Event Recommender Using Text Classification

Deepak Krishnan
School of Information, MSI Candidate
dkdeepak@umich.edu

Andrew Lin
School of Information, MSI Candidate
ljumsi@umich.edu

Kate Barron
School of Information, MSI Candidate
katebar@umich.edu

Omkar Sunkersett
School of Information, MSI Candidate
osunkers@umich.edu

ABSTRACT

In this paper, we outline the processes, methods, and results of a text classification program we created to classify events at Universities in certain predefined categories.

1 INTRODUCTION

The University of Michigan--Ann Arbor posts campus events on events.umich.edu. Every day, there are 80 events to choose from on average. Events are grouped into approximately 20 "event types" (e.g. Exhibition, Performance, Ceremony/Service, etc.) and tagged from among hundreds of subjects (e.g. Fitness, Books, Games, South Asia, etc.). Users can browse or keyword search to find events of interest; they can also filter results by event type or tag. However, our team feels the current system architecture does not effectively link users to event information. With a large number of events, event types, and tags to choose from, users undoubtedly retrieve events in which they are disinterested while missing many others of interest. Even when users isolate an event of interest, they must then manually compare the event date/time against their calendar. In combination, these factors defeat the purpose of the events.umich.edu page: to conveniently connect users with events for their enrichment or entertainment. Unlike formally teaching or learning in the university community, extracurricular event choice should be simple and fast.

To this end, our team has developed an event recommendation application for the University of Michigan--Ann Arbor. We have distilled event types and tags into 8 categories: Academics, Arts, Career, Workshops, Sports & Health, Community Service, Global Learning and Other. Using these categories, our team trained an algorithm to classify events based on their textual descriptions. We then applied the classifier to contemporary events. Finally, we developed a graphical user interface using Python's TKinter package. Through the graphical user interface and by providing access to their Google calendar (we used the Google Calendar API to filter out events conflicting with the user's schedule), users can select categories of interest and receive daily event recommendations compatible with their schedules.

2 EXPERIMENTAL AND COMPUTATIONAL DETAILS

2.1 Forming the Dataset

We wanted to capture a set of all events from the Michigan website events page. We used the json feed from events.umich.edu to fetch all events from Jan 1, 2014 until Dec 31st 2016. We did this by employing a python web crawler to fetch the data. Events dating back to 2014 were fetched in order to increase the amount of training data available to us. Overall, we retrieved approximately 32,000 records. We fetched the event title, description, start-date, end-date, start-time, end-time, and the first-listed tag. This information was then written into a CSV file. We did some manipulation of the data from the CSV file. We had a list of all tags from the website. These tags were condensed into 8 unique categories which we have explained in the introduction. We iterated through the list of tags for each event in the CSV file and replaced it with our categories. We combined the title and description fields into one string. We also created a Pandas dataframe which contained this combination of title and description and the equivalent tag according to our categorization.

2.2 Developing the Classifier

We considered many different types of classifiers. Some of the popular ones we took into account were Naive Bayes, LinearSVC, Random Forest and Extra Trees[1]. The final one we considered was the XGBoost classifier. Based on our research, XGBoost is the classifier which most data scientists are adopting to get better results[2]. We however had challenges in implementing sklearn. These challenges were mitigated when we tried using the other classifiers because we got very good results. When we had the Naive Bayes, our results were pretty poor. We only achieved an accuracy rate of 60%. We then switched on to using Random Forest which was the beginning of our phase of achieving better results. Random Forest pushed our scores beyond 80%. We understood that this is a good rate, however, we wanted to see if we could better this. So, we went on to Extra Trees which gave us pretty much the same accuracy. Our final payoff occurred when we used LinearSVC. This method, especially with a c-value of 1.03 pushed us beyond 90% and indicated to us that we had created a successful product. Our team used the scikit-learn package provided by Python to do this. Along with the sklearn module, we employed a LinearSVC model to train our dataset. The LinearSVC model works by using what is known as a Tf-Idf vectorizer, and removing the stopwords. We also manually removed all punctuation to improve the level of accuracy. We used the train_test_split function offered by sklearn to split the training data into 80% of training and 20% of test data. We then applied the learning model on the training data and used it to predict the categories of the test data. Our tests gave us a 90% accuracy rate. The end-product of the classifier after it is trained would be to apply the classifier to new data and provide categories for the events. The new data input required would be just the string format of title and description combination. The output would be labels which correspond to which of the 8 categories an event belongs to.

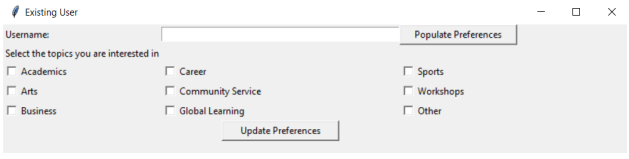

Fig.1.1 Screenshot of application for existing user displaying choices

2.3 Developing Front-End Interface (with Google API)

The front end was developed using the Tkinter module in Python. Tkinter gives a variety of GUI objects one can use such as buttons, labels, textboxes and checkboxes, all of which have been used in our project. We have a login page, where a user can login using their university uniqname. Once logged in, users would be asked to choose the categories they are interested in receiving event updates of. These choices are provided in the form of textboxes in Tkinter. These choice could be updated at any point of time. The information of user choices is stored in a SQLite database. Once users login, they are given two options: register as a new user or view events. Registering as a new user allows them to make their choices on which events they wish to see in their feed. The second page is the user events feed page. This would refer to the classifier and pull out those events the user is interested in viewing, which as mentioned before is captured in the SQLite database. We have also used the Google API[3] to fetch a user's calendar and find out those times when a user is free based on his or her schedule.

3 RESULTS AND DISCUSSION

The results are evaluated by accuracy rate. We have separated a testing data set from the training set by random selection. The ratio is 8:2. The accuracy rate of the tests are as follows:

Table 1: Comparison of Accuracy Rates from Models

Model	Accuracy Rate (%)
Random Forest	82
Extra Trees	84
Support Vector Machine (linear kernel)	90
Naive Bayes	62

The best performing one, as we can observe, is the Linear SVM model. One reason it performs better is that the dataset is relatively clean and groups are distributed smoothly. The number of outliers are expected to be few, allowing hyperplanes to separate the classes clean in the support vector machines model.

To further fine-tune the model, we have tried different parameters for the model to achieve the optimal result. We

have used a range of the penalty parameter C to find the following results:

Table 2: Comparison of Accuracy Rates from Penalty Parameter in SVM

Penalty Parameter (C)	Accuracy Rate (%)
0.95	86.08
0.96	86.08
0.97	86.13
0.98	86.9
0.99	87.21
1	88.15
1.01	89.34
1.02	89.87
1.03	90.05
1.04	90.02

As we probe around the default value of 1, we find that the accuracy rate increases with C. At C = 1.03, a local maximum is reached. We finalize our model with a SVM model with C = 1.03. The accuracy rate on the testing dataset is 90.02%.

The efficiency of the model is moderate. It takes 213 seconds to train the data on our worst computer. In comparison, the prediction time is trivial, since only events in one day fit in at a time. Still, because the classifier needs to be trained every time it runs, the efficiency of the end-product has a bottleneck.

We also discovered that many events fit into multiple categories. Consider the event "Asian Tradition Meets World Fusion with the Orchid Ensemble":

Established in 1997, the JUNO nominated Orchid Ensemble is Lan Tung (Taiwan/Canada) on the erhu/Chinese violin, Yu-Chen Wang (Taiwan/US) on the zheng/Chinese zither, and Jonathan Bernard (Canada) on percussion. The ensemble blends ancient musical instruments and traditions from China and beyond and has embraced a variety of musical including traditional and contemporary Chinese music, World Music, New Music, Jazz, and Creative Improvisation. Acclaimed as "One of the brightest blossoms on the world music scene" (Georgia Straight), the Orchid Ensemble has been tirelessly developing an innovative musical genre based on the cultural exchange between Western and Asian musicians.

Based on this description, the event could be categorized as "Arts and Global Learning". Right now, our system forces us to choose one category over the other (in this case, we selected "Arts"). An ideal system would categorize events with all relevant categories, establishing the strength of each category (i.e. the first listed category has the strongest influence on retrieval, the next listed category has the next strongest influence, etc.).

4 CONCLUSIONS

In retrospect, we find there is room to improve our categories. We developed our categories before having pored through the data; as such, we had to group a fair number of events as "Other." Further, our categories mixed event subjects and types (i.e. Workshops and Community Service vs. Academics and Arts). Because these categories existed on different axes, it was sometimes difficult to choose from among them (e.g. some Arts events are also Workshops). Now we better understand why the events.umich.edu made distinctions between type and tag filters. In the next iteration of our application, we can distill event types and tags separately, and include them both for user selection.

In developing the next version of our application, it would also be good to build a codex for classifying event types and tags. The would-be codex could benefit from test user input--that is to say, interviewing users to make sure we correctly understand their notions of the categories. By using a codex, our team could more accurately/consistently label the train/test data fed to our classifier, and improve performance.

Another direction we could take in the future is to incorporate new events into the training to improve the model, making it 'learn' by the day. What need to be solved is the increasing amount of time and space required. Some form of caching may be needed for the historical events.

5. REFERENCES

[1] Text classification and Naive Bayes
Text classification and Naive Bayes. (2017).
nlp.stanford.edu Retrieved 23 April 2017, from
https://nlp.stanford.edu/IR-book/html/htmledition/text-classification-and-naive-bayes-1.html

[2] Python), C., Python), C., & Jain, A. (2016). Complete Guide to Parameter Tuning in XGBoost (with codes in Python). Analytics Vidhya. Retrieved 23 April 2017, from https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

[3] API Reference. (2017). Google Calendar API. Retrieved 23 April 2017, from https://developers.google.com/google-apps/calendar/v3/reference