Midterm report

Introduction:

The initial objective of our project is to implement an event finder onto Amazon Alexa that will scour popular event websites for events based on a user profile. Initially, we planned to personalize results and sync them with users' calendars. Through feedback on our project, namely, data access limitations and privacy issues, we have altered the trajectory of this project to provide the best possible general recommendations of events. Instead, our project's goal is to find events for users to attend via Amazon Alexa based on user queries and based solely on the event description and tags, all data that is easily accessible via APIs online.

For all files please visit: https://github.com/akannan747/EventHound

For an easy to use instruction on our project:
https://github.com/akannan747/EventHound/tree/master/demo

Data Sources & Analysis

Attempt to use Ticketmaster

Even though we ultimately went with EventBrite data, we learned quite a bit about circumventing regulations to get data from Ticketmaster API. We've outlined the work we did getting data from their API here too.
https://github.com/akannan747/EventHound/blob/master/TM%20to%20DF%20Code.ipynb

Found events under postal code 90703, 94704

Variables to consider:
name','venues','pleasenote', 'type', 'genre', 'subGenre', 'info','pleaseNote','sales', 'classifications']
To get up to 1000 events we had to put a time delay of 0.21 seconds (with variability) to ensure that the requests didn't get flagged as a "spike" in traffic by their Event iterator object, which we had to iterate over to get event 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.

Our Savior, The Eventbrite API

Data Set On: https://github.com/akannan747/EventHound/blob/master/event_data4.csv

Data analysis / EDA results: Expected outcome: As our midterm deliverable, we scoped in our project to focus on the Eventbrite API. Unlike Ticketmaster, which limited event data to 1000 events and Facebook Events (which have a host of policies to thwart scrapers), Eventbrite offers open access via their API to all current events.

EventBrite API: https://github.com/akannan747/EventHound/blob/master/EventBriteHound.py

From EventBrite, the process of gathering data was far more straightforward (we wish we started with their API first!). We simply had to call to their API, iterate over URLs and over pages, and write the text under the description tag and the URL to a CSV file. We got about 10,000 event descriptions. This would be the corpus we would use to train our algorithm. We worried that 10,000 events wouldn't be sufficient (we still debate whether it was enough), but for the purpose of the midterm deliverable, it seemed to deliver enough accuracy to pass. Perhaps with more time in the future, we can merge it with TicketMaster or other event data to increase the size of our corpus.

Building Our Model

https://github.com/akannan747/EventHound/blob/master/word2vec.ipynb

As we had alluded to at the start of our project, we wanted to train our algorithm with some target variable in mind, such as event attendance or user preference. However, as aspiring data scientists, it seemed the data we had in hand with the EventBrite data only provided us details about the event. In spite of us attempting to find work arounds, find records about event attendance, or even playing with the idea of labeling data for "good" or "bad" by hand, we ultimately realized we had to do some unsupervised training with our event descriptions.

Without being able to access private event attendance records and train them on individuals' reviews from EventBrite 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. We noticed that many events on the EventBrite data lacked information on tags or other identifiers that would help users find those events. This seemed like a dilemma we could solve using unsupervised learning.

And so, we shifted focus from personalizing results to trying to find results for the user based completely on event descriptions.

We wanted to build a recommendation algorithm and explored the numerous studies and literature available on the subject, including briefly looking into the Stanford CS 224N video course on Natural Language Processing. We are highly indebted to the knowledge we got from these videos on unsupervised and recommendation tasks!

For the next steps, after much debate, we ultimately 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 from EventBrite, 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.

Here's a code snippet that shows the power of the Word2Vec model trained on our corpus.
https://github.com/akannan747/EventHound/blob/master/dance%20example.PNG
By calling for most_similar words to a given `query` one is able to see the words in event descriptions that are most related to it according to W2V.

We originally planned after building the W2V model to iterate over every word and take the average similarity for a description and display the top similarity score. However, we found in practice this was very vulnerable to descriptions that read repetitively like:

"drinks drinks drinks drinks drinks drinks drinks drinks drinks drinks drinks drinks drinks drinks drinks drinks drinks drinks drinks drinks, Vermont Vermont Vermont Vermont Vermont Vermont Vermont Vermont Vermont Vermont Vermont Vermont, wine wine wine wine wine wine wine wine wine wine wine wine wine"

We experimented with several attempts at a scoring system and ultimately built one that required the most similar words to the search word to appear in the description and multiply that by a repetition decay weighted (1/number of repetitions), divided by length of description. This proved to be the most successful, though not perfect, fit based on the similarity scores we could get from the Word2Vec algorithm.

Prototype
Instructions to Run on:
https://github.com/akannan747/EventHound/Demo

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 the tinkering we did, our model was able to do surprisingly well at at least in the top 3-5 results display a very good fit event based on the single word query, even for some more obscure queries as long as they were words in the corpus. We lack a way to quantify how

well our model performs (it’s unsupervised and we don’t have a truth for whether the recommendations are good) but our own use of the model seems to suggest mediocre (neither amazing, nor poor) performance and sometimes trips up on providing useful recommendations. 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. We could consider the usage of the Bag of Words assumption along with more sophisticated network models for making event recommendations.

We also feel an important criteria is the speed of our algorithm. At the moment, it takes quite a bit of time to load up. We feel that if we were able to get this algorithm to run on a website, perhaps we could train the model prior to load up and just load up the model weights and vector encoding on website load. Perhaps we could even experiment with reducing the size of our corpus or trying more efficient data iteration or storage techniques. These are all areas we would like to look into and track for our final deliverable. At the moment, on the Mac Computer on which this was written, it takes about 1:12 for the model to train and load. If we were able to “slice” our data and only run our scoring algorithm on less data that met user criteria (i.e. near me or after 9pm), we could cut down on query to result time as well.

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. The idea that we are considering pursuing is on deployment, tracking which event of the ones that are displayed, are clicked by the user and using that to retrain the global model progressively.

System Architecture & User Interface

1. The system is build on gathering data from the EventBrite API https://github.com/akannan747/EventHound/blob/master/EventBriteHound.py
2. This data is then converted into a CSV file

3. The CSV file is read by the form.py, trains the model on load

4. It then runs the user interface to gather the query data from the user (the only important data for our algorithm are "Enter the query word" and "How many results do you want to see?")

   See User Interface
   STEP 1: In command line, execute "python form.py".2. Click OK in the first welcome window.
   STEP 2: Click OK in the first welcome window.
   STEP 3: Enter your name and click OK to continue.
   STEP 4: Enter your search query (1 word!) and click OK to continue.
   STEP 5: Enter the number of results you would like to see and click OK to continue.
   STEP 6: The algorithm will run.
   STEP 7: Once it is done running, it will display the top X event links in a new window.
   STEP 8: You can click on the event links and it will open in the browser.

5. It iterates over all the events in the corpus to compute the description's similarity to the query
   https://github.com/akannan747/EventHound/blob/master/word2vec.ipynb
   This is an easy to trace version that's separated from our User Interface and exists solely on Notebooks (not connected to UI)!

6. The top (user input, default value = 5) number of event's with the highest similarities to our query are displayed to the user to click on.

Conclusion

In sum, our project transitioned to a goal of using descriptions from events to match them to a user query. We implemented a Word2Vec model that was trained on a corpus of 10,000 event descriptions we got from the EventBrite API. The similarity scores of words it outputs allowed us to develop a convoluted scoring system based on weighted summing of similarities, discounted by repetition and length, weighted toward extremely similar words. On a TKInter interface, we gather user input and display the top search results for them on this interface, which can be executed as .py file. In future iterations of our model we would like to consider the following list. We've coded the difficulty of the tasks [Easy: E, Medium: M, Hard: H]

- **H**: Using more than single vector representations for similarity of words to paragraphs (perhaps, Doc2Vec has some promise there)
- **M**: Improve the speed of training, or jointly, train the model prior to user loading
- **M**: Parse important words (Key term extraction) and allow for greater than single word inputs
- **M**: Based on important words, cut the data frame prior to running the similarity comparison algorithm to further cut down on runtime.
- **E**: Create a nicer user interface that sits on a desktop
- **H**: Create a feedback model based on the event the user selects to continuously train the model.

Thank you for reading through our report, and we look forward to delivering an even higher quality, more refined product for our final deliverable after this initial midterm report.

Works Used for Inspiration

https://github.com/akannan747/EventHound/blob/master/sources.txt

Thank you particularly to our team advisor, Professor Ojala.