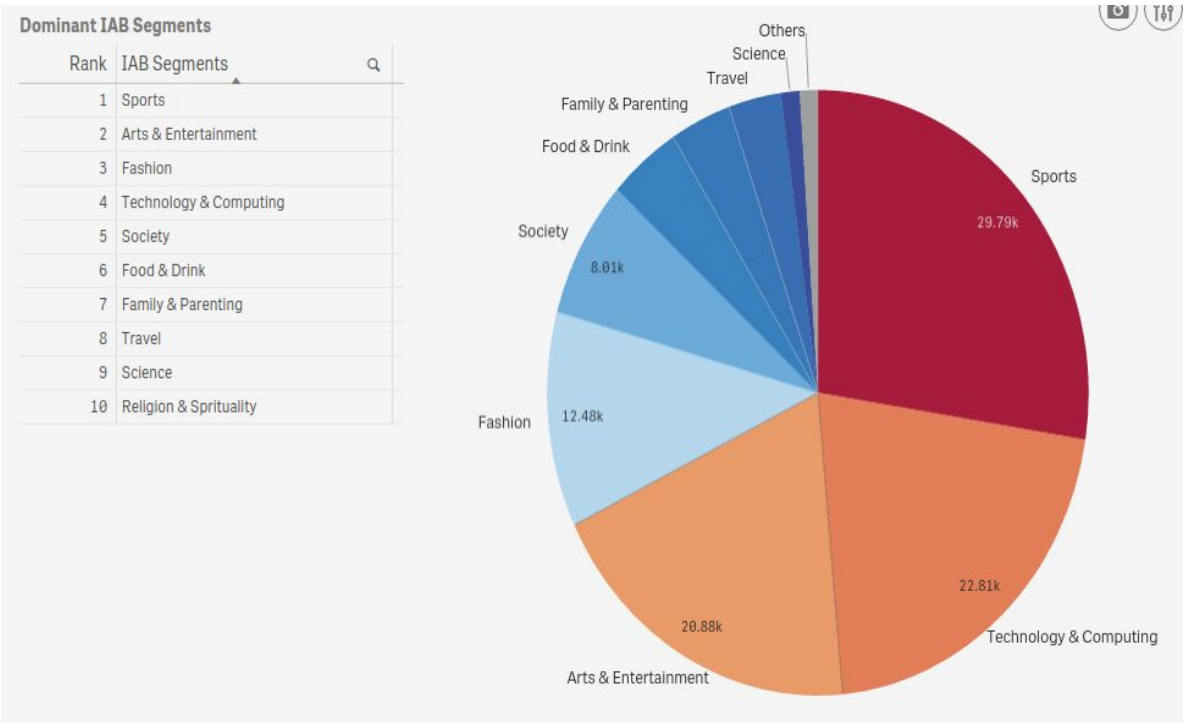
Justification:

- As mentioned earlier, the model's performance with respect word embeddings used specifically for twitter data increased by good 6.3% from 81.5% to 87.8%. This is because word vectors capture a lot bigger meanings of the words on which they are pretrained.
- The cross entropy loss reduced to 0.61 when 14400 steps were used compared to 2.21 for 2000 steps.
- The accuracy, on test data, increased to 0.87.

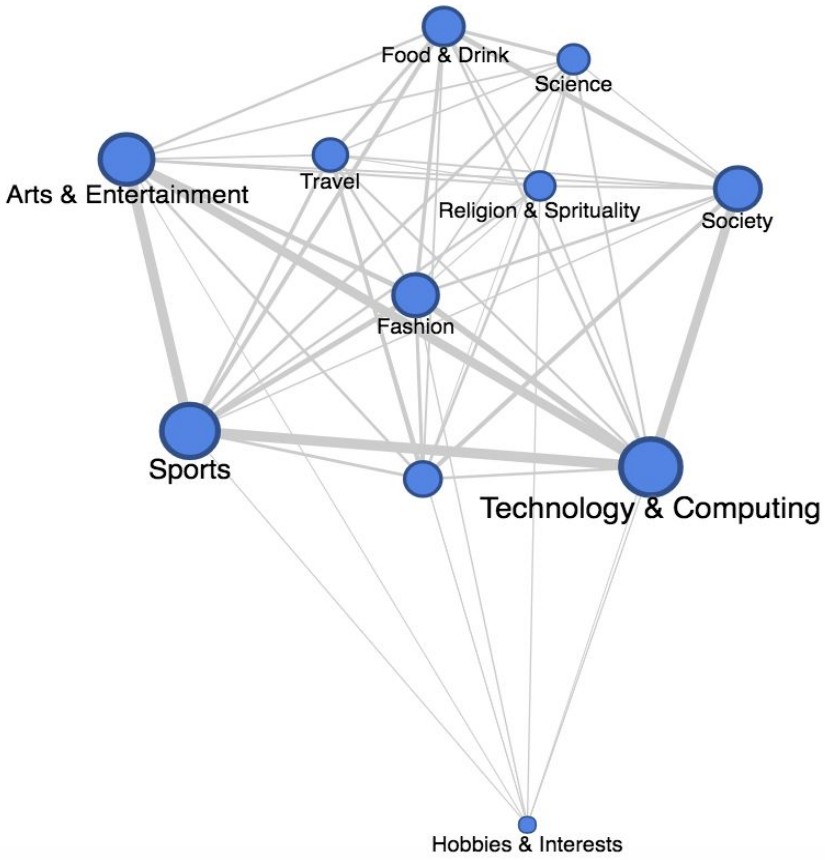
V CONCLUSION

Free-Form Visualization

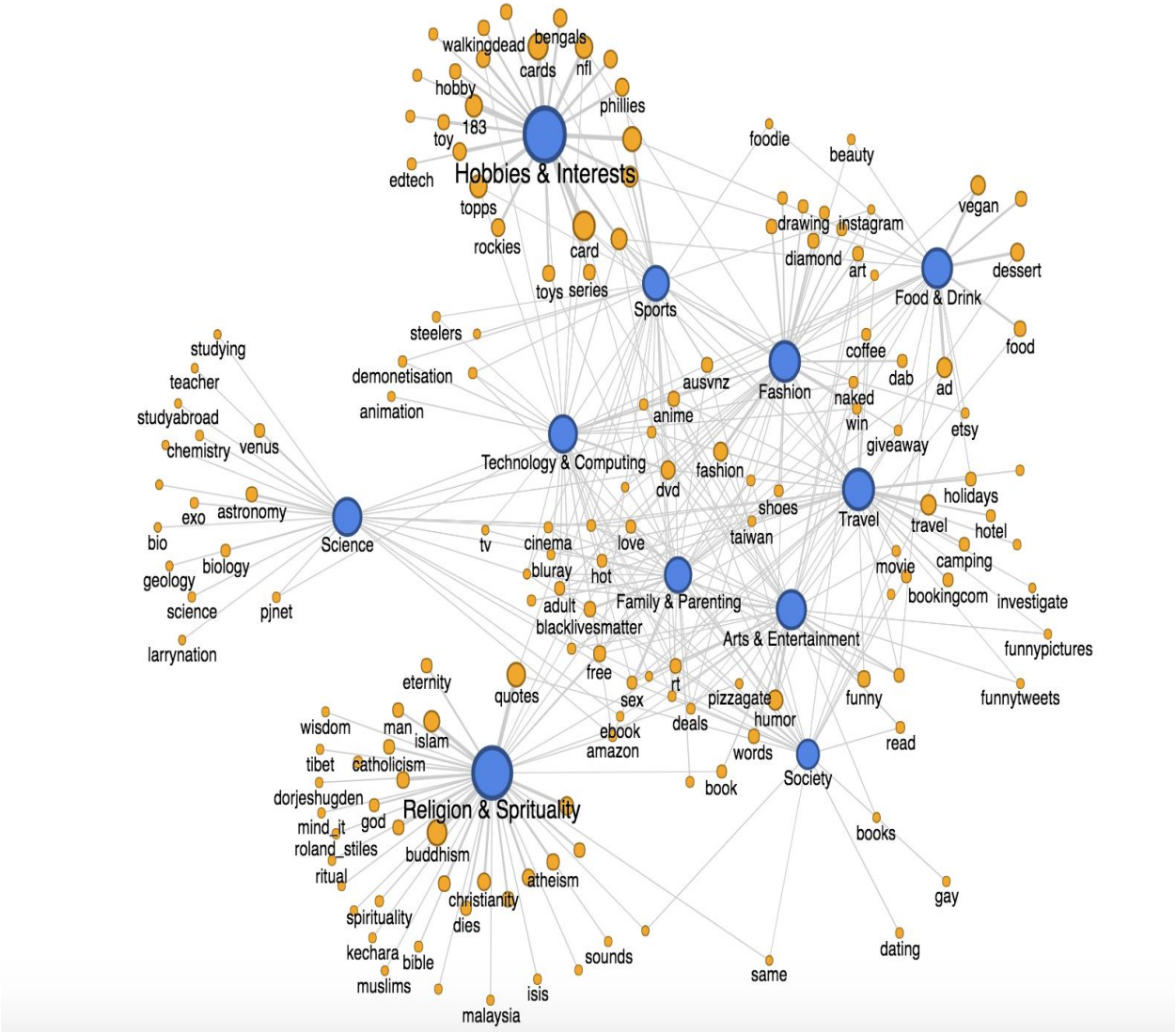
- **Audience reach in Twitter, per IAB segment**



- **Correlated IAB segments:**



- **Trending IAB Segments & Keywords**



Reflection

A high level project design is postulated below.

- Twitter stub, *developed as the part of the projec*t, would extract Tweets from Twitter stream and store it in the disk as flat files.
    - We can have a more robust design by storing Tweets in-memory cache but this is not related to the main outcome of what we plan to achieve.

- A Convolutional Neural Network (CNN) would be trained to classify Tweets for IAB Level one and Level 2 categories.
    - The trained model would be stored as a pickle object.
    - Tensor flow would be used for building and training the model.
    - The model file along with the training data would be provided as the part of the project -- sets of python files.

- The results of the classification would be visualized as a dashboard using Google Fusion tables and would be the part of the project

Improvement

CNN vs RNN

We do care a lot where in the sentence a word appears. Pixels close to each other are likely to be semantically related (part of the same object), but the same isn't always true for words. In many languages, parts of phrases could be separated by several other words. The compositional aspect isn't obvious either. Clearly, words compose in some ways, like an adjective modifying a noun, but how exactly this works what higher level representations actually "mean" isn't as obvious as in the Computer Vision case.
Given all this, it seems like CNNs wouldn't be a good fit for NLP tasks. Recurrent Neural Networks make more intuitive sense. They resemble how we process language (or at least how we think we process language): Reading sequentially from left to right. RNNs have shown great success in many NLP tasks. At this point it should be mentioned that the most commonly used type of RNNs are LSTMs, which are much better at capturing long-term dependencies.

Going forward, I would like to implement RNN for the reasons mentioned above. The tweets classification accuracy can then can be easily pushed upwards of 90% and more.