Parsing Reddit Data (Michael Matheny)

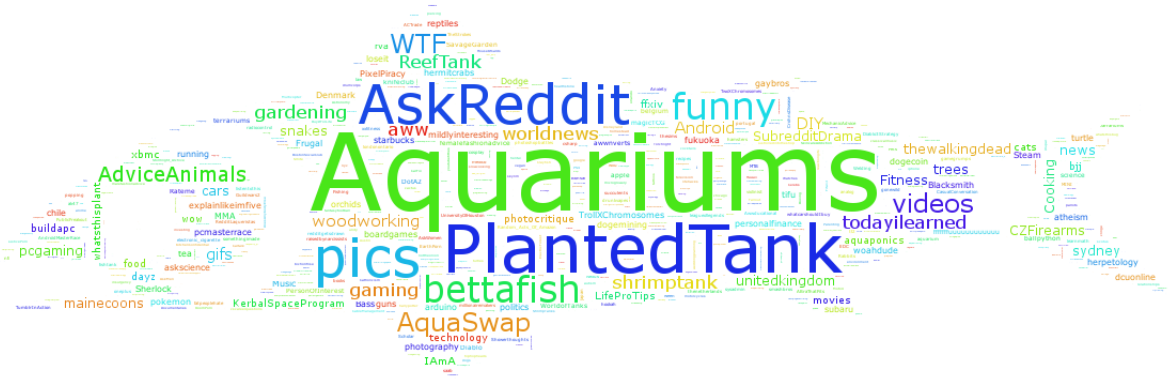
Reddit is very unique in that it provides an easy interface for people to write bots and even actively encourages the mining of their data. They do impose rate limits of 1 request per 2 seconds which can make data mining a bit slow in some cases. The data of every reddit page can be viewed as json file and there is an easy to use wrapper called Praw that provides a high lever interface for building reddit bots. For this project we didn't need all the functionality that Praw offered, so we wrote a wrapper around some of Praw allowing us to stream tuples containing only relavant comment data belonging to a particular subreddit.

tuple info	Explanations
body	The comment text
gilded	Whether this comment was given gold
score	The number of upvotes/downvotes
utc	The time the comment was posted
subreddit	The name of the subreddit
author	The user of the comment writer
submission_title	The title of the submission that the commentator posted under
submission_author	...
submission_score	...
submission_utc	...

We also wrote a lower level higher performance implementation for parsing user comments using the python requests library.

Clustering Users (Michael Matheny)

We want to find groups of users with similiar interests. To do this we decided to sample reddit users and then cluster the users based on the number of comments that they had posted in each subreddit.



Mining Users

We wrote a small script that request random subreddits and then scrapes the first few pages. After approximately 5 hours we had around 250k usernames. Praw can be used to scrape user histories, but it takes about a minute per 25 user comments. Writing our own parser using the requests python library sped this up to around 25 comments every 2 seconds. Each user can take up to 16 seconds, so we let the script run continously on a EC2 VM and at the time of writing this we had around 33k user comment histories.

Clustering Users

To cluster the user data we represent each user as a vector where the number of comments in a subreddit gives the value at an index in the vector. The matrix containing all the user comments after removing deleted users is approximately 20000 users by 30000 subreddits, but it is extremely sparse with only around .1 % of the elements nonzero. We use cosine distance as our distance metric for the data since we only want to compare users interests.

We initially clustered the data using the scikit learn cluster library, and experimented with numerous different methods and distance functions. We found that for this data cosine similarity seems to provide the most interesting and relevant clusters which makes sense since we only care about user interests and not the number of comment they produced.

As the data got larger many of clustering method began failing, taking too long, or would run out of memory. The only methods in scikit that work for this are Kmeans, Kmeans batch, and DBSCAN. DBSCAN clusters by density which doesn't make much sense for this data. Kmeans takes extremely long (10 minutes). Kmeans batch is extremely quick, but the quality of the clustering at times was significantly worse than Kmeans. We wanted to see if we could better so we looked into several methods.

1. JLT: Calculations would suggest that we need to accept quite high error per data point for there to be a sizable dimensionality reduction:

$$k = .01^{-2} \log(20000) = 99034$$

$$k = .1^{-2} \log(20000) = 990$$

2. SVD: Since the data is very sparse computing the first few rows of the SVD is probably not too bad.

3. Kmeans with LSH.

Our implementation of Kmeans with LSH works by projecting each point onto a set of random vectors. Each random vector provides a single bit. For each point we keep b bits grouped into r integers. At the begining of each iteration of Kmeans LSH we hash all centroids into r hash tables and then generate a list of the centers that each point collides with. The center that the point collides with the most determines the points cluster.

Results

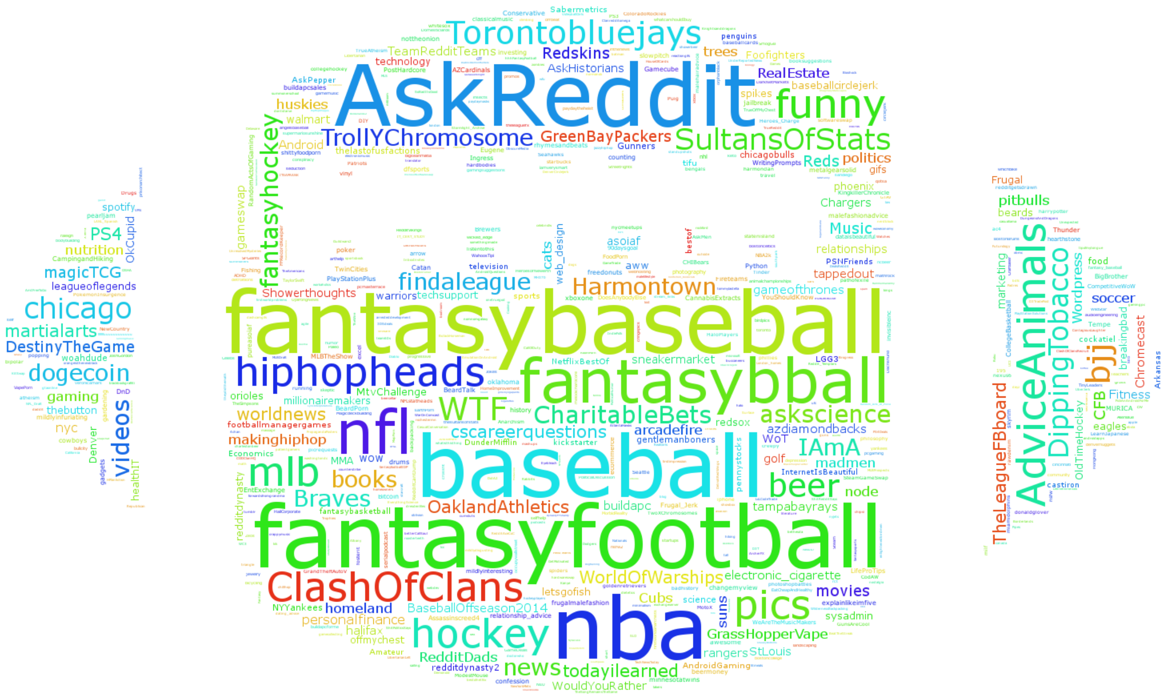
To compare implementations we found hyperparameters for each method that gave almost exactly the same average distances between points and their cluster centers as vanilla Kmeans. We used $k = 100$ for all results. SVD and Kmeans are both written in C with

Algorithm	Parameters	Time (sec)
Kmeans Vanilla	-	287
Kmeans SVD	k = 60	13
Kmeans JLT	k = 100	28
Kmeans LSH	b = 6, r=30	283

python handles which makes comparing times between these algorithms a bit error prone. For instance the label assigment code that maps points to centers in LSH takes cummulatively around 210 seconds to run and most of this time is from a single max operation over a dictionary of counts.

Clusters

Below are some clusters we found as subreddit clouds.


AskReddit
Torontobluejays
funny
SultansOfStats
TrollYChromosome
fantasybaseball
fantasybball
fantasyfootball
fantasyhockey
baseball
nfl
nba
mlb
hiphopheads
hockey
ClashOfClans
WTF
CharitableBets
askscience
IAmA
beer
pics
Braves
books
worldnews
findaleague
Harmontown
OaklandAthletics
chicago
dogecoin
magicTCG
videos
AdviceAnimals
DippingTobacco
TheLeagueFBboard
bjj


vegan
vegetarian
AskReddit
vegancirclejerk
running
IAmA
relationships
legaladvice
Random_Acts_Of_Amazon
washingtondc
WarshipPorn
worldnews
aww
stephenking
nutrition
Winnipeg
exmormon
videos
chicago