CS6109 - Compiler Design

Project Documentation

Predicting and Classifying Interests of Users Based on Direct Interactions

Team - 3

Vivek Ramkumar, 2018103082

Kariketi Tharun Reddy, 2018103034

G. R. Srikanth, 2018103603

Introduction:

The basis of our project is to provide a textual analysis of the interactions between multiple users, and predict their interests, as well as categorize them.

This is an important function of recommendation systems, that is, to offer a personalized experience for each user, by understanding their preferences.

Direct interactions between users will be subject to lexical, syntax and semantic analysis, from which we will obtain their preferences to items, as well as actions and hobbies.

List of Modules:

We will employ the standard principles of compiler design, up to intermediate code generation. Lexical, syntax and semantic analysis will be used in the following order to analyse the input text:

1. Identify tokens (cricket, leather bag, smart phone) and verbs in the continuous sense (e.g. eating, playing) and such, using lexical analysis.
2. Obtain the sentiment of the users to find out whether they like the item or not, and show the dependency trees for each dialogue.
3. Getting a long, unorganized list of possible recommendations based on the users' interests, without sentiment considerations.
4. Recommending the products to the users with respect to sentiment, and recommending alternatives based on the interests of friends to users with negative sentiments.

The result will be an organized collection of information obtained from the interaction, showing the preferences and their respective categories.

Literature Survey:

No	Publication	Author, Year	Methodology	Advantages	Limitations
1.	Short systematic review on e-learning recommender systems	W.M.Chugtai, A.B.Selama, Imran Ghani (2013)	Content-Based Filtering, Hybrid-Based Filtering, Knowledge-Based Filtering	Useful insight on the cold-start issue in e-learning recommender systems.	Limited to four types of filtering approaches.
2.	A systematic review of scholar context-aware recommender systems	Z.D.Champiri, S.R.Shahamirib, S.S.B.Salima (2015)	Kitchenham Systematic Review Methodology	Categorises recommendations into user's context, document context and env. context.	Requires tracking and analyzing previous students' interactions.
3.	Recommender Systems: Review	Akshita.J, Smita.A (2013)	Content-Based Filtering	Discusses various techniques.	Requires combination with hybrid recommenders.
4.	A literature review and classifications of recommender systems research	D.H.Park, H.K.Kim, I.Y.Choi, J.K.Kim (2012)	Data Mining Techniques (K-Means, Link Analysis, etc.)	Provides insight about future trends in the field of recommender systems.	Limited to eight types of data mining techniques.
5.	The use of machine learning algorithms in recommender systems	Ivens Portugal, Paulo Alencar, Donald Cowan (2015)	Bayesian and Decision Tree Algorithms	Identifies research opportunities for software engineering research.	Nowadays, DL algorithms surpass ML algorithms in speed and power.

Problem Statement:

We can understand that we now have pre-trained deep learning models to work with, over machine learning models.

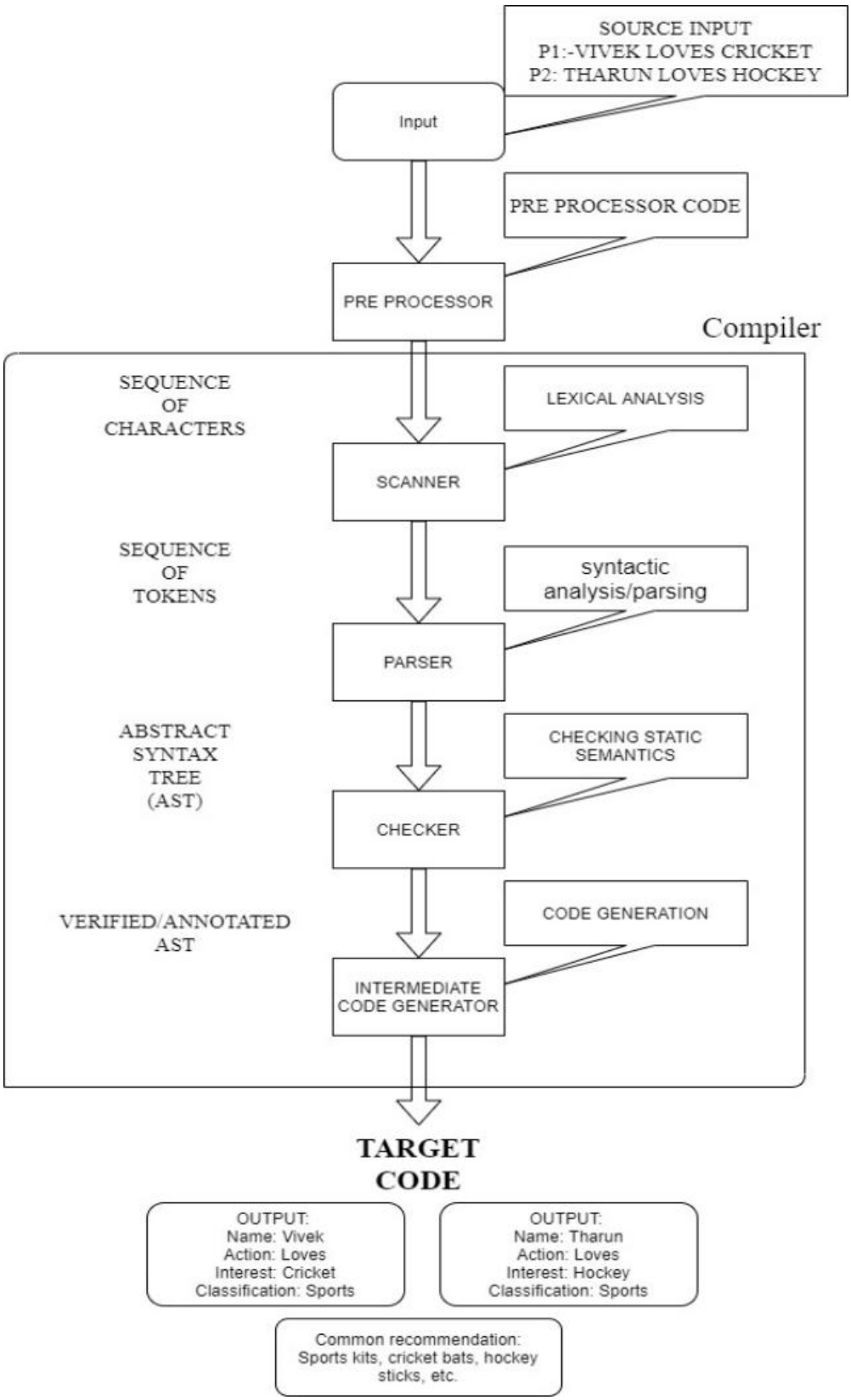
We will try to separate the tokens and tag them, whilst filtering out useless words.

The sentiment of the user's statement also needs to be considered while recommending.

To obtain the raw recommendations, we need to check for correlations between the tokens and the product names.

Organizing the recommendations and allocating them to each user will give us our final set of recommendations.

Architecture Diagram:



Pseudocode:

Module – I:

For each dialogue in conversation:

    Split dialogue in tokens and store them

    doc = nlp(dialogue)

    For each token in token_list:

        if tag == 'NOUN', add to noun_list

        else if tag == 'VERB' add to verb_list

    End for loop...

    #Using Spacy's functions, we can identify entities.

    For entity in doc.ents:

        Append entity to users

        Append entity_lab to user_types

    End for loop...

End for loop....

#Remove any leftover stop words in the nouns list

nouns = [word for word in nouns if not word in stopwords.words()]

Store all tokens in a data frame, and return the data frame

Module – II:

#Obtain sentiments using the NLTK Vader Sentiment Analyzer

Create a data frame to store sentiment values

sid = SentimentIntensityAnalyzer()

For each dialogue in conversation:

    Calculate polarity scores for each dialogue

    Append values to sentiments data frame corresponding

    to each sentiment

    Return the sentiment values data frame

End for loop....

#Displaying the dependency trees....

For each dialogue in conversation:

    #Use loaded dictionary of English words from Spacy library

    nlp = spacy.load('en_core_web_md')

    doc = nlp(dialogue)

    #Displacy is a visualization library for NLP

    displacy.render(doc, style = 'dep')

End for loop...

Module – III:

For each noun in noun_list:

For each item in complete_list:

If lower(noun) is present in lower(item name):

Append item name to temp_list

End for loop...

Append temp_list to init_rec

End for loop...

Return the list init_rec

Module – IV:

For index in the range of length of init_rec:

If sentiment for user[index] is positive:

Append random items to final_rec[index]

If sentiment for user[index] is negative:

Choose random items from other user interests

Append those to final_rec[index]

#We have chosen to recommend 5 items per user

If sentiment for user[index] is neutral:

Append random items to final_rec[index]

End for loop...

Return the list of lists final_rec

List of Modules (in detail):

Module – 1:

The main focus of lexical analysis is to obtain tokens from the given text input.

We need to differentiate between entities (users, organizations, religion, etc.) and interests (garments, tech, etc.).

The technique we are going to employ is called POS tagging. (POS -> Part of Speech)

POS tagging algorithms allocate tags to each token, such as "NOUN", "VERB", "ADJ", etc., based on the similarity to their respective tagged categories.

We will be using the Spacy library, which contains pre-trained NLP models and a huge dictionary of words.

The input is parsed sentence by sentence, from which nouns, verbs, users, and user types are obtained. The respective information is summarized in a pandas data frame.

Keep in mind the input is made to understand short conversations - especially one-liners. Basically, we separate the text blob by the '\n', or newline symbol. This is done by using the splitlines() function.

If any respective dialogue consists of two lines, the parser will be confused and allocate an extra user to the extra line.

This is done with consideration to social messaging applications, as users frequently communicate with short dialogues.

Module – II:

The main point of understanding the syntax of the sentence is to check the order by which tokens are parsed.

For example, seeing a "not" before a verb, say "enjoying", can imply a negative sentiment.

Sentiment analysis is an important tool in recommendation - it helps recommend products to those who want it.

People who express negative sentiment towards a product will not recieve recommendations on it.

Instead, they will get recommendations based on what their friends like.

The NTLK Vader Sentiment Analyzer is an excellent tool for quickly analyzing the sentiment of a sentence.

We will obtain the polarity scores in each sentiment category, i.e "Positive", "Negative", "Neutral", and "Compound", for each user.

The Spacy library consists of one more feature - the ability to show dependency trees.

Dependency trees are useful in understanding the parsing process, in pursuit of finding the sentiments.

The arrows infer dependencies between words. If an arrow points from 'a' to 'b', then it can be understood that 'a' depends on 'b'.

This is useful in seeing how the tokens are tagged, and how the sentence is syntactically analyzed.

Module – III:

We will check the list of nouns from our summary of tokens to see what the interests of the users are.

If the particular noun is present in any product name, we will store that product name in a new list.

This new list is considered as our initial set of recommendations.

Now, our initial recommendations are quite a large number.

We will pick five products at random, to recommend to each user.

Of course, we do not want to get the same products again in our random selection.

This is easily achieved by using random.sample(list, k), where k is the number of items selected. This function also ensures that there is no repetition in the selection of items.

Module – IV:

This is the final recommendation.

You can see the user's names followed by a list of recommendations based on their interests.

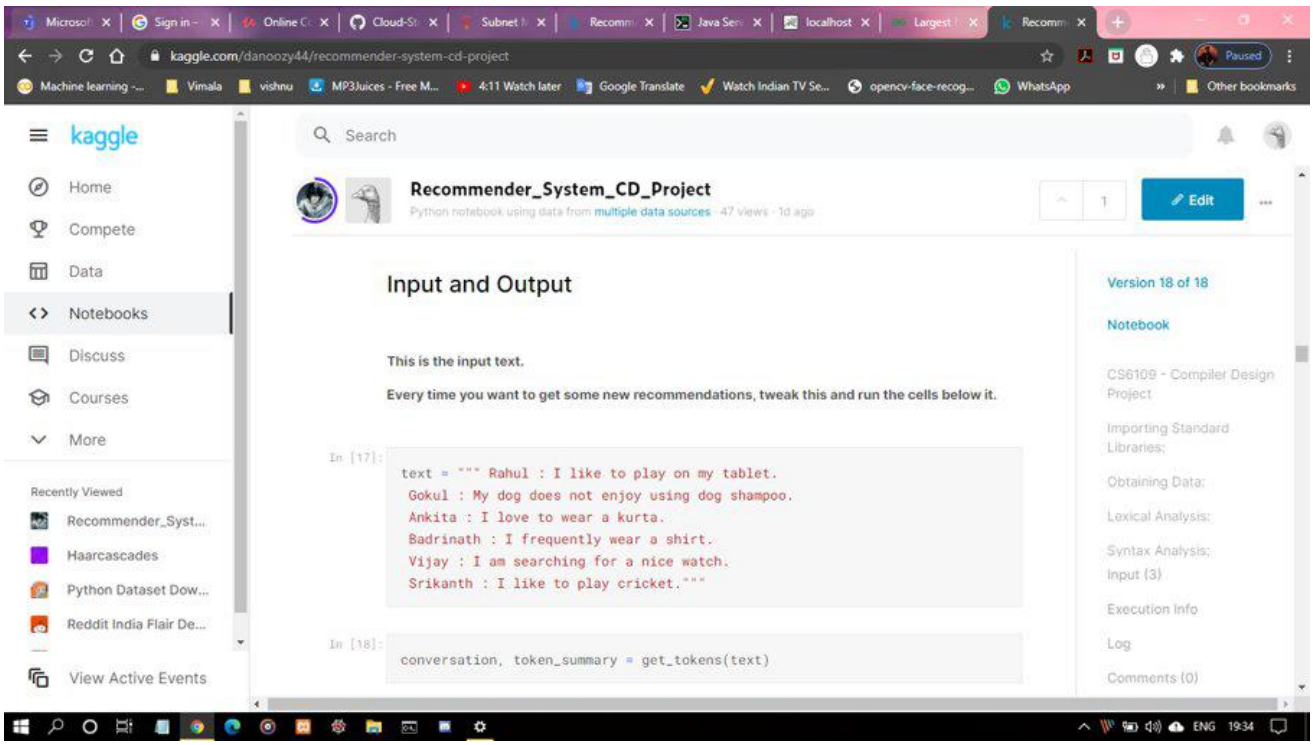
Keep in mind that for the most part the recommendations are accurate, but you may chance upon some outliers, due to the random nature of product selection.

As stated before, the sentiment of the user is considered as a factor in the recommendation process. The compounded polarity score is considered here.

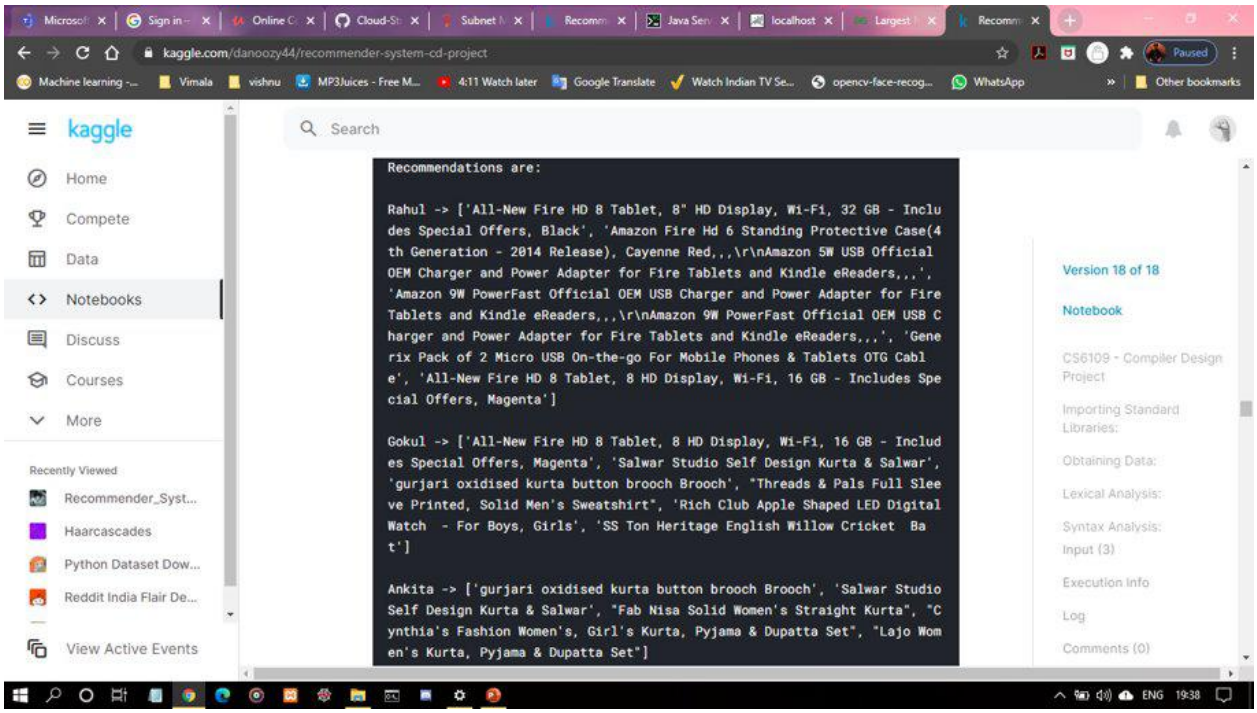
If a user's sentiment is negative, then we pick a random recommendation based on the interests of that user's friends.

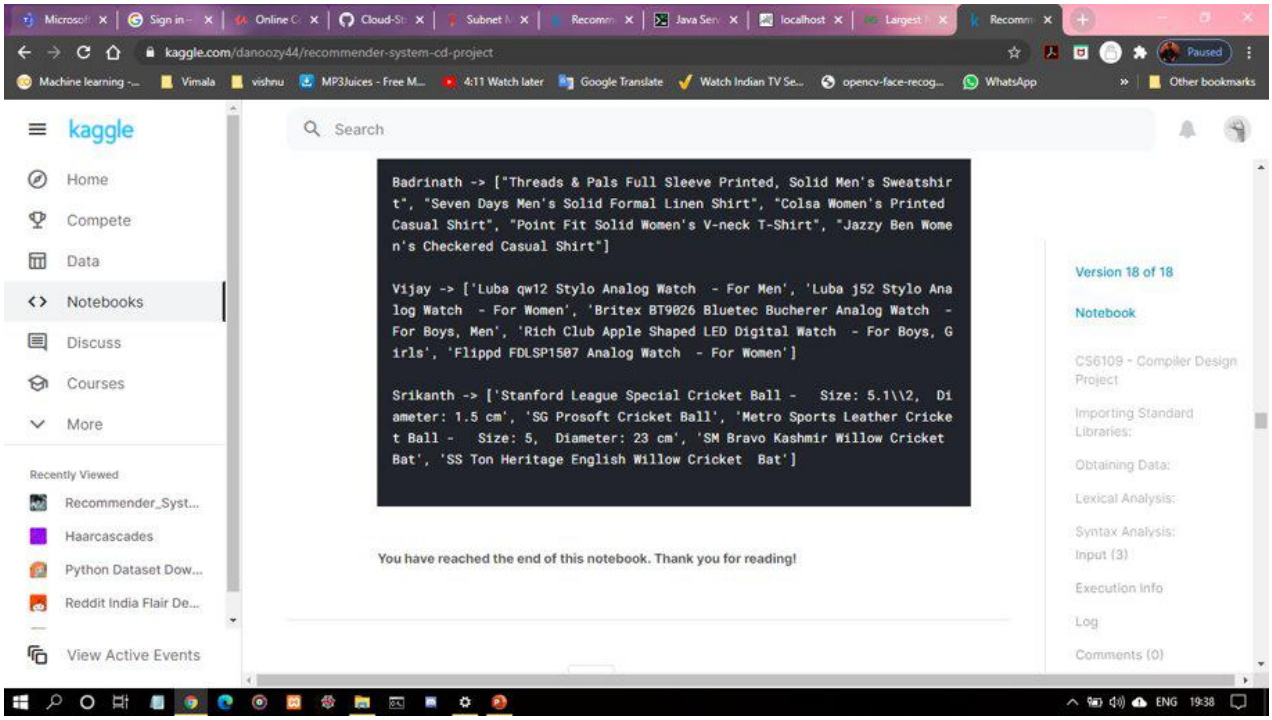
Otherwise, if it is positive or neutral, we can directly recommend based on their individual interest.

Input:



Output:





Results Discussion:

Since we've checked for subsequences of the occuring string, the nouns, inside product names, of course we cannot guarantee an accuracy of 100%.

Regardless, the recommendation system seems to works fairly well - it recommends what the users want based on their interests.

For users with negative sentiment - it doesn't recommend the products associated with the user. Rather, it recommends products associated with the user's friends.

We also ensure any product recommendation is not repeated for the same user.

The end result is fairly straight-forward. It works for most cases, with some exceptions, a rare case.

Conclusion:

The information obtained from direct interactions can be a valuable tool, as we can create recommendation systems on the basis of user-user interactions, such as comments on a forum, rather than user-site interactions.

Previous systems understand and provide recommendations based on similarities between users, which might lead to a bias evaluation for user preferences. This provides a solution for this issue by integrating direct interactions into recommendation systems.

However, there is still much left to improve, and a full accuracy of such systems is never guaranteed, but this approach will definitely up the bar in extending further recommendation systems.

References:

1. C.Li, F.Xiong, “Social Recommendations with Multiple Influence from Direct User Interactions”, Aug. 2017.

2. C. Pan, W. Li, Research paper recommendation with topic analysis in Computer Design and Applications IEEE, 4 (2010), pp. V4-264

3. J.A. Konstan, J. Riedl, Recommender Systems: From Algorithms to User Experience, User Model User-Adapt Interact, 22 (2012), pp. 101-123