Using Search History to Classify Users

COMP 3106

Joshua Hills 101142996

November 1, 2021

1 Introduction

I had an upcoming interview for a cybersecurity internship. Other than it being my major, I don't know much about cybersecurity outside of the basics. I used this project as an excuse to make something that I could share with my interviewer on how we can innovate the field of cybersecurity with AI. Is it possible to identify a user's identity based solely on their search history? Search engines like google collect as much information as possible about a user and can indentify a user if they are logged in. But what if the user is browsing anonymously or on someone else's computer? (They should be definetly treated as anonymous, but with how much I value privacy aside, this is the purpose of my project).

There are not many examples of people exploring search data. Usually this is looked at as private. Some of the quotes I heard while asking friends and family for their data "I'd rather just have my history between me and google." and "Are you going to know which is mine?". People don't want to share their data and companies can't openly research it so it makes it a relatively untapped field of research. The confidentiality that people want is likely due to how much search history can say about us, like where we're going, what we want to be, what we want to learn about.

2 Methods

In the following subsections I will address the data collection method, data used, methods of artificial intelligence used, validation.

2.1 Data Collection

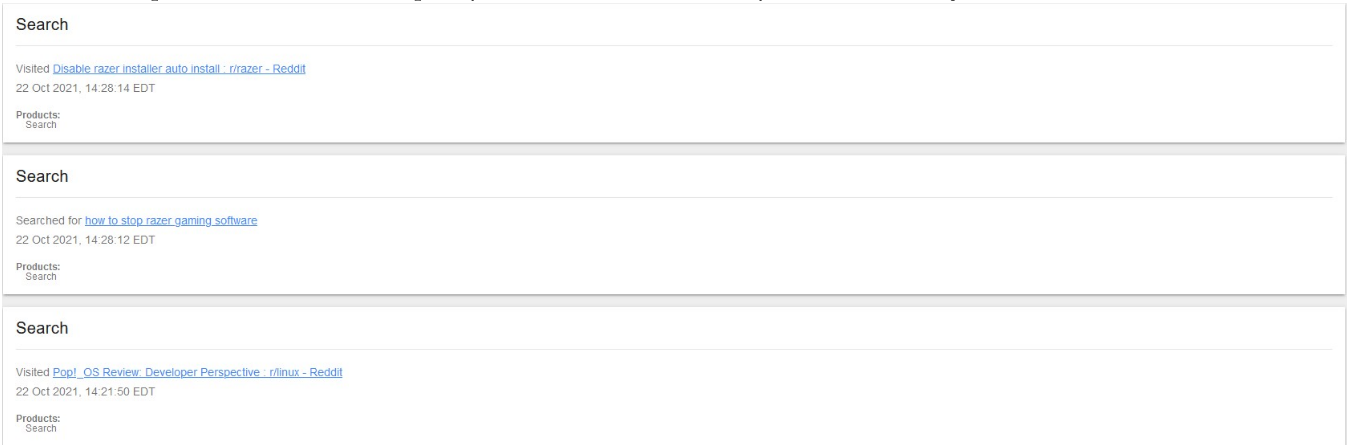
I was able to collect google search histories for 11 people including myself. Any screenshots showing search terms are strictly mine. I am respecting the privacy of the peole who shared their data with me. I wrote a guide for my friends to download their histories and share them with me. It was far more structured than the following summary, regardless. To summerize:

1. Go to takeout.google.com, this is where google allows it's users download and see what data they have collected about them.
2. Change your selected exports to just get search data.
3. Once google sends you the export, upload it anonymously through google forms.

With that 10 of my closest friends and family shared their data with me.

2.2 Data Used and Data Exploration

The data provided to me is in a pretty weird format. It's solely in html looking like this:


Search

Visited Disable razer installer auto install : r/razer - Reddit
22 Oct 2021, 14:28:14 EDT

Products:
Search

Search

Searched for how to stop razer gaming software
22 Oct 2021, 14:28:12 EDT

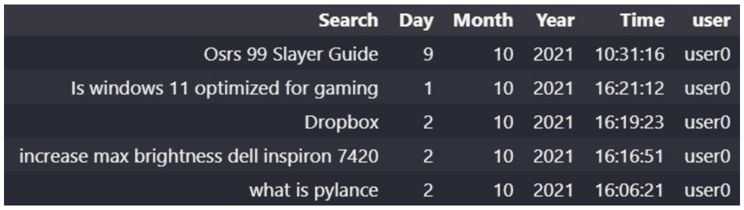
Products:
Search

Search

Visited Pop!_OS Review: Developer Perspective : r/linux - Reddit
22 Oct 2021, 14:21:50 EDT

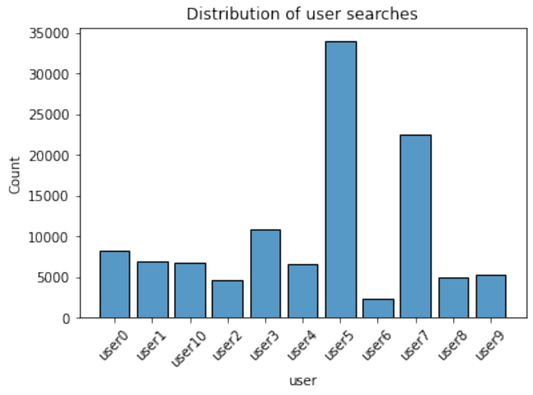
Products:
Search

I had to parse through this html file to get to the useful data. Using a python package called BeautifulSoup I iterated through each of these cells collecting the search query, day/month/year/timestamp of search. The dataframe looked like this after parsing:



Search	Day	Month	Year	Time	user
Osrs 99 Slayer Guide	9	10	2021	10:31:16	user0
Is windows 11 optimized for gaming	1	10	2021	16:21:12	user0
Dropbox	2	10	2021	16:19:23	user0
increase max brightness dell inspiron 7420	2	10	2021	16:16:51	user0
what is pylance	2	10	2021	16:06:21	user0

Note the user column, this is what we will be using as the target. Looking at the data we can see what % of searches belong to each user.



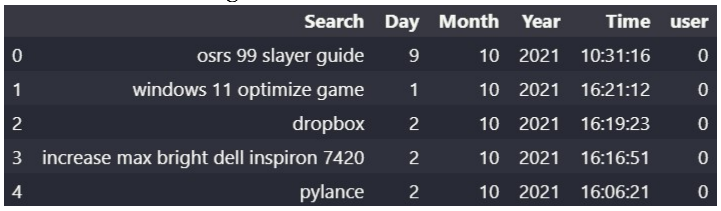
This data still isn't ready to be fed into a model, which takes us to the next subsection.

2.3 Data Preprocessing

When working with text it's important to clean it. This is done to ensure there are no distorations introduced into the model. Over all it will lead to a better accuracy. So, what does text cleaning entail?

- **Uppercase/lowercase:** we wouldn't want "House" and "house" to be considered different words. For this model I applied str.lower() to lowercase all words.
- **Punctuation:** characters such as "?", ".", "!" have been removed.
- **Stop words:** a stop words is a word without any predicting power, words like "the" and "what" can't help us identify a user. I was able to get a list of stop words from the python nltk(Natural Language Toolkit).
- **Lemmatization:** turns a word into its lemma, an example of this would be Singing -> Sing. I have used a Lemmatizer based in WordNet.

Encoding: There are many types of encoding but I've only spent time learning One-hot-encoding, label-encoding and target-encoding. It honestly wasn't required because my only categorical variable is my target. I just simply used label-encoding. Here is what our dataframe looks like now after this text cleaning/encoding.



	Search	Day	Month	Year	Time	user
0	osrs 99 slayer guide	9	10	2021	10:31:16	0
1	windows 11 optimize game	1	10	2021	16:21:12	0
2	dropbox	2	10	2021	16:19:23	0
3	increase max bright dell inspiron 7420	2	10	2021	16:16:51	0
4	pylance	2	10	2021	16:06:21	0

2.4 Downsampling

It is important that our data is balanced, unbalanced data will lead to a model with a misleading accuracy. Let's consider the above histogram showing each user's search counts. Without downsampling/oversampling this data set our model will often predict that user5 is making searches. This will lead to a good accuracy, but is misleading because it doesn't truly understand what is happening but just predicting the most frequent class. To avoid this we are forced to undersample the majority classes or oversample the minority classes. Oversampling is essentially resampling(duplicates) the classes with low amounts of searches. The risk of oversampling is overfitting, usually we use underfitting if we don't have much observations in the dataset. If you duplicate too many observations we are overfitting. But instead of oversampling, I used undersampling or downsampling. Because I collected so much data there is less of a risk of underfitting when downsampling. Overfitting is when our model fits exactly against our training data. It thinks it has a great understanding, with low training error, but when it is tested against our test split, it has high error. Underfitting happens when the model doesn't understand the data, leading to high training error and high test error.

2.5 Train-test Split

We need to set aside a test set to test the quality of the model we are using. For this project I used a random split with %85 in the training set, and %15 in the test set. We will perform hyperparameter tuning with cross validation using the training data, I'll explain this more in the model section. We will then fit the model and use the test data to generate an less biased performance metric.

2.6 Predictive Models

To stay within a reasonable time spent on this project, I've only explored one way of text classification - using TF-IDF bag-of-words. Bag-of-words is a representation that turns text into vectors by counting how many times each word appears in a class. Or in our case how many times a user searches a particular word. TF-IDF(Term Frequency-Inverse Document Frequency) is a form of bag-of-words that increases proportionally to the number of time a word appears in the document, this is offset by the number of documents. This considers the fact that some words are used more than others when searching. We will then use this frequency to predict what user is searching a term. We assume that someone that is searching something once will search the same terms again (which is proven by our results). I applied TF-IDF using an sklearn package called TfidfVectorizer.

3 Results

In this section we will be taking a look at the results of our models using different algorithms. We will also look at the hyperparameters and how they effect our result.

3.1 Models

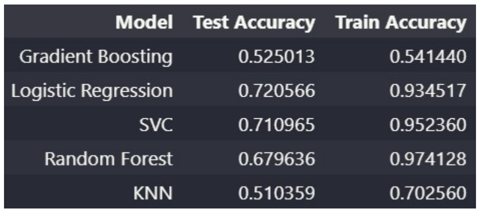
I have tested several machine learning models to see which would fit the data the best. For now like I said I'm sticking to classic machine learning models and avoiding deep learning, not because of lack of data, more because of time constraints, I will address this in the discussion. I have tried the following models:

- Gradient Boosting
- Logistic Regression
- Support Vector Machine(SVM)
- Random Forest
- K Nearest Neighbors (KNN)

3.2 Hyperparameter Tuning

Each one of these models have their own hyperparameters that need to be tuned to find the best result. We first figure out which hyperparameters we want to tune for each model, only considering those that have high influence on the model. If we tuned every hyperparameter these models have it would heavily increase computation time. I then defined a grid of possible values for the hyperparameters and performed a randomized search(100 iterations) using k-fold cross validation. In my case k=3. K-fold cross validation is a procedue used to estimate the skill of the model on new data. Essentially, we shuffle the dataset randomly, split it into 3 groups, 2 groups are training and one is test. We then compute the accuracy, whichever hyperparameters of the 100 random have the best accuracy are our tuned hyperparameter. Using 3-fold cross validation is useful here to reduce bias in our hyperparameters. There has to be a trade off when chosing k=3 and 100 iterations between shorter execution time and higher possible accuracy. Accuracy is the performace metric used in this tuning.

3.3 Model Accuracies



Model	Test Accuracy	Train Accuracy
Gradient Boosting	0.525013	0.541440
Logistic Regression	0.720566	0.934517
SVC	0.710965	0.952360
Random Forest	0.679636	0.974128
KNN	0.510359	0.702560

These are the accuracies along with the training accuracies for each of our models. Overall, we obtain really good accuries for every model outside of gradient boosting. I was very surprised and happy to see the accuries this high, I guess there is less of a similary in people's searches as I thought. We can observe that all models but Gradient Boosting seem to be overfitting since they have an extreamly high training set accuracy but a lower test set accuracy.

4 Discussion

In conclusion we have a pretty awesome model. Lets select Logistic Regression as our prefered model as it has the highest accuracy. With an impressive %72 accuracy we can predict which of 11 users made a search based on only one search! I though I was going to need to predict sessions, for example if a user makes 4 consecutive searches that would be one session of 4 searches.

I would say the objective was definetly proven. It is possible to predict users based solely on their search terms. This project was definetly limited to the amount of users I could get the data for. Ideally I would have more people's search histories and I would want all their searches to have higher counts. I'm sure if all users were like user5 then the model wouldn't require downsampling and would have even better results.

If I had more time I would use word embedding along with a neural network. I was experimenting with a pretained model that defines vector representations of for words. It defines the similarity between words. This one in particular was trained on Wikipedia data. The only way I could find to feed this data to a model is using a neural network. I did dimentionality reduction using tnse on this word embedding, which is taking the embedded words which in my case was 50 dimentions and reducing it to 2 dimentions. Plotting this on a graph shows clear clusers of different types of searches. Two examples of these clusters were linux and video games. We can imagine if I did this embedding for all users, the model would be able to differentiate what clusters of searches different users are doing.