# A Survey of Context Aware Recommender Systems

Sugandha Agrawal
University of Virginia, sa5ev@virginia.edu

***Abstract* – We live in an ocean of information, and the advancing computing technology has helped us devise solutions to handle the never ending stream of information we face in our daily lives. A range of software tools such as recommender systems, question answering systems, and sentiment analysis tools are constantly being updated, improvised, and researched upon to expand our information handling toolkit. These software tools all share commonalities that characterize their strengths and weaknesses. One area of weakness is the failure of these tools to capitalize on contextual information surrounding users. With the advent of microminiaturization, devices are now mobile and ubiquitous, meaning that humans are demanding new kind of help from software tools – help that is personalized based on where the user is, what the user is doing, who the user is with etc. Thus it is important for us to understand the underpinnings of context aware recommender systems, so we can better address the needs of consumers. Given the important of context, this paper applies an innovative way to cater to newfound demands of consumers by delivering information to them visually, which helps enhance their understanding and grasp of knowledge around them.**

*Index Terms* – Context aware, question answering systems, recommender systems, sentiment analysis, TREC

## 1. MOTIVATION AND GOALS

With the advent of computers, handling and processing information has been becoming significantly easier. The microminiaturization of circuits has made it possible for a common man to hold a thousand times more computing power than that of Apollo 11 mission's guidance computer systems [1]. It is now possible to explore hundreds of restaurants in your neighborhood with a few clicks and swipes on your0 portable devices. Unfortunately with this power to access vast amounts of information comes the problem of making sense of this information. Without software tools to help us break down information into manageable pieces, it is impossible for our brains to filter gigabytes, or terabytes of information.

To keep up with the pace of information production, researchers have been working with principles of natural language processing and information retrieval to create products such as question answering systems, recommender systems (including context aware systems), and sentiment analysis tools. To delve deeper into understanding the workings of the above mentioned systems, my research comprised of two parts, a) researching what characteristics make for successful systems and b) implementing a temporal aware sentiment analysis visualization tool.

The first part of my research focuses on comprehending the underpinnings of question answering systems, recommender systems, and sentiment analysis tools. It is crucial to understand how these systems differ, what their main components are, and what common features tie these main components together. A good grasp of these concepts enhances one's working knowledge of information handling software systems – a skill becoming increasingly valuable in today's world.

I focused in particular on context aware recommender systems (CARS) since that is a prime research area due to its various practical applications. As computing devices such as mobile phones and tablets are becoming more ubiquitous, it is highly important to be able to capitalize on the rich reservoir of contextual information such as time of day, social setting, location, weather, and season surrounding a user. Incorporating context of a user in making a recommendation improves the accuracy of the recommendation, thus allowing software to better match the user's needs. The inclusion of contextual information coupled with a sentiment analysis framework, which helps the recommendation process by assessing sentiment of reviews users have given the products, leads to a well-functioning CARS.

With the real time demand for CARS, that includes both a contextual and a sentiment analysis component arises a complimentary need – that of wanting to visualize the sentiment of a product, or a recommendation over time. This visualization provides a user friendly metric to the user that enhances their decision making capability by delivering information visually. Studies have shown that our brains not only process visuals faster, but they also retain and transmit much more information when it's delivered visually [2]. As we are living in a world where the rate at which information is being created is exponentially increasing, it becomes highly important to achieve high levels of retention and transmission of information for humans to keep up.

Therefore the second part of my project comprises building a sentiment analysis visualization tool that takes advantage of temporal dimensions in Twitter data to deliver meaningful insight into products for consumers. The application allows a user to specify the product they wish to track the sentiment of (for example the topic can be "apple", "twitter", "microsoft", "google"), and the user is then presented with a visual depiction of sentiment over time for that product.

## 2. QUESTION ANSWERING SYSTEMS – PRECURSOR OF RECOMMENDER SYSTEMS

Question answering (QA) systems are a computer science discipline under the fields of information retrieval (IR) and natural language processing (NLP), which is concerned with building systems that automatically answer questions posed by humans in natural language [3]. A request for a recommendation can be basically viewed as a question. Requests to recommender systems such as "What are the best sites to see around town?" or "Where can I find the best Indian food?" are all specialized questions. Thus the history of QA systems is important to discuss, as it offers a view of the building blocks of recommender systems.

### 2.1 Early QA Systems

Early QA systems were all closed domain, meaning focusing on one particular domain of knowledge as it was easier to program rules and data pertaining to just one area. Two such early closed domain QA systems were BASEBALL and LUNAR. BASEBALL was a computer program created at MIT in 1971 to answer baseball league questions such as "Where did each team play on July 7?" The system read the questions from punch cards, analyzed it syntactically (i.e. looking up words and idioms in a dictionary and determining the phrase structure) to produce attribute-value pairs specifying the information given and the information requested. The answer was then searched and constructed from a knowledge base that was rigidly structured and hand written by experts [4]. This was a common attribute of all early QA systems because in a limited domain or context, a small vocabulary is sufficient, syntactic analysis is considerably simplified, the data is simpler, and the subject-matter is familiar, thus easily lending itself to programmed hard coded rules. For example, LUNAR only had a vocabulary of about 3,500 words.

LUNAR was another closed domain, or topic defined, QA system developed in 1971 that answered questions about geological samples brought back by Apollo 11 missions. This system was also based on a structured knowledge base, carefully curated by humans to enhance accuracy [5]. SHRDLU was another QA system developed in the 60s and the 70s by Terry Winograd that proved to be highly successful. The program simulated activities of a robot inside a toy world, allowing users to ask the robot questions about the state of this toy world. Strength of closed domain systems include simplicity and ease of encoding rules for a fixed domain in a computer program. However the weaknesses of such closed domain QA systems outweighed the strengths. These systems had limited practical applications in real life, as they failed to generalize to the wider information demands of the common man. This resulted in such systems being unable to generate widespread interest outside the computer science research field. These systems were also brittle, prone to unexpected failures especially at the boundaries of their corresponding domain knowledge. Thus a shift towards simpler, pattern based techniques started occurring in the late 1980s, following a rise in computational linguistics.

### 2.2 Modern Day QA Systems

There was a shift from closed domain to open domain QA systems, eliminating the limitation to just one topic of context. Such systems resemble modern day QA systems, relying on statistical processing of a large, unstructured, natural language text corpus that includes information from eclectic sources such as newspapers, online databases, and the web. A variety of sources feeding into a large corpus has the advantage of data redundancy. The right information appears in many forms (for example on the web), thus lessening the burden on QA systems. Moreover the correct answers can be filtered from false positives more easily.

One of the first such open domain system was MURAX, developed in 1993 that used encyclopedias as its knowledge base to answer questions about practically anything [6]. These modern day QA systems include a sophisticated NLP pipeline, which serves to enable the following architecture: (1) Analyze the question, produce an IR query, (2) retrieve a collection of documents corresponding to the IR query, (3) extract answer candidates (strings) from the documents, and lastly (4) select the best answer string(s). Figure 1 shows an extensive layout of a typical QA system containing the above mentioned pipeline [7].

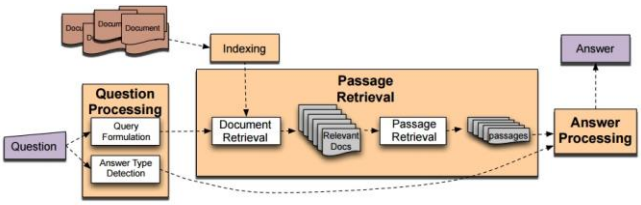


FIGURE 1
FULL NLP QA ARCHITECTURE

### 2.3 Question Processing

The question processing stage detects the question type, answer type needed to satisfy the question, extracts relations between entities, and formulates queries to send to the corpus. Consider for example the following question: *They're the two states you could be reentering if you're crossing Florida's northern border.*

This question would be broken down as such after the question processing stage:

- **Answer type**: Location (US state to be specific)
- **Query**: two states, border, Florida, north
- **Focus**: the two states
- **Relations**: borders (Florida, ?x, north)

There are 6 course classes for answer type – (1) abbreviation, (2) entity, (3) description, (4) human, (5) location, and (6) numeric. These course classes are further subdivided into 50 finer classes as depicted in Figure 2. These answer types help classify questions, for example by using a maximum entropy model. However determining the answer type is not that easy. Questions with "who" can have individuals or companies as their answers, or questions with "which" can have both humans and things as their answers. For example "Which president signed the National Banking Act?" In modern QA systems, machine learning techniques are used to automate answer type classification. Training data is annotated with answer types depicted in Figure 2 [8]. Classifiers are then trained for each answer type class using a rich set of features such as part of speech tags, parse features (headwords), named entities, semantically related words, and question words or phrases. Once the answer type has been chosen, the question is transformed into lexical terms, which represent an approximation of the question. For example, the question about Florida stated above can be adequately represented by the following lexical terms: two states, border, Florida, north.

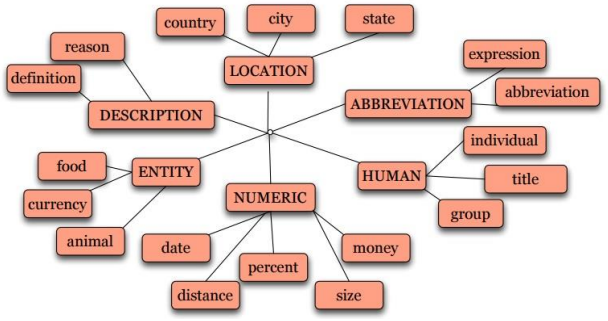


FIGURE 2
ANSWER TYPE TAXONOMY

## 2.4 Passage Extraction

Next stage is the passage extraction, where documents that contain all desired keywords are selected. Passages are selected by rule-based classifiers that may consist of features such as:

(1) Number of named entities of the right type in the passage
(2) Number of query words in passage
(3) Number of question N-grams in passage
(4) Proximity of query words to each other in passage
(5) Longest sequence of question words

One may have to perform keyword adjustment depending on the number of passages retrieved. If the number of passages returned from the IR query is too low, that implies the query is too strict and needs to be relaxed which means dropping a keyword. On the other hand if the number of passages is above a certain threshold, that entails adding a keyword as the query is too relaxed. Subsequently, the passages are sorted depending on a score that signifies how likely it is that a specific passage has the correct answer to the question. The passage scoring can be based on a variety of models, or a combination of different scores. Here is an example of a model that uses a combination of 3 scores to sort passages:

**Score 1:** Number of words from the question that are recognized in the same sequence in the window
**Score 2:** Number of words that separate the most distant keywords in the window
**Score 3:** Number of unmatched keywords in the window

## 2.5 Answer Extraction

To return specific answers from ranked passages, an answer type named entity tagger is run on passages that returns only strings with the right type. For example for the question below with the answer type *PERSON*:

*Who is the Prime Minister of India?*

The following candidate string is returned:

*Narendra Modi, Prime Minister of India, had told left leaders that the deal would not be renegotiated.*

In the case where there are multiple candidate answers, which may easily occur due to the vast size of the knowledge base nowadays, answers are ranked by using features like the ones listed below.

- **Answer type match**: candidate contains a phrase with the correct answer type
- **Pattern match**: regular expression pattern matches the candidate
- **Question keywords**: number of question keywords in the candidate.
- **Keyword distance**: distance in words between the candidate and the query keywords
- **Novelty factor**: A word in the candidate is not in the query
- **Apposition features**: candidate is an appositive to the question terms
- **Punctuation location**: candidate is immediately followed by a comma, period, quotation marks, semicolon, or exclamation mark
- **Sequences of question terms**: length of the longest sequence of question terms that occur in the candidate answer

## 2.6 QA Systems Evaluation

Since 1999, open domain QA systems are evaluated through TREC (Text Retrieval Conference) which is an ongoing series of workshops and competitions held annually focusing on different IR research tracks. The QA track in TREC consists of answering a set of 500 fact-based

questions, e.g., "When was Mozart born?" The answers to such questions exist in the collection as a text fragment, and the supporting information can be found in a single document. Each system returns five guesses per question with corresponding confidence intervals.

The best performing TREC QA systems have been able to answer 70% of the questions, where the questions relate to hundreds of domains. However, there are advancements in QA field to be made beyond TREC, as the questions in TREC are purely factoid questions. They fail to address complex questions such as "How do you tell if you have a cold?" or "How do you change the spawn point in Minecraft?"

# 3. CONTEXT AWARE RECOMMENDER SYSTEMS (CARS)

The needs of information aid have expanded in the last decade. There are thousands of choices to make, whether it be which speakers to buy on Amazon, or which restaurants to eat at in Manhattan. It is hard for humans to factor in hundreds of aspects, rank hundreds or thousands of choices in a convenient manner. Research has shown that when presented with too many choices, humans often end up making a poor decision that may not optimally align with their preferences [9]. Thus there has been a rise in recommendation systems (RS), which help recommend products or services to consumers to prevent them from feeling overloaded. For example, Netflix has a recommender system that makes it easy for consumers to enjoy Internet TV, as their choices have been simplified for them [10].

Unfortunately, the majority of existing approaches to recommender systems recommend items to users without taking into account any sort of contextual information such as time, place, location, or social setting such as whether the person is alone or with other people. However in many practical applications such as recommending movies or vacations, it may not be sufficient to model systems solely on users and items – it is also important to be able to recommend different things to users in different circumstances. For example, a user might want to watch a different movie on weekdays with his or her parents compared to a movie with friends on a weekend. This notion is supported by findings in behavioral research on consumer decision making, where it has been discovered that decision making is contingent on the context in which a decision is made. Therefore, accurately delivering on consumer preferences necessitates incorporating context of a user in the recommendation process.

## 3.1 Defining Context

There are numerous definitions of context, depending on the field in consideration. In E-commerce applications, the context denotes the *intent* with which the consumer makes a purchase. For example a consumer may buy a different book for personal work skills enhancement, as a gift for someone, or as a relaxing read on vacation. On the other hand, in mobile applications, context is defined as location of the user, the objects around, identity of the people around the user, and any changes in these elements. For the purposes of this paper, we will use a very comprehensive definition of context, one that can be generalized to any application. We will divide the varying definitions of context into four groups.

- **Group 1:** Physical context
  Pertaining to time, position, weather, season
- **Group 2:** Social context
  Relates to the presence, identity, and role of people around the user, e.g., is the user single, is he alone, is he married, is he at a party etc.
- **Group 3:** Interaction media context
  Denotes the device used to access the system. Mobile users have different demands that a tablet or PC user
- **Group 4:** Modal context
  Relating to the state of mind of the user, the user's goals, mood, experience, and cognitive capabilities.

Aside from the different types of context, there is another hierarchy of contexts. Overall the diverse definitions of context listed above can be classified under – representational or interactional views. Representation view considers context as a predefined set of observable attributes that do not change over time. This implies that contextual attributes are known *a priori*, and thus can be easily modeled, and accounted for in context aware applications. On the other hand, the interactional view of context assumes that there is a cyclical, dynamic relationship between contexts and activities – contexts influence activities and different activities give rise to different contexts. The dynamic nature of this view implies that contextual attributes cannot be identified *a priori*. This view assumes that the user action is induced by his or her context, however the context is not necessarily observable [11].

## 3.2 Modeling Context in Recommender Systems

Traditionally, recommender systems have been two dimensional, as they only consider two dimensions in the recommendation process – the *User* and the *Item*. Thus the traditional recommender system can be mathematically modeled by the rating function R as follows:

$$R: User\ X\ Item \rightarrow Rating$$

Here *Rating* is an ordered set. With the addition of context, the equation for a CARS is:

$$R: User\ X\ Item\ X\ Context \rightarrow Rating$$

Thus we shift from a 2D model of recommendation to a multi-dimensional one as depicted in Figure 3 [12].

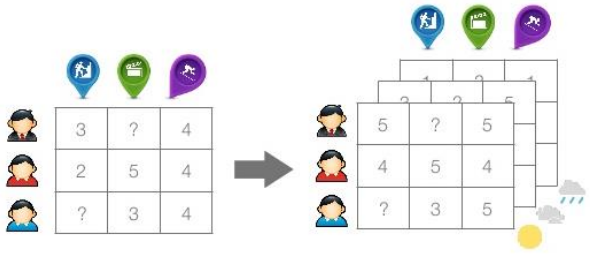


FIGURE 3
TRADITIONAL RS VS. CARS

Incorporating context can sometimes be hard, given the complex nature and the various forms that contextual information can take. One way to resolve this is to view contextual information as a hierarchical structure represented by trees. To understand this, let us look at an example. Assume an application's functionality is to recommend movies to its users. The contextual information available is of three types as listed below:

- **Theater**: the movie theaters showing the movies; it is defined as:
  Theater(*TheaterID, Name, Address, Capacity, City, State, Country*)
- **Time**: the time when the movie can be or has been seen; it is defined as:
  Time(*Date, DayOfWeek, TimeOfWeek, Month, Quarter, Year*)
  Here, attribute DayOfWeek has values Mon, Tue, Wed, Thu, Fri, Sat, Sun, and attribute TimeOfWeek has values "Weekday" and "Weekend"
- **Companion**: represents a person or a group of persons with whom one can see a movie. It is defined as:
  Companion(*companionType*)
  Attribute companionType has values "alone", "friends", "girlfriend/boyfriend", "family", "co-workers", and "others"

Two of the three above context types can be represented by the following hierarchies:

1. Theater: TheaterID → City → State → Country
2. Time: Date → DayOfWeek → TimeOfWeek
3. Time: Date → Month → Quarter → Year

This hierarchical view on context, easily modeled as trees is used in most modern CARS or profiling systems. Mathematically, this hierarchical structure can be represented by **K** – a set of contextual dimensions, where each dimension K in **K** is defined by a set of *q* attributes as such:

$$K = (K^1, \dots, K^q)$$

$K^1$ defines coarsest level of contextual information whereas the succeeding numbers define *finer* levels.

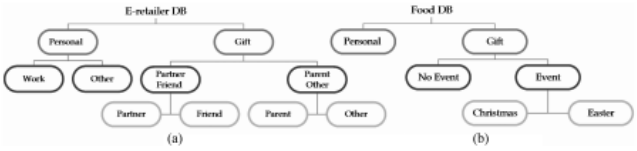


FIGURE 4
CONTEXTUAL INFORMATION HIERARCHICAL STRUCTURE IN TWO DIFFERENT APPLICATIONS

In figure 4, $K^1$ = {Personal, Gift} and $K^2$ = {PersonalWork, PersonalOther, GiftPartner/Friend, GiftPartner/Other}.

## 3.3 CARS Architecture Models

The actual recommendation process is driven by different methods such as content-based, collaborative or a hybrid approach combining the two. CARS often uses the same methods to recommend items, however their architecture differs from traditional recommender systems as there is another dimension – that of context involved. There are three main architectures for incorporating context:

1. Contextual Pre-Filtering
2. Contextual Post-Filtering
3. Contextual Modeling

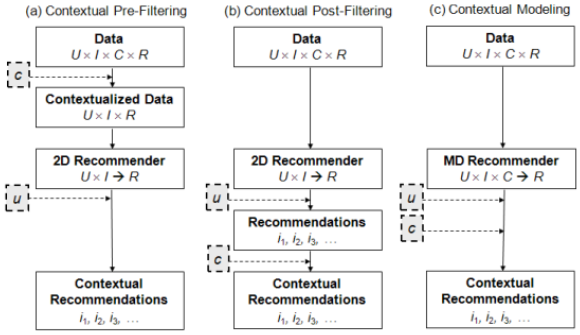


FIGURE 5
WAYS TO INCORPORATE CONTEXT IN CARS

In Contextual pre-filtering, depicted in Figure 5(a), contextual information drives data selection. This means that the context, let us assume it is called *c*, essentially serves as a query. This model acts as an exact pre-filter, for example if a person wants to see a movie on Saturday, only data records with Saturday ratings are used for the recommendation process. [Time=t] in the equation below acts as a simple contextual pre-filter. This reductionist based approach can be mathematically stated as follows:

$$\forall\, (u,i,t) \in U\ X\ I\ X\ T, R^{D}_{User\ X\ Item\ X\ Time}\,(u,i,t) = R^{D[Time=t](User,Item,Rating)}_{User\ X\ Item}\,(u,i)$$

This architecture suffers from over-specification, as using an exact context is often times too narrow. This leads to a sparsity problem where there may not be sufficient training examples for an accurate recommendation. For example, going to a Thai restaurant with a girlfriend on a Sunday is an overly specific context, where certain parts of the context are too restrictive and not significant, i.e. Sunday vs. weekend. One way to resolve this is the idea of *generalized pre-filtering* – where the context is dynamically "rolled up" to higher, more general levels when there are not enough data records for a specific context. This implies that instead of using [Time = t] as a filter, a generalized filter [Time $\in S_t$] will be used where $S_t$ acts as a *contextual segment* – a superset of context t.

A different approach to contextual pre-filtering, that achieves the same goal of reducing the problem of multidimensional recommendation process to a 2D modeling process involving *Users* and *Items as above*, is item and user splitting. The idea behind splitting is that the nature of an item for example, from the user's point of view, may change in different contextual conditions. Thus we can consider the item as multiple items – one for each contextual condition. Similarly, it may improve the accuracy of predictions to treat a user as multiple users if the user had significantly different preferences in different contexts. When the two splitting procedures are combined, we end up with User-Item splitting which splits items followed by splitting users. There are disadvantages connected to performing such splits – mainly the fact that it could lead to a sparsity problem. Therefore splitting should only be used when it is critical to improving the accuracy of the application. This can be judged by performing statistical tests such as $t_{mean}$ t-test that indicate whether user and item behavior differ significantly in different contexts.

Contextual post-filtering shown in Figure 5 (b) differs from pre-filtering in that the context is not used as an input for the recommendation process. The list of recommendations is first created, and then the context is used to filter out the recommendations. The contextual attributes are treated like features, and recommendations that do not have a significant number of features (above a minimal threshold) are filtered out. Then the items are ranked based on how many of the contextual features they have. This is a heuristic approach to implementing post-filtering. In contrast, a model-based approach builds predictive models that calculate the probability with which a user will choose a specific item in a given context. Based on this probability, recommendations are filtered out if the probability lies below a threshold, following which the recommendations are ranked based on how high the probability is of the user choosing that item in the given context. Again, the advantage of contextual post-filtering architectural model is that it enables use of implementation strategies used in 2D recommendation systems.

Finally, there is contextual modeling shown in Figure 5 (c), where the context is not used as a pre or post filter, but rather it is directly involved in the recommendation process as an independent variable affecting the user's rating for an item. This type of approach gives rise to true multidimensional recommendation models. These models can be implemented using heuristic or model based approaches, built by using decision trees, multidimensional regression models, and other such predictive models. These models tend to be extremely complex as the number of model parameters that need to be learned using training data grow exponentially with the number of contextual attributes. However, as long as the contextual dimensions are manageable, using this approach may be fruitful in some cases, as it may better fit the real life recommendation process [13].

We have seen three different approaches to incorporating context in the recommendation process. The first two – contextual pre/post filtering are easy to implement, as they allow for the problem to be reduced to 2D, which permits the use of traditional recommender system strategies such as collaborating filtering, content-based filtering or a hybrid approach combining the two. The last approach of contextual modeling is a bit more complicated, requiring novel statistical models to predict the ratings a user would give to items. The decision of which approach to use depends on experimental design, complexity of contextual features, and the nature of the recommendation process. So far we have talked about how to compute results on the back-end of a recommender system, the details of which are hidden from the user. However aside from listing the recommendations on the front-end to the user, it is also helpful to use visual means to convey characteristics about recommendations to users, so that they are equipped with more information with which they can make their decisions.

# 4. IMPLEMENTING A SENTIMENT ANALYSIS VISUALIZATION TOOL

## 4.1 Motivation

Often the information simplifying tools used by people are a black box to them – there exists no transparency in how the results are achieved. This is true for question answering systems such as AskMSR and Siri, or recommender systems such as that of Netflix. Making such systems more transparent, by visually depicting results in an innovative way will not only (a) enhance the confidence of users in the system, but also (b) give them more knowledge about the items and (c) how these items have been doing over a timeline of month, days, or years.

One way to do so is to create a sentiment analysis visualization tool that compliments recommender systems. There are numerous applications for such a visualization tool. For example, Yelp helps recommend services and restaurants to its customers. Along with recommendations, if a graph of how the sentiment of all the users changed over time for the recommended items was presented to the users, it would convey not only the logic behind the results, but also provide users with insight into what other people are thinking about a

given item. Thus I created an interactive sentiment visualization tool that depicts sentiment of topics or products over time, where the granularity of time can be changed as per the user's preferences.

### 4.2 Data

The dataset has been created by Niek J. Sanders for testing and training Twitter sentiment analysis algorithms. The data consists of 5513 hand-classified tweets that are subdivided into 4 topics as shown in the breakdown of data in Figure 6. Each of the 5513 entries contains:

- Tweet id
- Tweet text
- Tweet creation date
- Topic used for sentiment
- Sentiment label: positive, negative, neutral, or irrelevant

The positive and negative labels are self-explanatory. The neutral label if given to tweets that have either mixed positive or negative indicators, or have neither. It is also given to tweets that are simple factual statements, tweets that are on topic but the sentiment is undeterminable, and those that are questions with no strong emotions indicated. Irrelevant label is given to tweets that are either not in the English language (to simplify the implementation of this tool), or not on topic (i.e. spam) [14].

| Topic | # Positive | # Neutral | # Negative | # Irrelevant | Twitter Search Term |
|---|---|---|---|---|---|
| Apple | 191 | 581 | 377 | 164 | @apple |
| Google | 218 | 604 | 61 | 498 | #google |
| Microsoft | 93 | 671 | 138 | 513 | #microsoft |
| Twitter | 68 | 647 | 78 | 611 | #twitter |

FIGURE 6
BREAKDOWN OF TOPICS AND DATA

### 4.3 Feature Extraction

Each tweet was converted to an array of features, so that I can easily run various statistical models to test and compare the different models. There were 50 features used, and these test features were represented as a list of tuples where each entry looked something like this:

- ('hasAddict', ('addict'))
- ('hasExpense', ('expense','expensive'))
- ('hasHappy', (' happy',' happi'))
- ('hasWin', (' win ','winner','winning'))

Before extracting features from tweets, all tweets with the irrelevant label were labeled as neutral, to simplify sentiment classification. Following that, each tweet was converted into a tuple *(a, b)* where *a* was a dictionary with another tuple (*c, d). c* contained the feature key such as 'hasAddict' or 'hasWin' and *d* contained true or false depending on whether the tweet had words characteristic of that feature. For example, if a tweet had the word 'expense', then its value for (*c, d)* would be *('hasExpense', True). b* denotes the sentiment of the tweet. The sentiment of the tweet was given the following numerical mapping:

- Positive → 1
- Negative → -1
- Neutral → 0

After this processing, the tweets are randomly shuffled and divided into two subsets – a training set and a testing set. Then I ran a couple of different classifiers to see which one produced the best results. Running a naïve bayes classifier yielded an accuracy of 80% on average, whereas running a decision tree classifier from the NLTK library only yielded an accuracy of 73.7% on average. Multiple runs confirmed that naïve bayes performed better than decision trees. This is because simple decision trees tend to over fit the training data, which means that one has to employ sophisticated tree pruning techniques in order to improve the model. A confusion matrix showing the results in depicted in Figure 7.

```
          |             n      p |
          |      n      e      o |
          |      e      g      s |
          |      u      a      i |
          |      t      t      t |
          |      r      i      i |
          |      a      v      v |
          |      l      e      e |
----------+----------------------+
 neutral  | <76.0%>  1.9%   1.3%  |
negative  |   6.9%  <3.3%>  0.3%  |
positive  |   8.6%   0.3%  <1.5%>|
----------+----------------------+
(row = reference; col = test)
```

FIGURE 7
BREAKDOWN OF RESULTS

To gain a little more insight into the results, one can see the list of the most informative features used in any particular run through NLTK tools. Figure 8 shows a list of such informative features for a particular run.

```
Most Informative Features
             hasCrash = True           negati : neutra =     37.2 : 1.0
               hasFix = True           negati : neutra =     30.2 : 1.0
             hasIssue = True           negati : neutra =     30.2 : 1.0
           hasTrouble = True           negati : neutra =     24.1 : 1.0
              hasNice = True           positi : neutra =     21.2 : 1.0
              hasSlow = True           negati : neutra =     19.7 : 1.0
            hasBroken = True           negati : neutra =     19.7 : 1.0
           hasExpense = True           negati : neutra =     15.3 : 1.0
           hasSerious = True           negati : neutra =     14.4 : 1.0
              hasSuck = True           negati : neutra =     14.4 : 1.0
```

FIGURE 8
MOST INFORMATIVE FEATURES FOR A PARTICULAR RUN

I also experimented with implementing principle component analysis (PCA) reduction before running the above two classifiers. PCA linearly transforms the input space into an equal number of independent components in a manner that maximizes the variation explained by the first K components. In this case PCA failed to improve the accuracy, and actually resulted in a lower accuracy of 67%.

The predictions from the naïve bayes classifier were then fed into an R program where a server application was created that showed the sentiment over time for the four topics present in the dataset, depending on which topic was chosen from the interactive UI dropdown menu. The following four figures show the sentiment-time graph for the four topics in the dataset used. The results for Twitter and Google are strange due to the limited number of data records present. Twitter data for only given for a two hour window, out of which most of them were neutral, which is why the graph shows 0 as the sentiment from 2:00 to 3:00 am.

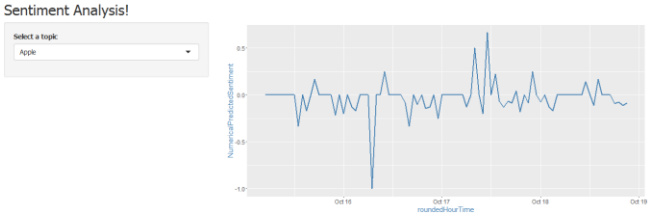


FIGURE 9
APPLE'S SENTIMENT OVER TIME

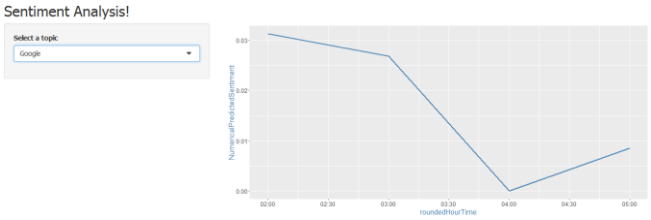


FIGURE 10
GOOGLE'S SENTIMENT OVER TIME

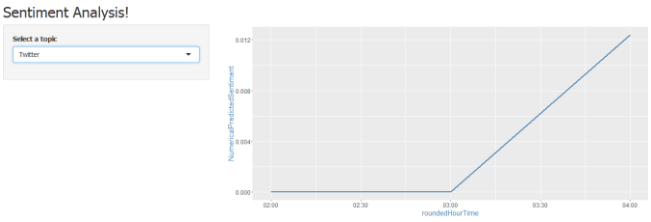


FIGURE 11
TWITTER'S SENTIMENT OVER TIME

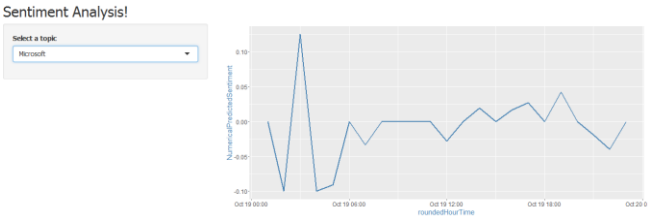


FIGURE 12
MICROSOFT'S SENTIMENT OVER TIME

## 5. CONCLUSIONS

This paper gave a literature review of question answering systems and context aware recommender systems. It analyzed the beginnings of early QA systems, and the weaknesses of early systems that prevented them from being adapted to widespread use. As mentioned in the paper, a rise in computational linguistics led a shift from closed-domain QA systems to modern day open-domain systems. Learning about the reasons behind this shift, and how the QA systems have transformed over time, gives a glimpse into the characteristics that lead to successful information handling tools such as CARS. These QA systems branched off to a field called recommender systems, the use of which has proliferated given the phenomenon of choice overload for an average consumer.

The use of context in the recommendation process is becoming increasingly crucial as computing devices are becoming ubiquitous. This has spawned a demand for context aware recommender systems. The paper provided 3 state of the art architectural models that are used currently to incorporate context into the recommendation process. However current CARS fail to pay much attention to visualization tools to help consumer digest information. The tool created through his research can be generalized to fulfill practical applications such as viewing sentiments of recommended movies on Netflix, or of restaurants on Yelp to give enhanced insight to consumers.

Future work in this area will pertain to achieving higher accuracy associated with context driven predictions of user preferences. CARS is still a relatively new research area, having many unexplored areas and topics such as coming up with optimal architectural models to better incorporate context.

## 6. ACKNOWLEDGEMENTS

I want to thank my research advisor, Professor Yanjun Qi, for assisting me and guiding me with my research project.

## 7. REFERENCES

[1] To the Moon and Back on 4KB of Memory. (2014, July 24). Retrieved from http://www.metroweekly.com/2014/07/to-the-moon-and-back-on-4kb-of-memory/

[2] D. A. Keim, "Information visualization and visual data mining," in IEEE Transactions on Visualization and Computer Graphics, vol. 8, no. 1, pp. 1-8, Jan/Mar 2002.
[3] Simmons, R. F. (1970). Natural language question-answering systems: 1969.Communications of the ACM, 13(1), 15-30.
[4] Green Jr, B. F., Wolf, A. K., Chomsky, C., & Laughery, K. (1961, May). Baseball: an automatic question-answerer. In Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference (pp. 219-224). ACM.
[5] L. HIRSCHMAN and R. GAIZAUSKAS (2001). Natural language question answering: the view from here. Natural Language Engineering, 7, pp 275-300.
[6] Julian Kupiec. MURAX: A Robust Linguistic Approach for Question Answering Using an On-Line Encyclopedia. In Robert Korfhage, Edie M. Rasmussen, and Peter Willett, editors, Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Pittsburgh, PA, USA, June 27 - July 1, pages 181-190. ACM, 1993.
[7] Marius Pasca and Sanda Harabagiu, High Performance Question/Answering, in ACM SIGIR-2001, September 2001, New Orleans LA, pages 366-374.
[8] Xin Li and Dan Roth. 2002. Learning question classifiers. In Proceedings of the 19th international conference on Computational linguistics - Volume 1 (COLING '02), Vol. 1.
[9] Barry Schwartz. 2015. The Paradox of Choice: Why More Is Less. Harper Perennial, New York, NY.
[10] Carlos A. Gomez-Uribe and Neil Hunt. 2015. The Netflix Recommender System: Algorithms, Business Value, and Innovation. ACM Trans. Manage. Inf. Syst. 6, 4, Article 13 (December 2015), 19 pages.
[11] C. Palmisano, A. Tuzhilin, and M. Gorgoglione. Using context to improve predictive modeling of customers in personalization applications. IEEE Transactions on Knowledge and Data Engineering, 20(11):1535–1549, 2008.
[12] Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). Recommender systems survey. Knowledge-Based Systems, 46, 109-132.
[13] Anand, M., & Rana, C. Paradigms for Incorporating Context in CARS.
[14] Sanders Analytics - Twitter Sentiment Corpus. (n.d.). Retrieved May 3, 2016, from http://www.sananalytics.com/lab/twitter-sentiment/