Midterm report

Introduction:

The objective of this project is to implement a personalized event finder onto Amazon Alexa that will scour popular event websites for events that would interest the user. Initially, we planned to personalize results and sync them with users' calendars. However, due to data access limitations and privacy issues, we have altered the trajectory of this project to provide the best possible general recommendations of events.

1. Dataset sources: Eventbrite and Ticketmaster

- Dataset 1
- Dataset 2

2. Data analysis / EDA results: Expected outcome: Based on user query and the descriptions of the events, we hope to provide a top five list of events that the user would be interested in. Events considered would be within a five mile radius of the target location as well as within the requested time frame or within a month of the search.

TM to DF Code.ipynb

Ticketmaster dataset

Method: Web scraping

Found events under postal code 90703, 94704

Variables to consider: name','venues','pleasenote', 'type', 'genre', 'subGenre', 'info','pleaseNote','sales', 'classifications']

Results:

27 listings of which all were related to music or arts and theatre. The three main events: The Secret Garden theatre showing, Shrek the Musical, and a series of concerts/music shows would not provide the variety necessary to encompass the range of events that would be sought after on an event finder, nor does it span people's general interests. Also, these 27 listings would not be sufficient in training a model.

Word2vec.ipynb

Eventbrite dataset

To supplement our data, we looked into Eventbrite, another event source with event descriptions, ratings, and events catered towards an expanded range of interests.

Without being able to access private event attendance records and train them on individuals' reviews from Ticketmaster alone, we would not be able to personalize event findings as we initially planned to. With this limited data dilemma in mind, we adjusted our feasible, short-term expected outcome to procure a list of the top five most popular events in the area, similar to a google query. We hoped that generalizing the search results and providing multiple options would increase the likelihood of users finding an event they would be interested in.

Therefore, for the next steps, we decided to use word2vec to compare user queries to the event descriptions to provide users the top search results. Our goal was to group similar words and train them to correlate to the average rating of an event.

In order for this method to be implemented, we first needed a large batch of text data drawn from a relevant domain. After collecting the reviews, we pre-processed the file to return a list of words as tokens. The resulting data file held about 10,000 entries of 100+ words per entry, which gave us over a million words to use. From there, the tokens were trained with a neural network with a single hidden layer. The weights of the instances were then recorded to determine the degree of similarity between words.

Example with "dance":

3. System Architecture Overview

For the frontend, we're using a Flask application to handle user queries. We decided on the Flask framework because it is lightweight and faster to get started with compared to building it on our own. The frontend is basically a form which takes in a user's name, a keyword, and a message. It should return the relevancy of the message relative to the word. The technique that we're using to determine the meaning of the message is word2vec.

Working with data coming from a browser can be complicated and hard to read. That's why we decided to use the WTForms library, which makes this process much easier. We have one class ReusableForm, which gets called each time the browser opens. The data gets submitted via the POST action and gets received via the GET action.

Right now, the word2vec implementation is working and we have trained/tested it on the Eventbrite dataset. Now that the model is ready, the next step is to encapsulate the similarity matching process into a function that takes in a String parameter and returns a relevancy score. This function will be called every time a user hits the Green submit form button. Once this is done, the frontend and backend will be connected.

We have also designed the UI to have dynamic responses and alert users if their form isn't filled out. All the styling is done on hello.html.

Performance Evaluation/Challenges:

Currently our algorithm is designed for one-word queries, likened to the example above with the word "drink", where we count the number of "similar words", which are words that reach a certain level of similarity to the query. The way that our algorithm is designed, it will look for the highest rate of the occurrence of these types of words within a description. This is definitely a start for us. With some tinkering of our model, it will be able to the description with the most occurrences of words like the query word. However, considering the goal of our project, this serves mostly as a foundation for us to work upon. First off, queries are usually more than one word. We do plan to strip voice-recognized queries of words such as "the" and "a" which don't give much insight. However, we will need to adjust our algorithm to be able to take in queries that are longer than one word. Examples of such queries are "jazz music" versus just "music" as a category. There are different avenues that we could take into regard with that. Algorithms such as the bag-of-words algorithm exist, or a more rudimentary and less effective method could be to run the current algorithm for each important word in the query. Also,

The main challenges that we encountered are related to the matter of evaluation. To know how good one's model is, there is often an evaluation metric to measure how accurate that model is. However, with the way that our project is designed, it was difficult to establish what would be our form of evaluation. Our original proposal was to have Alexa be asked regarding events occuring in a certain area, and then to have it ask enough questions to be able to give recommendations. However, we were unclear of a good way to evaluate how "good" the event was. Some suggestions that came to mind were to look for the amount of people who went to similar events and to train off of that. That in itself would have been another problem to tackle, in part because a lot of the information that we deemed to be evaluation metrics were not available to us through the API. Through our considerations, we decided to lean towards the direction of using a word-to-vector algorithm, or something of the like. Also, we had a challenge in regards to the framing of our project and the framing of our model. In consideration of our project, we realized that our previous view of the pipeline of our project could have been implemented would have been without any usage of machine learning algorithms. In its most rudimentary form, it would have been as simple as just looking for events with certain words in it. Because of this we shifted the implementation of our algorithm to incorporate an aspect of training data. Something that we will also need to consider moving forward is if it will be possible to be able to tailor events to a particular user's tastes, as well as if we would want to do that. For users that utilize this Alexa feature, there is an opportunity to help cater suggested events to a user dependent on previous events that they added to their calendar. At the same time, this is something that could: 1. Prevent the user from branching out of their comfort zone to explore new types of events and 2. Be slightly invasive of their data.

Something that we will also need to consider moving forward is if it will be possible to be able to tailor events to a particular user's tastes, as well as if we would want to do that.